

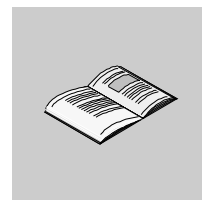
Twido programmable controllers

Software Reference Guide

TWD USE 10AE eng Version 3.2



Table of Contents



	Safety Information	11
	About the Book	15
Part I	Description of Twido Software	17
	At a Glance	17
Chapter 1	Introduction to Twido Software	19
	At a Glance	19
	Introduction to TwidoSoft	20
	Introduction to Twido Languages	21
Chapter 2	Twido Language Objects	25
	At a Glance	25
	Language Object Validation	26
	Bit Objects	27
	Word Objects	29
	Floating point and double word objects	32
	Addressing Bit Objects	36
	Addressing Word Objects	37
	Addressing floating objects	38
	Addressing double word objects	39
	Addressing Inputs/Outputs	40
	Network Addressing	42
	Function Block Objects	43
	Structured Objects	45
	Indexed objects	48
	Symbolizing Objects	50
Chapter 3	User Memory	51
	At a Glance	51
	User Memory Structure	52
	Backup and Restore without Backup Cartridge or Extended Memory	54
	Backup and Restore with a 32K Backup Cartridge	56
	Using the 64K Extended Memory Cartridge	58

Chapter 4	Controller Operating Modes	61
	At a Glance	61
	Cyclic Scan	62
	Periodic Scan	64
	Checking Scan Time	67
	Operating Modes	68
	Dealing with Power Cuts and Power Restoration	69
	Dealing with a warm restart	71
	Dealing with a cold start	73
	Initialization of objects	75
Chapter 5	Event task management	77
	In Brief	77
	Overview of event tasks	78
	Description of different event sources	79
	Event management	80
Part II	Special Functions	81
	At a Glance	81
Chapter 6	Communications	83
	At a Glance	83
	Presentation of the different types of communication	84
	TwidoSoft to Controller communications	86
	Communication between TwidoSoft and a Modem	92
	Remote Link Communications	104
	ASCII Communications	115
	Modbus Communications	126
	Standard Modbus Requests	144
	Transparent Ready Implementation Class (Twido Serial A05, Ethernet A15)	150
	Ethernet TCP/IP Communications Overview	151
	Quick TCP/IP Setup Guide for PC-to-Controller Ethernet Communication	153
	Connecting your Controller to the Network	159
	IP Addressing	160
	Assigning IP Addresses	161
	TCP/IP Setup	165
	IP Address Configure Tab	167
	Marked IP Tab	169
	Time out Tab	171
	Remote Devices Tab	173
	Viewing the Ethernet Configuration	175
	Ethernet Connections Management	176
	Ethernet LED Indicators	178
	TCP Modbus Messaging	180

Chapter 7	Built-In Analog Functions	185
	At a Glance	185
	Analog potentiometer	186
	Analog Channel	188
Chapter 8	Managing Analog Modules	189
	At a Glance	189
	Analog Module Overview	190
	Addressing Analog Inputs and Outputs	191
	Configuring Analog Inputs and Outputs	192
	Analog Module Status Information	198
	Example of Using Analog Modules	199
Chapter 9	Installing the AS-Interface V2 bus	201
	At a Glance	201
	Presentation of the AS-Interface V2 bus	202
	General functional description	203
	Software set up principles	206
	Description of the configuration screen for the AS-Interface bus	207
	Configuration of the AS-Interface bus	209
	Description of the debug screen	215
	Modification of Slave Address	218
	Updating the AS-Interface bus configuration in online mode	220
	Automatic addressing of an AS-Interface V2 slave	225
	How to insert a slave device into an existing AS-Interface V2 configuration	226
	Automatic replacement of a faulty AS-Interface V2 slave	227
	Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus	228
	Programming and diagnostics for the AS-Interface V2 bus	229
	AS-Interface V2 bus interface module operating mode	233
Chapter 10	Installing and Configuring the CANopen Fieldbus	235
	A a Glance	235
10.1	CANopen Fieldbus Overview	237
	At a Glance	237
	CANopen Knowledge Base	238
	About CANopen	239
	CANopen Boot-Up	242
	Process Data Object (PDO) Transmission	245
	Access to Data by Explicit Exchanges (SDO)	247
	"Node Guarding" and "Life Guarding"	248
	Internal Bus Management	250
10.2	Implementing the CANopen Bus	251
	Overview	251
	Overview	252
	Hardware Setup	253

	Configuration Methodology	254
	Declaration of CANopen Master	256
	Network CANopen Slave Declaration	257
	CANopen Objects Mapping	261
	CANopen Objects Linking	265
	CANopen Objects Symbolization.	267
	Addressing PDOs of the CANopen master	269
	Programming and diagnostics for the CANopen fieldbus	270
Chapter 11	Configuring the TwidoPort Ethernet Gateway	277
	At a Glance	277
11.1	Normal Configuration and Connection of TwidoPort	279
	At a Glance	279
	Normal Configuration with TwidoSoft.	280
	BootP Configuration.	285
11.2	TwidoPort's Telnet Configuration.	286
	At a Glance	286
	Introducing Telnet Configuration	287
	Telnet Main Menu	288
	IP/Ethernet Settings.	289
	Serial Parameter Configuration	290
	Configuring the Gateway	291
	Security Configuration	292
	Ethernet Statistics	293
	Serial Statistics	294
	Saving the Configuration	295
	Restoring Default Settings.	296
	Upgrading the TwidoPort Firmware	297
	Forget Your Password and/or IP Configuration?	299
11.3	Communication Features.	300
	At a Glance	300
	Ethernet Features	301
	Modbus/TCP Communications Protocol	302
	Locally Supported Modbus Function Codes	303
Chapter 12	Operator Display Operation	305
	At a Glance	305
	Operator Display	306
	Controller Identification and State Information.	309
	System Objects and Variables.	311
	Serial Port Settings	317
	Time of Day Clock	318
	Real-Time Correction Factor	319

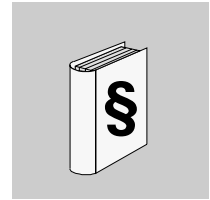
Part III	Description of Twido Languages	321
	At a Glance	321
Chapter 13	Ladder Language	323
	At a Glance	323
	Introduction to Ladder Diagrams	324
	Programming Principles for Ladder Diagrams	326
	Ladder Diagram Blocks	328
	Ladder Language Graphic Elements	330
	Special Ladder Instructions OPEN and SHORT	333
	Programming Advice	334
	Ladder/List Reversibility	337
	Guidelines for Ladder/List Reversibility	338
	Program Documentation	340
Chapter 14	Instruction List Language	343
	At a Glance	343
	Overview of List Programs	344
	Operation of List Instructions	346
	List Language Instructions	347
	Using Parentheses	350
	Stack Instructions (MPS, MRD, MPP)	352
Chapter 15	Grafcet	355
	At a Glance	355
	Description of Grafcet Instructions	356
	Description of Grafcet Program Structure	359
	Actions Associated with Grafcet Steps	362
Part IV	Description of Instructions and Functions	365
	At a Glance	365
Chapter 16	Basic Instructions	367
	At a Glance	367
16.1	Boolean Processing	369
	At a Glance	369
	Boolean Instructions	370
	Understanding the Format for Describing Boolean Instructions	372
	Load Instructions (LD, LDN, LDR, LDF)	374
	Assignment instructions (ST, STN, R, S)	376
	Logical AND Instructions (AND, ANDN, ANDR, ANDF)	378
	Logical OR Instructions (OR, ORN, ORR, ORF)	380
	Exclusive OR, instructions (XOR, XORN, XORR, XORF)	382
	NOT Instruction (N)	384

16.2	Basic Function Blocks	385
	At a Glance	385
	Basic Function Blocks	386
	Standard function blocks programming principles	388
	Timer Function Block (%TMi)	390
	TOF Type of Timer	392
	TON Type of Timer	393
	TP Type of Timer	394
	Programming and Configuring Timers	395
	Up/Down Counter Function Block (%Ci)	398
	Programming and Configuring Counters	402
	Shift Bit Register Function Block (%SBRi)	404
	Step Counter Function Block (%SCi)	406
16.3	Numerical Processing	409
	At a Glance	409
	Introduction to Numerical Instructions	410
	Assignment Instructions	411
	Comparison Instructions	416
	Arithmetic Instructions on Integers	418
	Logic Instructions	422
	Shift Instructions	423
	Conversion Instructions	425
	Single/double word conversion instructions	427
16.4	Program Instructions	428
	At a Glance	428
	END Instructions	429
	NOP Instruction	431
	Jump Instructions	432
	Subroutine Instructions	433
Chapter 17	Advanced Instructions	435
	At a Glance	435
17.1	Advanced Function Blocks	437
	At a Glance	437
	Bit and Word Objects Associated with Advanced Function Blocks	438
	Programming Principles for Advanced Function Blocks	440
	LIFO/FIFO Register Function Block (%Ri)	443
	LIFO Operation	444
	FIFO,operation	445
	Programming and Configuring Registers	446
	Pulse Width Modulation Function Block (%PWM)	448
	Pulse Generator Output Function Block (%PLS)	451
	Drum Controller Function Block (%DR)	454
	Drum Controller Function Block %DRi Operation	455
	Programming and Configuring Drum Controllers	457

	Fast Counter Function Block (%FC)	459
	Very Fast Counter Function Block (%VFC)	462
	Transmitting/Receiving Messages - the Exchange Instruction (EXCH)	476
	Exchange Control Function Block (%MSGx)	477
17.2	Clock Functions	480
	At a Glance	480
	Clock Functions	481
	Schedule Blocks	482
	Time/Date Stamping	485
	Setting the Date and Time	487
17.3	Twido PID Quick Start Guide	490
	At a Glance	490
	Purpose of Document	491
	Step 1 - Configuration of Analog Channels Used for Control	493
	Step 2 - Prerequisites for PID Configuration	495
	Step 3 - Configuring the PID	497
	Step 4 - Initialization of Control Set-Up	504
	Step 5 - Control Set-Up AT + PID	509
	Step 6 - Debugging Adjustments	513
17.4	PID Function	516
	At a Glance	516
	Overview	517
	Principal of the Regulation Loop	518
	Development Methodology of a Regulation Application	519
	Compatibilities and Performances	520
	Detailed characteristics of the PID function	521
	How to access the PID configuration	524
	General tab of PID function	525
	Input tab of the PID	528
	PID tab of PID function	530
	AT tab of PID function	532
	Output tab of the PID	537
	How to access PID debugging	540
	Animation tab of PID function	541
	Trace tab of PID function	543
	PID States and Errors Codes	545
	PID Tuning With Auto-Tuning (AT)	549
	PID parameter adjustment method	557
	Role and influence of PID parameters	559
	Appendix 1: PID Theory Fundamentals	563
	Appendix 2: First-Order With Time Delay Model	565
17.5	Floating point instructions	567
	At a Glance	567
	Arithmetic instructions on floating point	568
	Trigonometric Instructions	572

	Conversion instructions	574
	Integer Conversion Instructions <-> Floating	575
17.6	Instructions on Object Tables	578
	At a Glance	578
	Table summing functions	579
	Table comparison functions	580
	Table search functions	582
	Table search functions for maxi and mini values	584
	Number of occurrences of a value in a table	585
	Table rotate shift function	586
	Table sort function	588
	Floating point table interpolation function	589
	Mean function of the values of a floating point table	594
Chapter 18	System Bits and System Words	595
	At a Glance	595
	System Bits (%S)	596
	System Words (%SW)	604
Glossary		617
Index		629

Safety Information



Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.



The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.



This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER

DANGER indicates a hazardous situation, which **will result** in death, serious injury or equipment damage.

WARNING

WARNING indicates a situation presenting risks liable to **provoke** death, serious injury or equipment damage.

CAUTION

CAUTION indicates a potentially hazardous situation, which, **can result** in personal injury or equipment damage.

PLEASE NOTE

Electrical equipment should be serviced only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material. This document is not intended as an instruction manual for untrained persons. Assembly and installation instructions are provided in the Twido Hardware Reference Manual, TWD USE 10AE.

(c) 2002-2005 Schneider Electric All Rights Reserved

Additional Safety Information

Those responsible for the application, implementation or use of this product must ensure that the necessary design considerations have been incorporated into each application, completely adhering to applicable laws, performance and safety requirements, regulations, codes and standards.

General Warnings and Cautions

 DANGER

HAZARD OF ELECTRIC SHOCK, BURN OR EXPLOSION

Turn off all power before starting installation, removal, wiring, maintenance or inspection of the smart relay system.

Failure to follow this instruction will result in death, serious injury, or equipment damage.

 WARNING

EXPLOSION HAZARD

- Substitution of components may impair suitability for Class I, Div 2 compliance.
- Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.

Failure to follow this instruction can result in death, serious injury, or equipment damage.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Turn power off before installing, removing, wiring, or maintaining.
- This product is not intended for use in safety critical machine functions. Where personnel and or equipment hazards exist, use appropriate safety interlocks.
- Do not disassemble, repair, or modify the modules.
- This controller is designed for use within an enclosure.
- Install the modules in the operating environment conditions described.
- Use the sensor power supply only for supplying power to sensors connected to the module.
- For power line and output circuits, use a fuse designed to Type T standards per IEC60127. The fuse must meet the circuit voltage and current requirements.
Recommended: Littelfuse® 218 Series, 5x20mm time lag (slow blow) fuses.

Failure to follow this instruction can result in death, serious injury, or equipment damage.

Safe Battery Disposal

The TWDLCA•40DRF compact bases use an optional external lithium battery for longer duration of data backup. (Note: The lithium battery is not supplied with the compact bases; you must purchase it separately.)

WARNING

EXPLOSION AND TOXIC HAZARD

- Do not incinerate a lithium battery for it may explode and release toxic substances.
- Do not handle damaged or leaking lithium battery.
- Dead batteries shall be disposed of properly, for improper disposal of unused batteries can cause harm, as well as environmental damage.
- In some areas, the disposal of lithium batteries with household or business trash collection may be prohibited. In any case, it is your responsibility to always conform to local regulations in your area, as regard to battery disposal.

Failure to follow this instruction can result in death, serious injury, or equipment damage.

**Reverse Polarity
Warning**

Reverse-Polarity at Transistor Output is Not Allowed

The TWDLCA•40DRF compact bases transistor outputs cannot withstand any reverse polarity.

 CAUTION

RISK OF REVERSE-POLARITY DAMAGE AT TRANSISTOR OUTPUTS

- Make sure to conform to the polarity markings on the transistor output terminals.
- Use of a reverse polarity can permanently damage or destroy the output circuits.

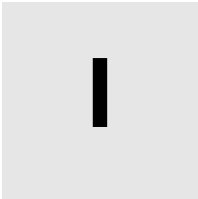
Failure to follow this instruction can result in injury or equipment damage.

About the Book

At a Glance

Document Scope	This is the Software Reference manual for Twido programmable controllers and consists of the following major parts: • Description of the Twido programming software and an introduction to the fundamentals needed to program Twido controllers. • Description of communications, managing analog I/O, installing the AS-Interface bus interface module, the CANopen fieldbus master module and other special functions. • Description of the software languages used to create Twido programs. • Description of instructions and functions of Twido controllers.
Validity Note	The information in this manual is applicable only for Twido programmable controllers.
Product Related Warnings	Schneider Electric assumes no responsibility for any errors that appear in this document. No part of this document may be reproduced in any form or means, including electronic, without prior written permission of Schneider Electric.
User Comments	We welcome your comments about this document. You can reach us by e-mail at techpub@schneider-electric.com

Description of Twido Software



At a Glance

Subject of this Part

This part provides an introduction to the software languages and the basic information required to create control programs for Twido programmable controllers.

What's in this Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	Introduction to Twido Software	19
2	Twido Language Objects	25
3	User Memory	51
4	Controller Operating Modes	61
5	Event task management	77

Introduction to Twido Software

1

At a Glance

Subject of this Chapter

This chapter provides a brief introduction to TwidoSoft, the programming and configuration software for Twido controllers, and to the List, Ladder, and Grafcet programming languages.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Introduction to TwidoSoft	20
Introduction to Twido Languages	21

Introduction to TwidoSoft

Introduction

TwidoSoft is a graphical development environment for creating, configuring, and maintaining applications for Twido programmable controllers. TwidoSoft allows you to create programs with different types of languages (See *Twido Languages*, p. 21), and then transfer the application to run on a controller.

TwidoSoft

TwidoSoft is a 32-bit Windows-based program for a personal computer (PC) running Microsoft Windows 98 Second Edition, Microsoft Windows 2000 Professional or Microsoft Windows XP operating systems.

The main software features of TwidoSoft:

- Standard Windows user interface
- Program and configure Twido controllers
- Controller communication and control

Note: The Controller-PC link uses the TCP/IP protocol. It is essential for this protocol to be installed on the PC.

Minimum configuration

The minimum configuration for using TwidoSoft is:

- Pentium 300MHz,
- 128 Mb of RAM,
- 40 Mb of available space on the hard disk.

Introduction to Twido Languages

Introduction

A programmable controller reads inputs, writes to outputs, and solves logic based on a control program. Creating a control program for a Twido controller consists of writing a series of instructions in one of the Twido programming languages.

Twido Languages

The following languages can be used to create Twido control programs:

- **Instruction List Language:**
An Instruction List program is a series of logical expressions written as a sequence of Boolean instructions.
- **Ladder Diagrams:**
A Ladder diagram is a graphical means of displaying a logical expression.
- **Grafcet Language:**
Grafcet language is made up of a series of steps and transitions. Twido supports the use of Grafcet list instructions, but not graphical Grafcet.

You can use a personal computer (PC) to create and edit Twido control programs using these programming languages.

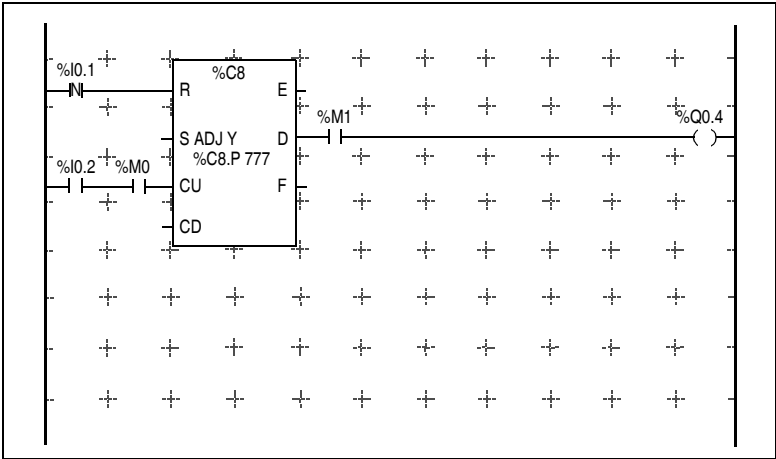
A List/Ladder reversibility feature allows you to conveniently reverse a program from Ladder to List and from List to Ladder.

Instruction List Language

A program written in Instruction List language consists of a series of instructions executed sequentially by the controller. The following is an example of a List program.

```
0  BLK  %C8
1  LDF  %I0.1
2  R
3  LD   %I0.2
4  AND  %M0
5  CU
6  OUT_BLK
7  LD   D
8  AND  %M1
9  ST   %Q0.4
10 END_BLK
```

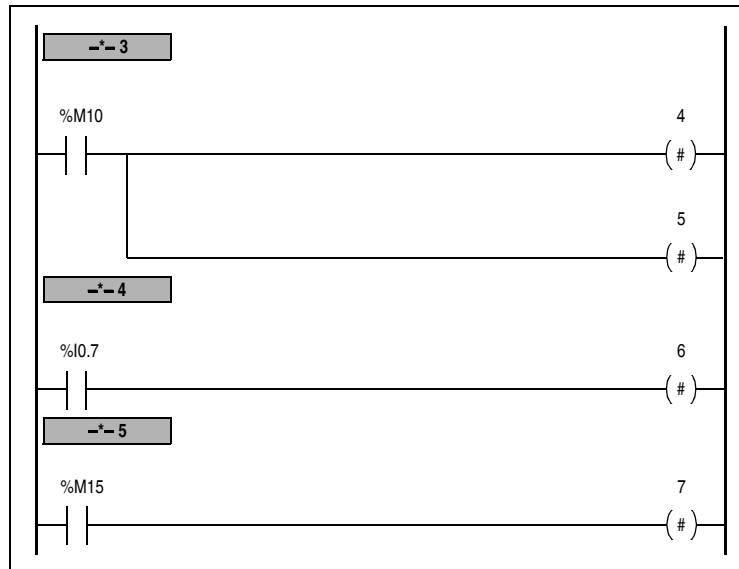
Ladder Diagrams Ladder diagrams are similar to relay logic diagrams that represent relay control circuits. Graphic elements such as coils, contacts, and blocks represent instructions. The following is an example of a Ladder diagram.



**Grafcet
Language**

The Grafcet analytical method divides any sequential control system into a series of steps, with which actions, transitions, and conditions are associated. The following illustration shows examples of Grafcet instructions in List and Ladder programs respectively.

0	-*-	3
1	LD	%M10
2	#	4
3	#	5
4	-*-	4
5	LD	%I0.7
6	#	6
7	-*-	5
8	LD	%M15
9	#	7
10	...	



Twido Language Objects

2

At a Glance

Subject of this Chapter

This chapter provides details about the language objects used for programming Twido controllers.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Language Object Validation	26
Bit Objects	27
Word Objects	29
Floating point and double word objects	32
Addressing Bit Objects	36
Addressing Word Objects	37
Addressing floating objects	38
Addressing double word objects	39
Addressing Inputs/Outputs	40
Network Addressing	42
Function Block Objects	43
Structured Objects	45
Indexed objects	48
Symbolizing Objects	50

Language Object Validation

Introduction

Word and bit objects are valid if they have been allocated memory space in the controller. To do this, they must be used in the application before downloaded to the controller.

Example

The range of valid objects is from zero to the maximum reference for that object type. For example, if your application's maximum references for memory words is %MW9, then %MW0 through %MW9 are allocated space. %MW10 in this example is not valid and can not be accessed either internally or externally.

Bit Objects

Introduction

Bit objects are bit-type software variables that can be used as operands and tested by Boolean instructions. The following is a list of bit objects:

- I/O bits
- Internal bits (memory bits)
- System bits
- Step bits
- Bits extracted from words

List of Operand Bits

The following table lists and describes all of the main bit objects that are used as operands in Boolean instructions.

Type	Description	Address or value	Maximum number	Write access (1)
Immediate values	0 or 1 (False or True)	0 or 1	-	-
Inputs Outputs	These bits are the "logical images" of the electrical states of the I/O. They are stored in data memory and updated during each scan of the program logic.	%Ix.y.z (2) %Qx.y.z (2)	Note (4)	No Yes
AS-Interface Inputs Outputs	These bits are the "logical images" of the electrical states of the I/O. They are stored in data memory and updated during each scan of the program logic.	%IAx.y.z %QAx.y.z	Note (5)	No Yes
Internal (Memory)	Internal bits are internal memory areas used to store intermediary values while a program is running. Note: Unused I/O bits can not be used as internal bits.	%Mi	128 TWDLC•A10DRF, TWDLC•A16DRF 256 All other controllers	Yes
System	System bits %S0 to %S127 monitor the correct operation of the controller and the correct running of the application program.	%Si	128	According to i
Function blocks	The function block bits correspond to the outputs of the function blocks. These outputs may be either directly connected or used as an object.	%Tmi.Q, %Ci.P, and so on.	Note (4)	No (3)

Type	Description	Address or value	Maximum number	Write access (1)
Reversible function blocks	Function blocks programmed using reversible programming instructions BLK, OUT_BLK, and END_BLK.	E, D, F, Q, TH0, TH1	Note (4)	No
Word extracts	One of the 16 bits in some words can be extracted as operand bits.	Variable	Variable	Variable
Grafcet steps	Bits %X1 to %Xi are associated with Grafcet steps. Step bit Xi is set to 1 when the corresponding step is active, and set to 0 when the step is deactivated.	%X21	62 TWDLC•A10DRF, TWDLC•A16 DRF 96 TWDLC•A24DRF, TWDLCA•40DRF and Modular controllers	Yes

Legends:

1. Written by the program or by using the Animation Tables Editor.
2. See I/O Addressing.
3. Except for %SBRi.j and %SCi.j, these bits can be read and written.
4. Number is determined by controller model.
5. Where, x = address of the expansion module (0..7); y = AS-Interface address (0A..31B); z = channel number (0..3). (See *Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus*, p. 228.)

Word Objects

Introduction

Word objects that are addressed in the form of 16-bit words that are stored in data memory and can contain an integer value between -32768 and 32767 (except for the fast counter function block which is between 0 and 65535).

Examples of word objects:

- Immediate values
- Internal words (%MWi) (memory words)
- Constant words (%KWi)
- I/O exchange words (%IWi, %QWi%)
- AS-Interface analog I/O words (IWAi, %QWAI)
- System words (%SWi)
- Function blocks (configuration and/or runtime data)

Word Formats

The contents of the words or values are stored in user memory in 16-bit binary code (two's complement) using the following convention:

F E D C B A 9 8 7 6 5 4 3 2 1 0																Bit position
0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	1	Bit state
+	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	Bit value

In signed binary notation, bit 15 is allocated by convention to the sign of the coded value:

- Bit 15 is set to 0: the content of the word is a positive value.
- Bit 15 is set to 1: the content of the word is a negative value (negative values are expressed in two's complement logic).

Words and immediate values can be entered or retrieved in the following format:

- Decimal
Min.: -32768, Max.: 32767 (1579, for example)
- Hexadecimal
Min.: 16#0000, Max.: 16#FFFF (for example, 16#A536)
Alternate syntax: #A536

Descriptions of Word Objects

The following table describes the word objects.

Words	Description	Address or value	Maximum number	Write access (1)
Immediate values	These are integer values that are in the same format as the 16-bit words, which enables values to be assigned to these words.		-	No
	Base 10	-32768 to 32767		
	Base 16	16#0000 to 16#FFFF		
Internal (Memory)	Used as "working" words to store values during operation in data memory. Words %MW0 to %MW255 are read or written directly by the program.	%MWi	3000	Yes
Constants	Store constants or alphanumeric messages. Their content can only be written or modified by using TwidoSoft during configuration. Constant words %KW0 through %KW63 are read-only by the program.	%KWi	256	Yes, only by using TwidoSoft
System	These 16-bit words have several functions: ● Provide access to data coming directly from the controller by reading %SWi words.) ● Perform operations on the application (for example, adjusting schedule blocks).	%SWi	128	According to i
Function blocks	These words correspond to current parameters or values of function blocks.	%TM2.P, %Ci.P, etc.		Yes
Network exchange words	Assigned to controllers connected as Remote Links. These words are used for communication between controllers:			
	Network Input	%INWi.j	4 per remote link	No
	Network Output	%QNWj.i	4 per remote link	Yes
Analog I/O words	Assigned to analog inputs and outputs of AS-Interface slave modules.			
	Analog Inputs	%IWAx.y.z	Note (3)	No
	Analog Outputs	%QWAx.y.z	Note (3)	Yes

Words	Description	Address or value	Maximum number	Write access (1)
Extracted bits	It is possible to extract one of the 16 bits from the following words:			
	Internal	%MWi:Xk	1500	Yes
	System	%SWi:Xk	128	Depends on i
	Constants	%KWi:Xk	64	No
	Input	%IWi.j:Xk	Note (2)	No
	Output	%QWi.j:Xk	Note (2)	Yes
	AS-Interface Slave Input	%IWAx.y.z:Xk	Note (2)	No
	AS-Interface Slave Output	%QWAx.y.z:Xk	Note (2)	Yes
	Network Input	%INWi.j:Xk	Note (2)	No
	Network Output	%QNWj.j:Xk	Note (2)	Yes

Note:

1. Written by the program or by using the Animation Tables Editor.
2. Number is determined by the configuration.
3. Where, x = address of the expansion module (0..7); y = AS-Interface address (0A..31B); z = channel number (0..3). (See *Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus*, p. 228.)

Floating point and double word objects

Introduction

TwidoSoft allows you to perform operations on floating point and double integer word objects.

A floating point is a mathematical argument which has a decimal in its expression (examples: 3.4E+38, 2.3 or 1.0).

A double integer word consists of 4 bytes stored in data memory and containing a value between -2147483648 and +2147483647.

Floating Point Format and Value

The floating format used is the standard IEEE STD 734-1985 (equivalent IEC 559). The length of the words is 32 bits, which corresponds to the single decimal point floating numbers.

Table showing the format of a floating point value:

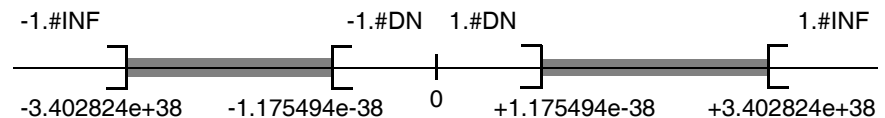
Bit 31	Bits {30...23}	Bits {22...0}
S	Exponent	Fractional part

The value as expressed in the above format is determined by the following equation:

$$32\text{-bit Floating Value} = (-1)^S * 2^{(\text{Exponent} - 127)} * 1.\text{Fractional part}$$

Floating values can be represented with or without an exponent; but they must always have a decimal point (floating point).

Floating values range from -3.402824e+38 and -1.175494e-38 to 1.175494e-38 and 3.402824e+38 (grayed out values on the diagram). They also have the value 0, written 0.0



When a calculation result is:

- Less than -3.402824e+38, the symbol -1.#INF (for -infinite) is displayed,
- Greater than +3.402824e+38, the symbol 1.#INF (for +infinite) is displayed,
- Between -1.175494e-38 and 1.175494e-38, it is rounded off to 0.0. A value within these limits cannot be entered as a floating value.
- Indefinite (for example the square root of a negative number) the symbol 1.#NAN or -1.#NAN is displayed.

Representation precision is 2-24. To display floating point numbers, it is unnecessary to display more than 6 digits after the decimal point.

Note:

- the value "1285" is interpreted as a whole value; in order for it to be recognized as a floating point value, it must be written thus: "1285.0"

Limit range of Arithmetic Functions on Floating Point

The following table describes the limit range of arithmetic functions on floating point objects

Arithmetic Function		Limit range and invalid operations	
Type	Syntax	#QNAN (Invalid)	#INF (Infinite)
Square root of an operand	SQRT(x)	$x < 0$	$x > 1.7E38$
Power of an integer by a real EXPT(%MF,%MW)	EXPT(y, x) (where: $x^y = \%MW^{\%MF}$)	$x < 0$	$y.\ln(x) > 88$
Base 10 logarithm	LOG(x)	$x \leq 0$	$x > 2.4E38$
Natural logarithm	LN(x)	$x \leq 0$	$x > 1.65E38$
Natural exponential	EXP(x)	$x < 0$	$x > 88.0$

Hardware compatibility

Floating point and double word operations are not supported by all Twido controllers.

The following table shows hardware compatibility:

Twido controller	Double words supported	Floating points supported
TWDLMDA40DUK	Yes	Yes
TWDLMDA40DTK	Yes	Yes
TWDLMDA20DUK	Yes	No
TWDLMDA20DTK	Yes	No
TWDLMDA20DRT	Yes	Yes
TWDLCA•40DRF	Yes	Yes
TWDLCA•A24DRF	Yes	No
TWDLCA•A16DRF	Yes	No
TWDLCA•A10DRF	No	No

Validity Check

When the result is not within the valid range, the system bit %S18 is set to 1. The status word %SW17 bits indicate the cause of an error in a floating operation: Different bits of the word %SW17:

%SW17:X0	Invalid operation, result is not a number (1.#NAN or -1.#NAN)
%SW17:X1	Reserved
%SW17:X2	Divided by 0, result is infinite (-1.#INF or 1.#INF)
%SW17:X3	Result greater in absolute value than +3.402824e+38, result is infinite (-1.#INF or 1.#INF)
%SW17:X4 to X15	Reserved

This word is reset to 0 by the system on cold start, and also by the program for re-usage purposes.

Description of Floating Point and Double Word Objects

The following table describes floating point and double word objects:

Type of object	Description	Address	Maximum number	Write access	Indexed form
Immediate values	Integers or decimal numbers with identical format to 32 bit objects.	-	[-]	No	-
Internal floating point	Objects used to store values during operation in data memory.	%MFi	1500	Yes	%MFi[index]
Internal double word		%MDi	1500	Yes	%MDi[index]
Floating constant value	Used to store constants.	%KFi	128	Yes, only using TwidoSoft	%KFi[index]
Double constant		%KDi	128	Yes, only using TwidoSoft	%KDi[index]

Possibility of Overlap between Objects

Single, double length and floating words are stored in the data space in one memory zone. Thus, the floating word %MF_i and the double word %MD_i correspond to the single length words %MW_i and %MW_{i+1} (the word %MW_i containing the least significant bits and the word %MW_{i+1} the most significant bits of the word %MF_i). The following table shows how floating and double internal words overlap:

Floating and Double	Odd address	Internal words
%MF0 / %MD0		%MW0
	%MF1 / %MD1	%MW1
%MF2 / %MD2		%MW2
	%MF3 / %MD3	%MW3
%MF4 / %MD4		%MW4
	...	%MW5
...		...
	%MF _i / %MD _i	%MW _i
%MF _{i+1} / %MD _{i+1}		%MW _{i+1}

The following table shows how floating and double constants overlap:

Floating and Double	Odd address	Internal words
%KF0 / %KD0		%KW0
	%KF1 / %KD1	%KW1
%KF2 / %KD2		%KW2
	%KF3 / %KD3	%KW3
%KF4 / %KD4		%KW4
	...	%KW5
...		...
	%kF _i / %kD _i	%KW _i
%KF _{i+1} / %KD _{i+1}		%KW _{i+1}

Example:

%MF0 corresponds to %MW0 and %MW1. %KF543 corresponds to %KW543 and %KW544.

Addressing Bit Objects

Syntax

Use the following format to address internal, system, and step bit objects:

%	M, S, or X	i
Symbol	Object type	Number

Description

The following table describes the elements in the addressing format.

Group	Item	Description
Symbol	%	The percent symbol always precedes a software variable.
Type of object	M	Internal bits store intermediary values while a program is running.
	S	System bits provide status and control information for the controller.
	X	Step bits provide status of step activities.
Number	i	The maximum number value depends on the number of objects configured.

Examples of bit object addressing:

- %M25 = internal bit number 25
- %S20 = system bit number 20
- %X6 = step bit number 6

Bit Objects Extracted from Words

TwidoSoft is used to extract one of the 16 bits from words. The address of the word is then completed by the bit row extracted according to the following syntax:

WORD	: X	k
Word address		Position k = 0 - 15 bit rank in the word address.

Examples:

- %MW5:X6 = bit number 6 of internal word %MW5
- %QW5.1:X10 = bit number 10 of output word %QW5.1

Addressing Word Objects

Introduction

Addressing word objects, except for input/output addressing (see *Addressing Inputs/Outputs*, p. 40) and function blocks (see *Function Block Objects*, p. 43), follows the format described below.

Syntax

Use the following format to address internal, constant and system words:

%	M, K or S	W	i
Symbol	Object type	Format	Number

Description

The following table describes the elements in the addressing format.

Group	Item	Description
Symbol	%	The percent symbol always precedes an internal address.
Type of object	M	Internal words store intermediary values while a program is running.
	K	Constant words store constant values or alphanumeric messages. Their content can only be written or modified by using TwidoSoft.
	S	System words provide status and control information for the controller.
Syntax	W	16-bit word.
Number	i	The maximum number value depends on the number of objects configured.

Examples of word object addressing:

- %MW15 = internal word number 15
- %KW26 = constant word number 26
- %SW30 = system word number 30

Addressing floating objects

Introduction Addressing floating objects, except for input/output addressing (see *Addressing Inputs/Outputs*, p. 40) and function blocks (see *Function Block Objects*, p. 43), follows the format described below.

Syntax Use the following format to address internal and constant floating objects:

%	M or K	F	i
Symbol	Type of object	Syntax	Number

Description The following table describes the elements in the addressing format.

Group	Item	Description
Symbol	%	The percent symbol always precedes an internal address.
Type of object	M	Internal floating objects store intermediary values while a program is running.
	K	Floating constants are used to store constant values. Their content can only be written or modified by using TwidoSoft.
Syntax	F	32 bit object.
Number	i	The maximum number value depends on the number of objects configured.

- Examples of floating object addresses:**
- %MF15 = internal floating object number 15
 - %KF26 = constant floating object number 26

Addressing double word objects

Introduction

Addressing double word objects, except for input/output addressing (see *Addressing Inputs/Outputs*, p. 40) and function blocks (see *Function Block Objects*, p. 43), follows the format described below.

Syntax

Use the following format to address internal and constant double words:

%	M or K	D	i
Symbol	Type of object	Syntax	Number

Description

The following table describes the elements in the addressing format.

Group	Item	Description
Symbol	%	The percent symbol always precedes an internal address.
Type of object	M	Internal double words are used to store intermediary values while a program is running.
	K	Constant double words store constant values or alphanumeric messages. Their content can only be written or modified by using TwidoSoft.
Syntax	D	32 bit double word.
Number	i	The maximum number value depends on the number of objects configured.

Examples of double word object addressing:

- %MD15 = internal double word number 15
- %KD26 = constant double word number 26

Addressing Inputs/Outputs

Introduction

Each input/output (I/O) point in a Twido configuration has a unique address: For example, the address "%I0.0.4" is assigned to input 4 of a controller.

I/O addresses can be assigned for the following hardware:

- Controller configured as Remote Link Master
- Controller configured as Remote I/O
- Expansion I/O modules

The TWDNOI10M3 AS-Interface bus interface module and the TWDNCO1M CANopen fieldbus module each uses its own special address system for addressing the I/Os of slave devices connected to its bus:

- For TWDNOI10M3, see *Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus*, p. 228.
- For TWDNCO1M, see *Addressing PDOs of the CANopen master*, p. 269.

Multiple References to an Output or Coil

In a program, you can have multiple references to a single output or coil. Only the result of the last one solved is updated on the hardware outputs. For example, %Q0.0.0 can be used more than once in a program, and there will not be a warning for multiple occurrences. So it is important to confirm only the equation that will give the required status of the output.

CAUTION

UNINTENDED OPERATION

No duplicate output checking or warnings are provided. Review the use of the outputs or coils before making changes to them in your application.

Failure to follow this instruction can result in injury or equipment damage.

Format

Use the following format to address inputs/outputs.

%	I, Q	x	.	y	.	z
Symbol	Object type	Controller position	point	I/O type	point	Channel number

Use the following format to address inputs/output exchange words.

%	I, Q	W	x	.	y
Symbol	Object type	Format	Controller position	point	I/O Type

Description

The table below describes the I/O addressing format.

Group	Item	Value	Description
Symbol	%	-	The percent symbol always precedes an internal address.
Object type	I	-	Input. The "logical image" of the electrical state of a controller or expansion I/O module input.
	Q	-	Output. The "logical image" of the electrical state of a controller or expansion I/O module output.
Controller position	x	0 1 - 7	Master controller (Remote Link master). Remote controller (Remote Link slave).
I/O Type	y	0 1 - 7	Base I/O (local I/O on controller). Expansion I/O modules.
Channel Number	z	0 - 31	I/O channel number on controller or expansion I/O module. Number of available I/O points depends on controller model or type of expansion I/O module.

Examples

The table below shows some examples of I/O addressing.

I/O object	Description
%I0.0.5	Input point number 5 on the base controller (local I/O).
%Q0.3.4	Output point number 4 on the expansion I/O module at address 3 for the controller base (expansion I/O).
%I0.0.3	Input point number 3 on base controller.
%I3.0.1	Input point number 1 on remote I/O controller at address 3 of the remote link.
%I0.3.2	Input point number 2 on the expansion I/O module at address 3 for the controller base.

Network Addressing

Introduction Application data is exchanged between peer controllers and the master controller on a Twido Remote Link network by using the network words %INW and %QNW. See *Communications* , p. 83 for more details.

Format Use the following format for network addressing.

%	IN,QN	W	x	.	j
Symbol	Object type	Format	Controller position	point	Word

Description of Format The table below describes the network addressing format.

Group	Element	Value	Description
Symbol	%	-	The percent symbol always precedes an internal address.
Object type	IN	-	Network input word. Data transfer from master to peer.
	QN	-	Network output word. Data transfer from peer to master.
Format	W	-	A16-bit word.
Controller position	x	0	Master controller (Remote Link master).
		1 - 7	Remote controller (Remote Link slave).
Word	j	0 - 3	Each peer controller uses from one to four words to exchange data with the master controller.

Examples The table below shows some examples of networking addressing.

Network object	Description
%INW3.1	Network word number 1 of remote controller number 3.
%QNW0.3	Network word number 3 of the base controller.

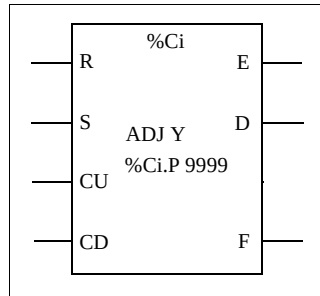
Function Block Objects

Introduction

Function blocks provide bit objects and specific words that can be accessed by the program.

Example of a Function Block

The following illustration shows a counter function block.



Up/down counter block

Bit Objects

Bit objects correspond to the block outputs. These bits can be accessed by Boolean test instructions using either of the following methods:

- Directly (for example, LD E) if they are wired to the block in reversible programming (see *Standard function blocks programming principles*, p. 388).
- By specifying the block type (for example, LD %Ci.E).

Inputs can be accessed in the form of instructions.

Word Objects

Word objects correspond to specified parameters and values as follows:

- Block configuration parameters: some parameters are accessible by the program (for example, pre-selection parameters), and some are inaccessible by the program (for example, time base).
- Current values: for example, %Ci.V, the current count value.

Word Objects

Double word objects increase the computational capability of your Twido controller while executing system functions, such as fast counters (%FC), very fast counters (%VFC) and pulse generators (%PLS).

Addressing of 32-bit double word objects used with function blocks simply consists in appending the original syntax of the standard word objects with the "D" character. The following example, shows how to address the current value of a fast counter in standard format and in double word format:

- %FCi.V is current value of the fast counter in standard format.
- %FCi.VD is the current value of the fast counter in double word format.

Note: Double word objects are not supported by all Twido controllers. Refer to *Hardware compatibility*, p. 33 to find out if your Twido controller can accommodate double words.

Objects Accessible by the Program

See the following appropriate sections for a list of objects that are accessible by the program.

- For Basic Function Blocks, see *Basic Function Blocks*, p. 386.
 - For Advanced Function Blocks, see *Bit and Word Objects Associated with Advanced Function Blocks*, p. 438.
-

Structured Objects

Introduction

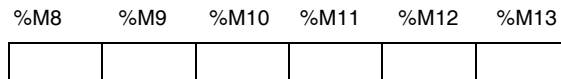
Structured objects are combinations of adjacent objects. Twido supports the following types of structured objects:

- Bit Strings
- Tables of words
- Tables of double words
- Tables of floating words

Bit Strings

Bit strings are a series of adjacent object bits of the same type and of a defined length (L).

Example:Bit string %M8:6



Note: %M8:6 is acceptable (8 is a multiple of 8), while %M10:16 is unacceptable (10 is not a multiple of 8).

Bit strings can be used with the Assignment instruction (see *Assignment Instructions*, p. 411).

Available Types of Bits

Available types of bits for bit strings:

Type	Address	Maximum size	Write access
Discrete input bits	%I0.0:L or %I1.0:L (1)	$0 < L < 17$	No
Discrete output bits	%Q0.0:L or %Q1.0:L (1)	$0 < L < 17$	Yes
System bits	%Si:L with i multiple of 8	$0 < L < 17$ and $i+L \leq 128$	Depending on i
Grafcet Step bits	%Xi:L with i multiple of 8	$0 < L < 17$ and $i+L \leq 95$ (2)	Yes (by program)
Internal bits	%Mi:L with i multiple of 8	$0 < L < 17$ and $i+L \leq 256$ (3)	Yes

Key:

1. Only I/O bits 0 to 16 can be read in bit string. For controllers with 24 inputs and 32 I/O modules, bits over 16 cannot be read in bit string.
2. Maximum of $i+L$ for TWWDLCAA10DRF and TWDLCAA16DRF is 62
3. Maximum of $i+L$ for TWWDLCAA10DRF and TWDLCAA16DRF is 128

Tables of words

Word tables are a series of adjacent words of the same type and of a defined length (L).

Example: Word table %KW10:7

%KW10	16 bits
%KW16	

Word tables can be used with the Assignment instruction (see *Assignment Instructions*, p. 411).**Available Types of Words**

Available types of words for word tables:

Type	Address	Maximum size	Write access
Internal words	%MWi:L	$0 < L < 256$ and $i+L < 3000$	Yes
Constant words	%KWi:L	$0 < L < 256$ and $i+L < 256$	No
System Words	%SWi:L	$0 < L$ and $i+L < 128$	Depending on i

Tables of double words

Double word tables are a series of adjacent words of the same type and of a defined length (L).

Example: Double word table %KD10:7

%KD10	32 Bit
%KD22	

Double word tables can be used with the Assignment instruction (see *Assignment Instructions*, p. 411).

Available Types of Double Words

Available types of words for double word tables:

Type	Address	Maximum size	Write access
Internal words	%MDi:L	0<L<256 and i+L< 3000	Yes
Constant words	%KDi:L	0<L and i+L< 256	No

Tables of floating words

Floating word tables are a series of adjacent words of the same type and of a defined length (L).

Example: Floating point table %KF10:7

%KF10	32 Bit
%KF22	

Floating point tables can be used with the Assignment instruction (see Advanced instructions).

Types of Floating Words Available

Available types of words for floating word tables:

Type	Address	Maximum size	Write access
Internal words	%MFi:L	0<L<256 and i+L< 3000	Yes
Constant words	%KFi:L	0<L and i+L<256	No

Indexed objects

Introduction

An indexed word is a single or double word or floating point with an indexed object address. There are two types of object addressing:

- Direct addressing
 - Indexed addressing
-

Direct Addressing

A direct address of an object is set and defined when a program is written.

Example: %M26 is an internal bit with the direct address 26.

Indexed Addressing

An indexed address of an object provides a method of modifying the address of an object by adding an index to the direct address of an object. The content of the index is added to the object's direct address. The index is defined by an internal word %MWi. The number of "index words" is unlimited.

Example: %MW108[%MW2] is a word with an address consisting of the direct address 108 plus the contents of word %MW2.

If word %MW2 has a value of 12, writing to %MW108[%MW2] is equivalent to writing to %MW120 (108 plus 12).

Objects Available for Indexed Addressing

The following are the available types of objects for indexed addressing.

Type	Address	Maximum size	Write access
Internal words	%MWi[MWj]	$0 \leq i + \%MWj < 3000$	Yes
Constant words	%KWi[%MWj]	$0 \leq i + \%MWj < 256$	No
Internal double words	%MDi[MWj]	$0 \leq i + \%MWj < 2999$	Yes
Double constant words	%KDi[%MWj]	$0 \leq i + \%MWj < 255$	No
Internal floating points	%MFi[MWj]	$0 \leq i + \%MWj < 2999$	Yes
Constant floating points	%KFi[%MWj]	$0 \leq i + \%MWj < 255$	No

Indexed objects can be used with the assignment instructions (see *Assignment Instructions*, p. 411 for single and double words) and in comparison instructions (see *Comparison Instructions*, p. 416 for single and double words). This type of addressing enables series of objects of the same type (such as internal words and constants) to be scanned in succession, by modifying the content of the index object via the program.

**Index Overflow
system bit %S20**

An overflow of the index occurs when the address of an indexed object exceeds the limits of the memory zone containing the same type of object. In summary:

- The object address plus the content of the index is less than 0.
- The object address plus the content of the index is greater than the largest word directly referenced in the application. The maximum number is 2999 (for words %MWi) or 255 (for words %KWi).

In the event of an index overflow, the system sets system bit %S20 to 1 and the object is assigned an index value of 0.

Note: The user is responsible for monitoring any overflow. Bit %S20 must be read by the user program for possible processing. The user must confirm that it is reset to 0.

%S20 (initial status = 0):

- On index overflow: set to 1 by the system.
 - Acknowledgment of overflow: set to 0 by the user, after modifying the index.
-

Symbolizing Objects

Introduction

You can use Symbols to address Twido software language objects by name or customized mnemonics. Using symbols allows for quick examination and analysis of program logic, and greatly simplifies the development and testing of an application.

Example

For example, WASH_END is a symbol that could be used to identify a timer function block that represents the end of a wash cycle. Recalling the purpose of this name should be easier than trying to remember the role of a program address such as %TM3.

Guidelines for Defining Symbols

The following are guidelines for defining symbols:

- A maximum of 32 characters.
- Letters (A-Z), numbers (0 -9), or underscores (_).
- First character must be an alphabetical or accented character. You can not use the percentile sign (%).
- Do not use spaces or special characters.
- Not case-sensitive. For example, Pump1 and PUMP1 are the same symbol and can only be used once in an application.

Editing Symbols

Symbols are defined and associated with language objects in the Symbol Editor. Symbols and their comments are stored with the application on the PC hard drive, but are not stored on the controller. Therefore, they can not be transferred with the application to the controller.

User Memory



At a Glance

Subject of this Chapter

This chapter describes the structure and usage of Twido user memory.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
User Memory Structure	52
Backup and Restore without Backup Cartridge or Extended Memory	54
Backup and Restore with a 32K Backup Cartridge	56
Using the 64K Extended Memory Cartridge	58

User Memory Structure

Introduction	The controller memory accessible to your application is divided into two distinct sets: ● Bit values ● Word values (16-bit signed values) and double word values (32-bit signed values)
Bit Memory	The bit memory is located in the controller's built-in RAM. It contains the map of 128 bit objects.
Word Memory	The word memory (16 bits) supports: ● Dynamic words: runtime memory (stored in RAM only). ● Memory words (%MW) and double words (%MD): dynamic system data and system data. ● Program: descriptors and executable code for tasks. ● Configuration data: constant words, initial values, and input/output configuration.
Memory Storage Types	The following are the different types of memory storage for Twido controllers. ● Random Access Memory. Internal volatile memory: Contains dynamic words, memory words, program and configuration data. ● EEPROM An integrated 32KB EEPROM that provides internal program and data backup. Protects program from corruption due to battery failure or a power outage lasting longer than 30 days. Contains program and configuration data. Holds a maximum of 512 memory words. Program is not backed up here If a 64K extended memory cartridge is being used and Twido has been configured to accept the 64K extended memory cartridge. ● Erase 32K backup cartridge An optional external cartridge used to save a program and transfer that program to other Twido controllers. Can be used to update the program in controller RAM. Contains program and constants, but no memory words. ● 64K extended memory cartridge An optional external cartridge that stores a program up to 64K. Must remain plugged into the controller as long as that program is being used.
Saving Memory	Your controller's program and memory words can be saved in the following: ● RAM (for up to 30 days with good battery) ● EEPROM (maximum of 32 KB) Transferring the program from the EEPROM memory to the RAM memory is done automatically when the program is lost in RAM (or if there is no battery). Manual transfer can also be performed using TwidoSoft.

Memory Configurations

The following tables describe the types of memory configurations possible with Twido compact and modulare controllers.

Memory Type	Compact Controllers				
	10DRF	16DRF	24DRF	40DRF (32k)	40DRF** (64k)
Internal RAM Mem 1*	10KB	10KB	10KB	10KB	10KB
External RAM Mem 2*		16KB	32KB	32KB	64KB
Internal EEPROM	8KB	16KB	32KB	32KB	32KB***
External EEPROM	32KB	32KB	32KB	32KB	64KB
Maximum program size	8KB	16KB	32KB	32KB	64KB
Maximum external backup	8KB	16KB	32KB	32KB	64KB

Memory Type	Modular Controllers		
	20DUK 20DTK	20DRT 40DUK 40DTK (32k)	20DRT 40DUK 40DTK** (64k)
Internal RAM Mem 1*	10KB	10KB	10KB
External RAM Mem 2*	32KB	32KB	64KB
Internal EEPROM	32KB	32KB	32KB***
External EEPROM	32KB	32KB	64KB
Maximum program size	32KB	32KB	64KB
Maximum external backup	32KB	32KB	64KB

(*) Mem 1 and Mem 2 in memory usage.

(**) in this case the 64KB cartridge must be installed on the Twido and declared in the configuration, if it has not already been declared,

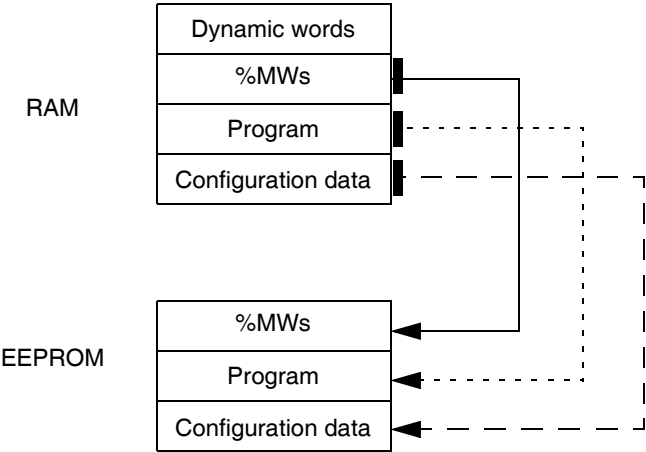
(***) reserved for backup of the first 512 %MW words or the first 256 %MD double words.

Backup and Restore without Backup Cartridge or Extended Memory

Introduction The following information details backup and restore memory functions in modular and compact controllers without a backup cartridge or extended memory plugged in.

At a Glance Twido programs, memory words and configuration data can be backed up using the controllers internal EEPROM. Because saving a program to the internal EEPROM clears any previously backed up memory words, the program must be backed up first, then the configured memory words. Dynamic data can be stored in memory words then backed up to the EEPROM. If there is no program saved to the internal EEPROM you cannot save memory words to it.

Memory Structure Here is a diagram of a controller’s memory structure. The arrows show what can be backed up to the EEPROM from RAM:



Program Backup Here are the steps for backing up your program into EEPROM.

Step	Action
1	The following must be true: There is a valid program in RAM.
2	From the Twido software window bring down the menu under ‘Controller’, scroll down to ‘Backup’ and click on it.

- Program Restore** During power up there is one way the program will be restored to RAM from the EEPROM (assuming there is no cartridge or extended memory in place):
- The RAM program is not valid
- To restore a program manually from EEPROM do the following:
- From the Twido software window bring down the menu under 'Controller', scroll down to 'Restore' and click on it.

Data (%MWs) Backup Here are the steps for backing up data (memory words) into the EEPROM:

Step	Action
1	For this to work the following must be true: A valid program in RAM (%SW96:X6=1). The same valid program already backed up into the EEPROM. Memory words configured in the program.
2	Set %SW97 to the length of the memory words to be saved. Note: Length cannot exceed the configured memory word length, and it must be greater than 0 but not greater than 512.
3	Set %SW96:X0 to 1.

Data (%MWs) Restore Restore %MWs manually by setting system bit %S95 to 1.
For this to work the following must be true:

- A valid backup application is present in the EEPROM
- The application in RAM matches the backup application in EEPROM
- The backup memory words are valid

Backup and Restore with a 32K Backup Cartridge

Introduction

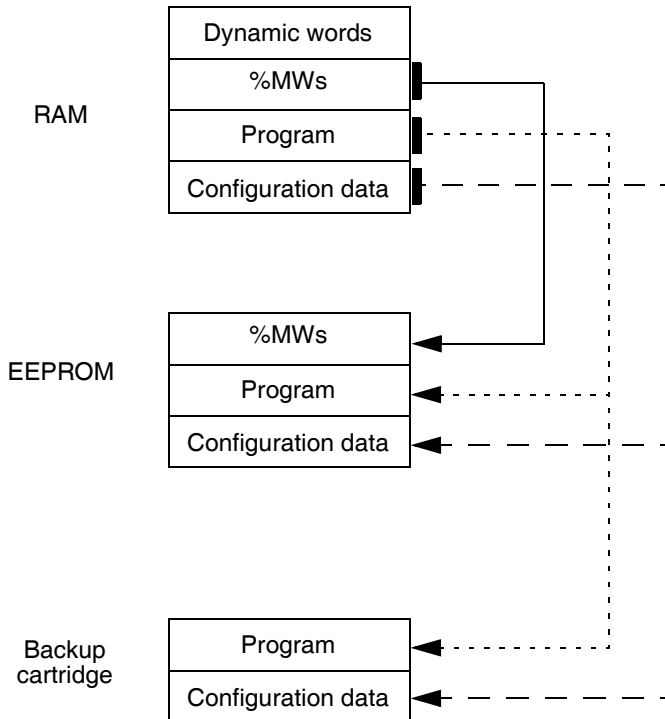
The following information details backup and restore memory functions in modular and compact controllers using a 32K backup cartridge.

At a Glance

The backup cartridge is used to save a program and transfer that program to other Twido controllers. It should be removed from a controller and set aside once the program has been installed or saved. Only program and configuration data can be saved to the cartridge (%MWs cannot be saved to the 32K backup cartridge). Dynamic data can be stored in memory words then backed up to the EEPROM. When program installation is complete any %MWs that were backed up to the internal EEPROM prior to installation will be lost.

Memory Structure

Here is a diagram of a controller's memory structure with the backup cartridge attached. The arrows show what can be backed up to the EEPROM and cartridge from RAM:



Program Backup Here are the steps for backing up your program into the backup cartridge:

Step	Action
1	Power down the controller.
2	Plug in the backup cartridge.
3	Powerup the controller.
4	From the Twido software window bring down the menu under 'Controller', scroll down to 'Backup' and click on it.
5	Power down the controller.
6	Remove backup cartridge from controller.

Program Restore To load a program saved on a backup cartridge into a controller do the following:

Step	Action
1	Power down the controller.
2	Plug in the backup cartridge.
3	Powerup the controller. (If Auto Start is configured you must power cycle again to get to run mode.)
4	Power down the controller.
5	Remove backup cartridge from controller.

Data (%MWs) Backup Here are the steps for backing up data (memory words) into the EEPROM:

Step	Action
1	For this to work the following must be true: A valid program in RAM. The same valid program already backed up into the EEPROM. Memory words configured in the program.
2	Set %SW97 to the length of the memory words to be saved. Note Length cannot exceed the configured memory word length, and it must be greater than 0 but not greater than 512.
3	Set %SW96:X0 to 1.

Data (%MWs) Restore Restore %MWs manually by setting system bit %S95 to 1.
For this to work the following must be true:

- A valid backup application is present in the EEPROM
- The application in RAM matches the backup application in EEPROM
- The backup memory words are valid

Using the 64K Extended Memory Cartridge

Introduction

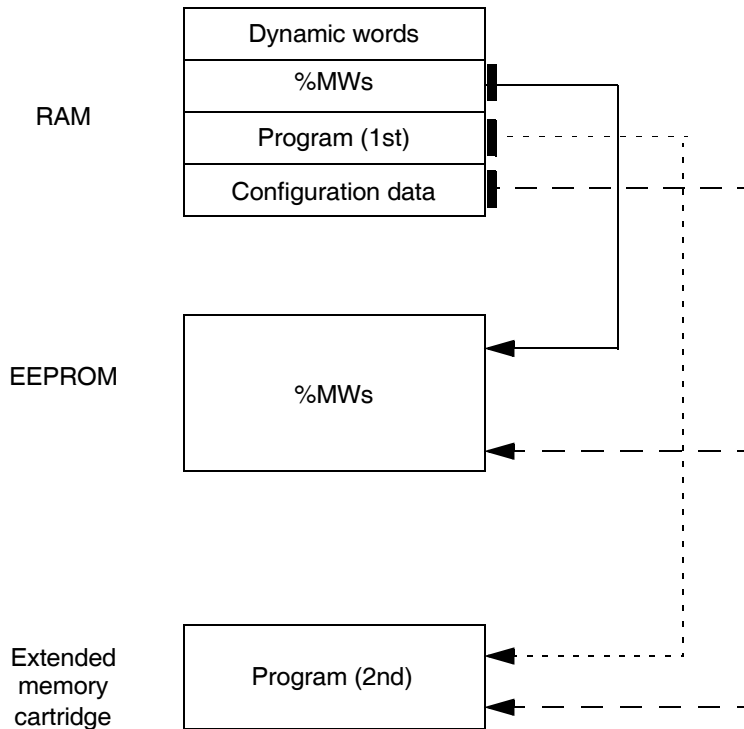
The following information details using the memory functions in modular controllers using a 64K extended memory cartridge.

At a Glance

The 64K extended memory cartridge is used to extend the program memory capability of your Twido controller from 32K to 64K. It must remain plugged into the controller as long as the extended program is being used. If the cartridge is removed the controller will enter the stopped state. Memory words are still backed up into the EEPROM in the controller. Dynamic data can be stored in memory words then backed up to the EEPROM. The 64K extended memory cartridge has the same power up behavior as the 32K backup cartridge.

Memory Structure

Here is a diagram of a controller's memory structure using an extended memory cartridge. The arrows show what is backed up into the EEPROM and the 64K extended memory cartridge from RAM:



Configure Software and Install Extended Memory

Before you begin writing your extended program, you must install the 64K extended memory cartridge into your controller. The following four steps show you how:

Step	Action
1	Under the Hardware option menu on you Twido software window enter 'TWDXCPMF64'.
2	Power down the controller.
3	Plug in the 64K extended memory cartridge.
4	Powerup the controller.

Save your program.

Once your 64K extended memory cartridge has been installed and your program written:

- From the Twido software window bring down the menu under 'Controller', scroll down to 'Backup' and click on it.

Data (%MWs) Backup

Here are the steps for backing up data (memory words) into the EEPROM:

Step	Action
1	For this to work the following must be true: A valid program is present Memory words are configured in the program.
2	Set %SW97 to the length of the memory words to be saved. Note: Length cannot exceed the configured memory word length, and it must be greater than 0 but not greater than 512.
3	Set %SW96:X0 to 1.

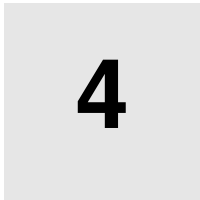
Data (%MWs) Restore

Restore %MWs manually by setting system bit %S95 to 1.

For this to work the following must be true:

- A valid program is present
- The backup memory words are valid

Controller Operating Modes



At a Glance

Subject of this Chapter This chapter describes controller operating modes and cyclic and periodic program execution. Included are details about power outages and restoration.

What's in this Chapter? This chapter contains the following topics:

Topic	Page
Cyclic Scan	62
Periodic Scan	64
Checking Scan Time	67
Operating Modes	68
Dealing with Power Cuts and Power Restoration	69
Dealing with a warm restart	71
Dealing with a cold start	73
Initialization of objects	75

Cyclic Scan

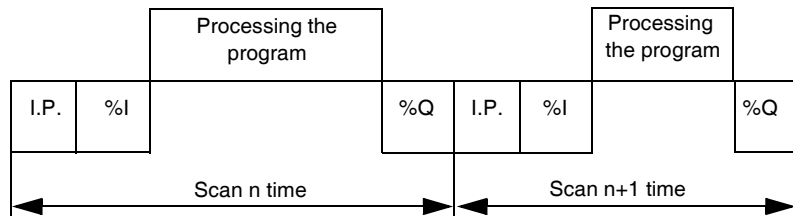
Introduction

Cyclic scanning involves linking controller cycles together one after the other. After having effected the output update (third phase of the task cycle), the system executes a certain number of its own tasks and immediately triggers another task cycle.

Note: The scan time of the user program is monitored by the controller watchdog timer and must not exceed 500 ms. Otherwise a fault appears causing the controller to stop immediately in Halt mode. Outputs in this mode are forced to their default fallback state.

Operation

The following drawing shows the running phases of the cyclical scan time.



Description of the phases of a cycle

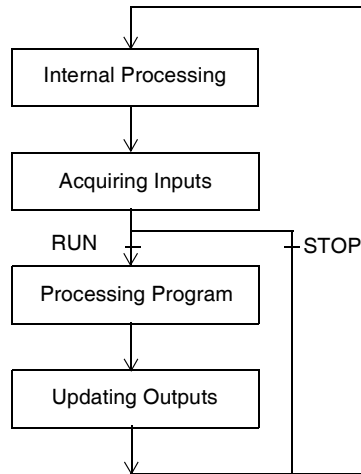
The following table describes the phases of a cycle.

Address	Phase	Description
I.P.	Internal processing	The system implicitly monitors the controller (managing system bits and words, updating current timer values, updating status lights, detecting RUN/STOP switches, etc.) and processes requests from TwidoSoft (modifications and animation).
%I, %IW	Acquisition of input	Writing to the memory the status of discrete and application specific module inputs.
-	Program processing	Running the application program written by the user.
%Q, %QW	Updating of output	Writing output bits or words associated with discrete and application specific modules.

- Operating mode**
- Controller in RUN**, the processor carries out:
- Internal processing
 - Acquisition of input
 - Processing the application program
 - Updating of output
- Controller in STOP**, the processor carries out:
- Internal processing
 - Acquisition of input
-

Illustration

The following illustration shows the operating cycles.

**Check Cycle**

The check cycle is performed by watchdog.

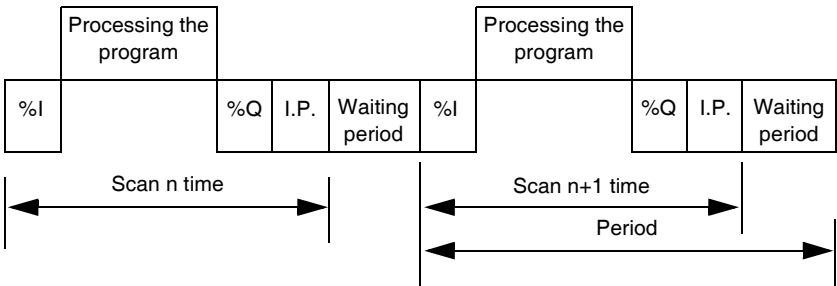
Periodic Scan

Introduction

In this operating mode, acquiring inputs, processing the application program, and updating outputs are done periodically according to the time defined at configuration (from 2-150 ms).
At the beginning of the controller scan, a timer, the value of which is initialized at the period defined at configuration, starts to count down. The controller scan must end before the timer has finished and relaunches a new scan.

Operation

The following drawing shows the running phases of the periodic scan time.



Description of Operating Phases

The table below describes the operating phases.

Address	Phase	Description
I.P.	Internal processing	The system implicitly monitors the controller (managing system bits and words, updating current timer values, updating status lights, detecting RUN/STOP switches, etc.) and processes requests from TwidoSoft (modifications and animation).
%I, %IW	Acquisition of input	Writing to the memory the status of discrete and application specific module inputs.
-	Program processing	Running the application program written by the user.
%Q, %QW	Updating of output	Writing output bits or words associated with discrete and application specific modules.

Operating mode**Controller in RUN**, the processor carries out:

- Internal processing
- Acquisition of input
- Processing the application program
- Updating of output

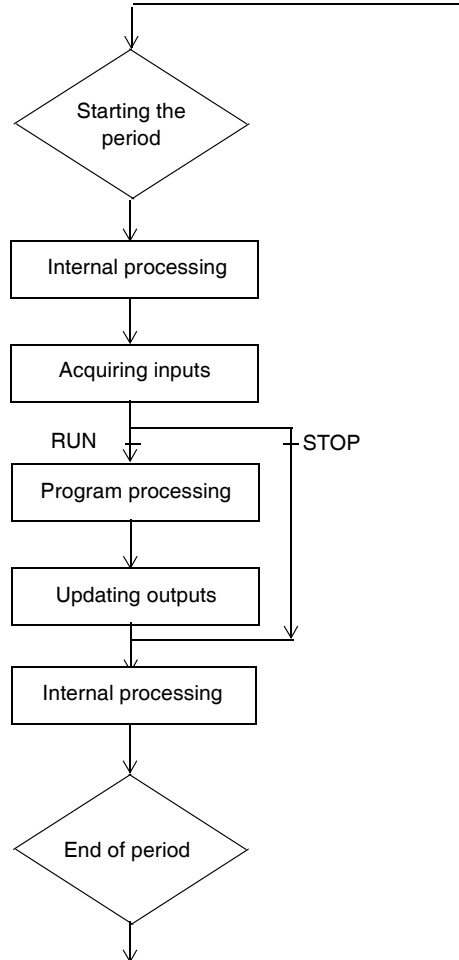
If the period has not finished, the processor completes its operating cycle until the end of the internal processing period. If the operating time is longer than that allocated to the period, the controller indicates that the period has been exceeded by setting the system bit %S19 to 1. The process continues and is run completely. However, it must not exceed the watchdog time limit. The following scan is linked in after writing the outputs of the scan in progress implicitly.

Controller in STOP, the processor carries out:

- Internal processing
 - Acquisition of input
-

Illustration

The following illustration shows the operating cycles.

**Check Cycle**

Two checks are carried out:

- Period overflow
- Watchdog

Checking Scan Time

General

The task cycle is monitored by a watchdog timer called Tmax (a maximal duration of the task cycle). It permits the showing of application errors (infinite loops, and so on.) and assures a maximal duration for output refreshing.

Software WatchDog (Periodic or Cyclic Operation)

In periodic or cyclic operation, the triggering of the watchdog causes a software error. The application passes into a HALT state and sets system bit %S11 to 1. The relaunching of the task necessitates a connection to Twido Soft in order to analyze the cause of the error, modification of the application to correct the error, then reset the program to RUN.

Note: The HALT state is when the application is stopped immediately because of an application software error such as a scan overrun. The data retains the current values, which allows for an analysis of the cause of the error. The program stops on the instruction in progress. Communication with the controller is open.

Check on Periodic Operation

In periodic operation an additional check is used to detect the period being exceeded:

- **%S19** indicates that the period has been exceeded. It is set to:
 - 1 by the system when the scan time is greater than the task period,
 - 0 by the user.
- **%SW0** contains the period value (0-150 ms). It is:
 - Initialized when starting from a cold start by the value selected on the configuration,
 - Able to be modified by the user.

Using Master Task Running Time

The following system words are used for information on the controller scan cycle time:

- **%SW11** initializes to the maximum watchdog time (10 to 500 ms).
- **%SW30** contains the execution time for the last controller scan cycle.
- **%SW31** contains the execution time for the longest controller scan cycle since the last cold start.
- **%SW32** contains the execution time for the shortest controller scan cycle since the last cold start.

Note: This different information can also be accessed from the configuration editor.

Operating Modes

Introduction

Twido Soft is used to take into account the three main operating mode groups:

- Checking
- Running or production
- Stopping

Starting through Grafcet

These different operating modes can be obtained either starting from or using the following Grafcet methods:

- Grafcet initialization
- Presetting of steps
- Maintaining a situation
- Freezing charts

Preliminary processing and use of system bits ensure effective operating mode management without complicating and overburdening the user program.

Grafcet System Bits

Use of bits %S21, %S22 and %S23 is reserved for preliminary processing only. These bits are automatically reset by the system. They must be written by Set Instruction **S** only.

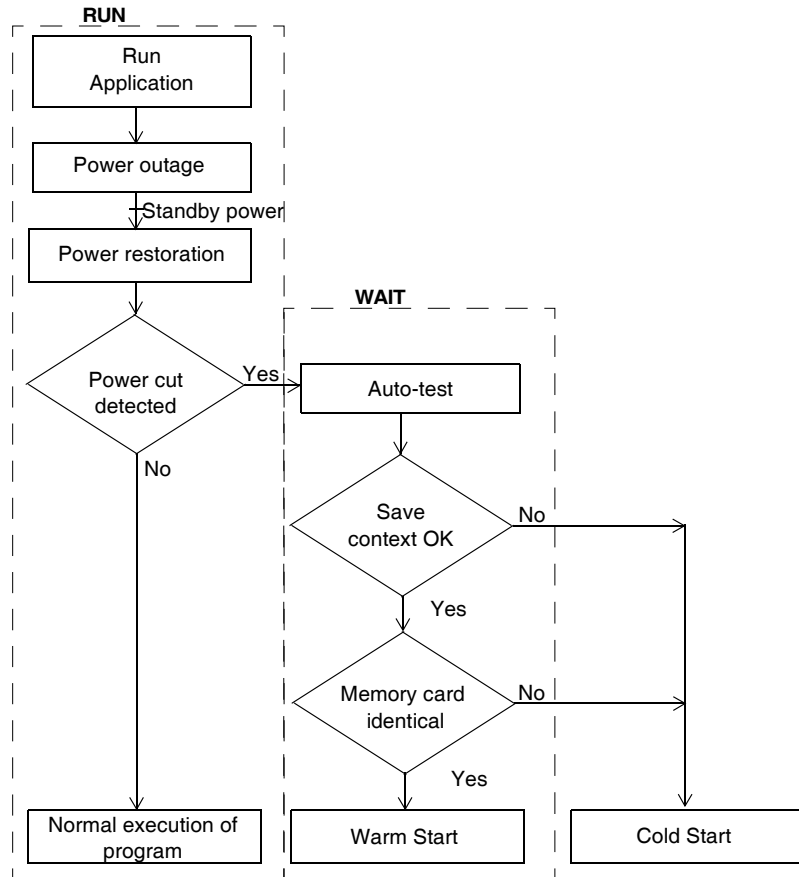
The following table provides Grafcet-related system bits:

Bit	Function	Description
%S21	GRAFCET initialization	Normally set to 0, it is set to 1 by: ● a cold-start, %S0=1; ● The user, in the pre-processing program part only, using a Set Instruction S %S21 or a set coil -(S)- %S21. Consequences: ● Deactivation of all active steps. ● Activation of all initial steps.
%S22	GRAFCET RESET	Normally set to 0, it can only be set to 1 by the program in pre-processing. Consequences: ● Deactivation of all active steps. ● Scanning of sequential processing stopped.
%S23	Preset and freeze GRAFCET	Normally set to 0, it can only be set to 1 by the program in pre-processing. ● Prepositioning by setting %S22 to 1. ● Preposition the steps to be activated by a series of S Xi instructions. ● Enable prepositioning by setting %S23 to 1. Freezing a situation: ● In initial situation: by maintaining %S21 at 1 by program. ● In an "empty" situation: by maintaining %S22 at 1 by program. ● In a situation determined by maintaining %S23 at 1.

Dealing with Power Cuts and Power Restoration

Illustration

The following illustration shows the various power restarts detected by the system. If the duration of the cut is less than the power supply filtering time (about 10 ms for an alternating current supply or 1 ms for a direct current supply), this is not noticed by the program which runs normally.



Note: The context is saved in a battery backed-up RAM. At power up, the system checks the state of the battery and the saved context to decide if a warm start can occur.

Run/Stop Input Bit Versus Auto Run

The Run/Stop input bit has priority over the "Automatic Start in Run" option that is available from the Scan Mode dialog box. If the Run/Stop bit is set, then the controller will restart in the Run Mode when power is restored. The mode of the controller is determined as follows:

Run/Stop Input Bit	Auto Start in Run	Resulting State
Zero	Zero	Stop
Zero	One	Stop
Rising edge	No effect	Run
One	No effect	Run
Not configured in software	Zero	Stop
Not configured in software	One	Run

Note: For all Compact type of controllers of software version V1.0, if the controller was in Run mode when power was interrupted, and the "Automatic Start in Run" flag was not set from the Scan Mode dialog box, the controller will restart in Stop mode when power is restored. Otherwise will perform a cold restart.

Note: For all Modular and Compact type of controllers of software version V1.11, if the battery in the controller is operating normally when power was interrupted, the controller will startup in the mode that was in effect at the time the power was interrupted. The "Automatic Start in Run" flag, that was selected from the Scan Mode dialog, will have no effect on the mode when the power is restored.

Operation

The table below describes the processing phases for power cuts.

Phase	Description
1	In the event of a power cut the system stores the application context and the time of the cut.
2	All outputs are set to fallback status (0).
3	When power is restored, the context saved is compared with the one in progress which defines the type of start to run: • If the application context has changed (loss of system context or new application), the controller initializes the application: Cold restart (systematic for compact). • If the application context is the same, the controller restarts without initializing data: warm restart.

Dealing with a warm restart

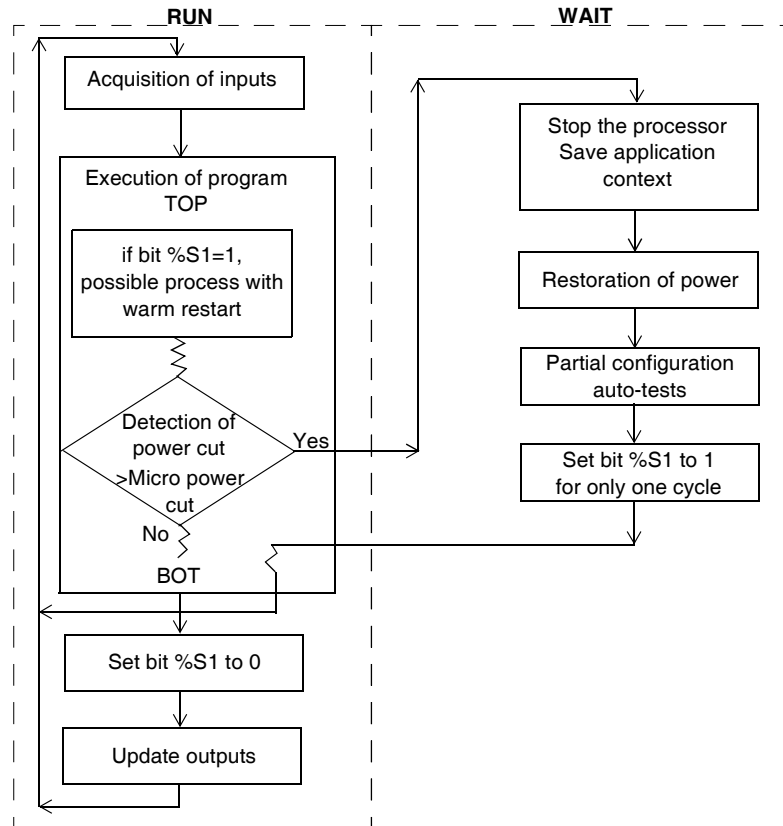
Cause of a Warm Restart

A warm restart can occur:

- When power is restored without loss of application context,
- When bit **%S1** is set to state 1 by the program,
- From the Operator Display when the controller is in STOP mode

Illustration

The drawing below describes a warm restart operation in RUN mode.



Restart of the Program Execution

The table below describes the restart phases for running a program after a warm restart.

Phase	Description
1	The program execution resumes from the same element where it was prior to the power cut, without updating the outputs. Note: Only the same element from the user code is restarted. The system code (for example, the updating of outputs) is not restarted.
2	At the end of the restart cycle, the system: ● Unreserves the application if it was reserved (and provokes a STOP application in case of debugging) ● Reinitializes the messages
3	The system carries out a restart cycle in which it: ● Relaunches the task with bits %S1 (warm-start indicator) and %S13 (first cycle in RUN) set to 1 ● Resets bits %S1 and %S13 to 0 at the end of the first task cycle

Processing of a Warm-Start

In the event of a warm-start, if a particular application process is required, bit **%S1** must be tested at the start of the task cycle, and the corresponding program called up.

Outputs after Power Failure

Once a power outage is detected, outputs are set to (default) fallback status (0). When power is restored, outputs are at last state until they are updated again by the task.

Dealing with a cold start

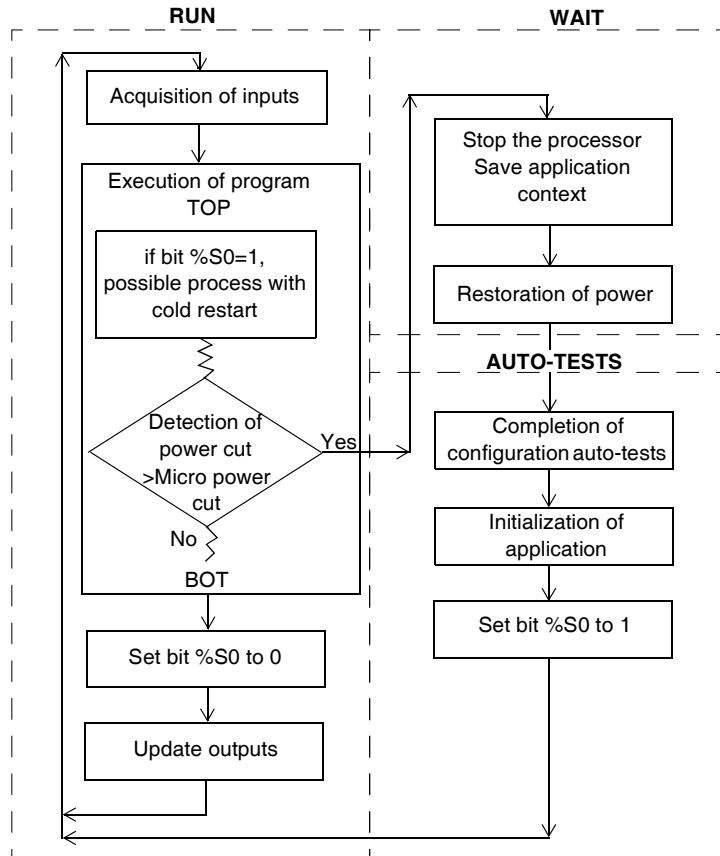
Cause of a Cold Start

A cold-start can occur:

- When loading a new application into RAM
- When power is restored with loss of application context
- When system bit **%S0** is set to state 1 by the program
- From the Operator Display when the controller is in STOP mode

Illustration

The drawing below describes a cold restart operation in RUN mode.



Operation

The table below describes the restart phases for running a program after a cold restart.

Phase	Description
1	At start up, the controller is in RUN. At a cold restart after a stop due to an error, the system forces a cold restart. The program execution restarts at the beginning of the cycle.
2	The system: ● Resets internal bits and words and the I/O images to 0 ● Initializes system bits and words ● Initializes function blocks from configuration data
3	For this first restart cycle, the system: ● Relaunches the task with bits %S0 (cold-start indicator) and %S13 (first cycle in RUN) set to 1 ● Resets bits %S0 and %S13 to 0 at the end of this first task cycle ● Sets bits %S31 and %S38 (event control indicators) to their initial state 1. ● Resets bits %S39 (event control indicator) and word %SW48 (counts all events executed except periodic events).

Processing of a Cold-Start

In the event of a cold-start, if a particular application process is required, bit %S0 (which is at 1) must be tested during the first cycle of the task.

Outputs after Power Failure

Once a power outage is detected, outputs are set to (default) fallback status (0). When power is restored, outputs are at zero until they are updated again by the task.

Initialization of objects

Introduction

The controllers can be initialized by Twido Soft by setting system bits **%S0** (a cold restart) and **%S1** (a warm restart).

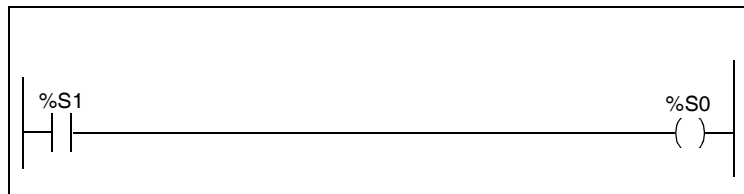
Cold Start Initialization

For a cold start initialization, system bit **%S0** must be set to 1.

Initialization of objects (identical to cold start) on power-up using %S0 and %S1

To initialize objects on power-up, system bit **%S1** and **%S0** must be set to 1.

The following example shows how to program a warm restart object initialization using system bits.



LD %S1 If %S1 = 1 (warm restart), set %S0 to 1 initialize the controller.
ST %S0 These two bits are reset to 0 by the system at the end of the following scan.

Note: Do not set %S0 to 1 for more than one controller scan.

Event task management

5

In Brief...

At a Glance

This chapter describes event tasks and how they are executed in the controller.

Note: Event tasks are not managed by the 10 I/O Twido controller (TWDLCAA10DRF).

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Overview of event tasks	78
Description of different event sources	79
Event management	80

Overview of event tasks

Introduction

The previous chapter presented periodic (See *Periodic Scan*, p. 64) and cyclic (See *Cyclic Scan*, p. 62) tasks in which objects are updated at the start and end of the task. Event sources may cause a certain task to be stopped while higher priority (event) tasks are executed to allow objects to be updated more quickly.

An event task:

- is a part of a program executed when a given condition is met (event source),
 - has a higher priority than the main program,
 - guarantees a rapid response time enabling the overall response time of the system to be reduced.
-

Description of an Event

An event is composed of:

- an event source which can be defined as a software or hardware interrupt condition to interrupt the main program (See *Description of different event sources*, p. 79),
 - a section which is a independent programmed entity related to an event,
 - an event queue which can be used to store a list of events until they are executed,
 - a priority level which specifies the order of event execution.
-

Description of different event sources

Overview of Different Event Sources

An event source needs to be managed by the software to make the sure the main program is properly interrupted by the event, and to call the programming section linked to the event. The application scan time has no effect on the execution of the events.

The following 9 event sources are allowed:

- 4 conditions linked to the VFC function block thresholds (2 events per %VFC instance),
- 4 conditions linked to the physical inputs of a controller base,
- 1 periodic condition.

An event source can only be attached to a single event, and must be immediately detected by TwidoSoft. Once it is detected, the software executes the programming section attached to the event: each event is attached to a subroutine labeled **SRI**: defined on configuration of the event sources.

Physical Input Events of a Controller Base

Inputs %I0.2, %I0.3, %I0.4 and %I0.5 can be used as event sources, provided they are not locked and that the events are allowed during configuration.

Event processing can be activated by inputs 2 to 5 of a controller base (position 0), on a rising or falling edge.

For further details on configuring this event, refer to the section entitled "Hardware Configuration -> Input Configuration" in the "TwidoSoft Operation Guide" on-line help.

Output Event of a %VFC Function Block

Outputs TH0 and TH1 of the %VFC function block are event sources. Outputs TH0 and TH1 are respectively set:

- to 1 when the value is greater than threshold S0 and threshold S1,
- to 0 when the value is less than threshold S0 and threshold S1.

A rising or falling edge of these outputs can activate an event process.

For further details on configuring this event, refer to the section entitled "Software Configuration -> Very Fast Counters" in the "TwidoSoft Operation Guide" on-line help.

Periodic event

This event periodically executes a single programming section. This task has higher priority than the main task (master).

However, this event source has lower priority than the other event sources.

The period of this task is set on configuration, from 5 to 255 ms. Only one periodic event can be used.

For further details on configuring this event, refer to the section entitled "Configuring Program Parameters -> Scan Mode" in the "TwidoSoft Operation Guide" on-line help.

Event management

Events queue and priority

Events have 2 possible priorities: High and Low. But only **one** type of event (thus only one event source) can have High priority. The other events therefore have Low priority, and their order of execution depends on the order in which they are detected.

To manage the execution order of the event tasks, there are two event queues:

- in one, up to 16 High priority events can be stored (from the same event source),
- in the other, up to 16 Low priority events can be stored (from other event sources).

These queues are managed on a FIFO basis: the first event to be stored is the first to be executed. But they can only hold 16 events, and all additional events are lost. The Low priority queue is only executed once the High priority queue is empty.

Event Queue Management

Each time an interrupt appears (linked to an event source), the following sequence is launched:

Step	Description
1	Interrupt management: • recognition of the physical interrupt, • event stored in the suitable event queue, • verification that no event of the same priority is pending (if so the event stays pending in the queue).
2	Save context.
3	Execution of the programming section (subroutine labeled SRi:) linked to the event.
4	Updating of output
5	Restore context

Before the context is re-established, all the events in the queue must be executed.

Event check

System bits and words are used to check the events (See *System Bits and System Words*, p. 595):

- %S31: used to execute or delay an event,
- %S38: used to decide whether or not to place events in the events queue,
- %S39: used to find out if events are lost,
- %SW48: shows how many events have been executed since the last cold start (counts all events except periodic events.)

The value of bit %S39 and word %SW48 is reset to zero and that of %S31 and %S38 is set to its initial state 1 on a cold restart or after an application is loaded, but remains unchanged after a warm restart. In all cases, the events queue is reset.

Special Functions



At a Glance

Subject of this Part	This part describes communications, built-in analog functions, managing analog I/O modules, installing the AS-Interface V2 bus and the CANopen fieldbus for Twido controllers.
What's in this Part?	This part contains the following chapters:

Chapter	Chapter Name	Page
6	Communications	83
7	Built-In Analog Functions	185
8	Managing Analog Modules	189
9	Installing the AS-Interface V2 bus	201
10	Installing and Configuring the CANopen Fieldbus	235
11	Configuring the TwidoPort Ethernet Gateway	277
12	Operator Display Operation	305

Communications

6

At a Glance

Subject of this Chapter

This chapter provides an overview of configuring, programming, and managing communications available with Twido controllers.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Presentation of the different types of communication	84
TwidoSoft to Controller communications	86
Communication between TwidoSoft and a Modem	92
Remote Link Communications	104
ASCII Communications	115
Modbus Communications	126
Standard Modbus Requests	144
Transparent Ready Implementation Class (Twido Serial A05, Ethernet A15)	150
Ethernet TCP/IP Communications Overview	151
Quick TCP/IP Setup Guide for PC-to-Controller Ethernet Communication	153
Connecting your Controller to the Network	159
IP Addressing	160
Assigning IP Addresses	161
TCP/IP Setup	165
IP Address Configure Tab	167
Marked IP Tab	169
Time out Tab	171
Remote Devices Tab	173
Viewing the Ethernet Configuration	175
Ethernet Connections Management	176
Ethernet LED Indicators	178
TCP Modbus Messaging	180

Presentation of the different types of communication

At a Glance

Twido provides one or two serial communications ports used for communications to remote I/O controllers, peer controllers, or general devices. Either port, if available, can be used for any of the services, with the exception of communicating with Twido Soft, which can only be performed using the first port. Three different base protocols are supported on each Twido controller: Remote Link, ASCII, or Modbus (modbus master or modbus slave).

Moreover, the TWDLCAE40DRF compact controller provides one RJ-45 Ethernet communications port. It supports the Modbus TCP/IP client/server protocol for peer-to-peer communications between controllers over the Ethernet network.

Remote Link

The remote link is a high-speed master/slave bus designed to communicate a small amount of data between the master controller and up to seven remote (slave) controllers. Application or I/O data is transferred, depending on the configuration of the remote controllers. A mixture of remote controller types is possible, where some can be remote I/O and some can be peers.

ASCII

The ASCII protocol is a simple half-duplex character mode protocol used to transmit and/or receive a character string to/from a simple device (printer or terminal). This protocol is supported only via the "EXCH" instruction.

Modbus

The Modbus protocol is a master/slave protocol that allows for one, and only one, master to request responses from slaves, or to act based on the request. The master can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a message (response) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

Modbus master - The modbus master mode allows the Twido controller to send a modbus query to a slave and await its reply. The modbus master mode is supported only via the "EXCH" instruction. Both Modbus ASCII and RTU are supported in modbus master mode.

Modbus Slave - The modbus slave mode allows the Twido controller to respond to modbus queries from a modbus master, and is the default communications mode if no other type of communication is configured. The Twido controller supports the standard modbus data and control functions and service extensions for object access. Both Modbus ASCII and RTU are supported in modbus slave mode.

Note: 32 devices (without repeaters) can be installed on an RS-485 network (1 master and up to 31 slaves), the addresses of which can be between 1 and 247.

Modbus TCP/IP

Note: Modbus TCP/IP is solely supported by TWDLCAE40DRF series of compact controllers with built-in Ethernet network interface.

The following information describes the Modbus Application Protocol (MBAP).

The Modbus Application Protocol (MBAP) is a layer-7 protocol providing peer-to-peer communication between programmable logic controllers (PLCs) and other nodes on a LAN.

The current Twido controller TWDLCAE40DRF implementation transports Modbus Application Protocol over TCP/IP on the Ethernet network. Modbus protocol transactions are typical request-response message pairs. A PLC can be both client and server depending on whether it is querying or answering messages.

TwidoSoft to Controller communications

At a Glance

Each Twido controller has on its Port 1 a built-in EIA RS-485 terminal port. This has its own internal power supply. Port 1 must be used to communicate with the TwidoSoft programming software.

No optional cartridge or communication module can be used for this port. A modem, however, can use this port.

There are several ways to connect the PC to the Twido controller RS-485 Port 1:

- By TSXPCX cable,
- By telephone line: Modem connection.

Moreover, the TWDLCAE40DRF compact controller has a built-in RJ-45 Ethernet network connection port that can be used to communicate with the Ethernet-capable PC running the TwidoSoft programming software.

There are two ways for the Ethernet-capable PC to communicate with the TWDLCAE40DRF Twido controller RJ-45 port:

- By direct cable connection via a UTP Cat5 RJ45 Ethernet crossover cable (not recommended),
- By connection to the Ethernet network via a SFTP Cat5 RJ45 Ethernet cable available from the Schneider Electric catalog (cable reference: 490NTW000●●).

CAUTION

EQUIPMENT DAMAGE

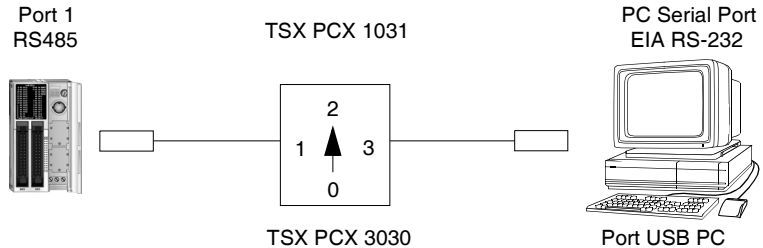
TwidoSoft may not sense the disconnection when physically moving the TSXPCX1031, TSX PCX 3030 or Ethernet communication cable from a first controller and quickly inserting it in a second controller. To avoid this condition, use TwidoSoft to disconnect before moving the cable.

Failure to follow this instruction can result in injury or equipment damage.

TSXPCX Cable Connection

The EIA RS-232C or USB port on your personal computer is connected to the controller's Port 1 using the TSXPCX1031 or TSX PCX 3030 multi-function communication cable. This cable converts signals between EIA RS-232 and EIA RS-485 for the TSX PCX 1031 and between USB and EIA RS-485 for the TSX PCX 3030. This cable is equipped with a 4-position rotary switch to select different modes of operation. The switch designates the four positions as "0-3", and the appropriate setting for TwidoSoft to Twido controller is location 2.

This connection is illustrated in the diagram below.



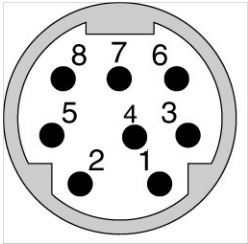
Note: For this cable, the DPT signal on pin 5 is not tied to 0V. This indicates to the controller that the current connection is a TwidoSoft connection. The signal is pulled up internally, informing the firmware executive that this is a TwidoSoft connection.

Pin outs of Male and Female Connectors

The following figure shows the pin outs of a male 8-pin miniDIN connector and of a terminal:

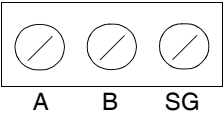
Mini DIN

TWD NAC232D, TWD NAC485D
TWD NOZ485D, TWD NOZ232D



Terminal block

TWD NAC485T
TWD NOZ485T

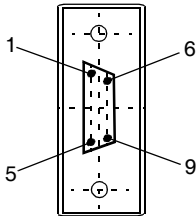


Pin outs	Base RS485	RS485 option	RS232-C
1	D1 (A+)	D1 (A+)	RTS
2	D0 (B-)	D0 (B-)	DTR
3	NC	NC	TXD
4	/DE	NC	RXD
5	/DPT	NC	DSR
6	NC	NC	GND
7	0 V	0 V	GND
8	5 V	5 V	5 V

Pin outs	RS485
A	D1 (A+)
B	D0 (B-)
SG	0V

Note: Maximum total consumption for 5V mode (pin 8): 180mA

The following figure shows the pin outs of a SubD female 9-pin connector for the TSX PCX 1031.



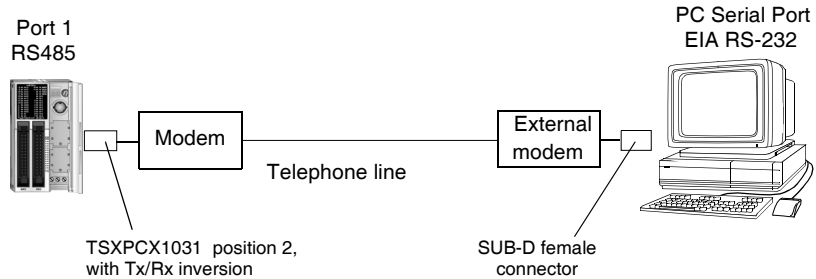
Pin outs	RS232
1	DCD
2	RX
3	TX
4	DTR
5	SG
6	NC
7	RTS
8	CTS
9	NC

Telephone Line Connection

A modem (See *Communication between TwidoSoft and a Modem*, p. 92) connection enables programming of and communication with the controller using a telephone line.

The modem associated with the controller is a **receiving** modem connected to port 1 of the controller. The modem associated with the PC can be internal, or external and connected to a COM serial port.

This connection is illustrated in the diagram below.



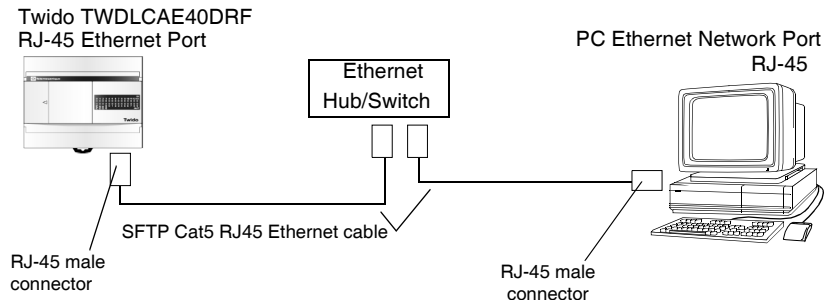
Note: Only one modem can be connected to port 1 of the controller.

Note: Caution. Remember to install the software provided with the modem, as TwidoSoft only takes into account the installed modems.

Ethernet Network Connection

Note: Although direct cable connection (using a Ethernet crossover cable) is supported between the Twido TWDLCAE40DRF and the PC running the TwidoSoft programming software, we do not recommend it. Therefore, you should always favor a connection via a network Ethernet hub/switch.

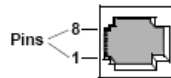
The following figure shows a PC-to-Twido connection via a network Ethernet hub/switch:



Note: The PC running the TwidoSoft application must be Ethernet-capable.

The Twido TWDLCAE40DRF features a RJ-45 connector to connect to the 100 BASE-TX network Ethernet with auto negotiation. It can accommodate both 100Mbps and 10 Mbps network speeds.

The following figure shows the RJ-45 connector of the Twido controller:



The eight pins of the RJ-45 connector are arranged vertically and numbered in order from bottom to top. The pinout for the RJ-45 connector is described in the table below:

Pinout	Function	Polarity
8	NC	
7	NC	
6	RxD	(-)
5	NC	
4	NC	
3	RxD	(+)

Pinout	Function	Polarity
2	TxD	(-)
1	TxD	(+)

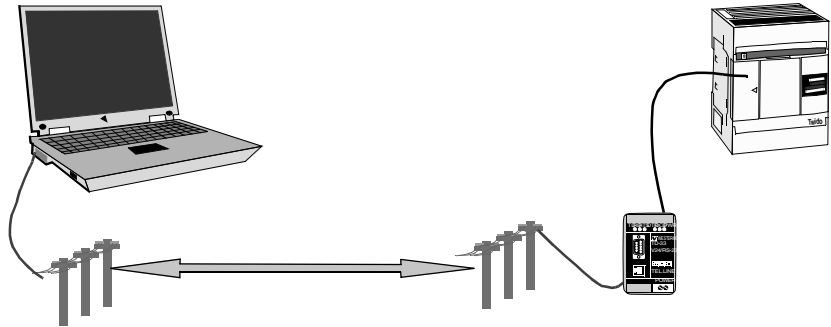
Note:

- The same connector and pinout is used for both 10Base-T and 100Base-TX.
- When connecting the Twido controller to a 100Base-TX network, you should use at least a category 5 Ethernet cable.

Communication between TwidoSoft and a Modem

General

A PC executing Twidosoft can be connected to a Twido controller for transferring applications, animating objects and executing operator mode commands. It is also possible to connect a Twido controller to other devices, such as another Twido controller, for establishing communication with the application process.



Installing the Modem

All modems the user wishes to use with Twidosoft must be installed running Windows from your PC. To install your modems running Windows, follow the Windows documentation. This installation is independent from Twidosoft.

**Establishing
Connection**

The default communication connection between Twidosoft and the Twido controller is made by a serial communication port, using the TSX PCX 1031 cable and a crossed adaptater (see *Appendix 1, p. 101*).
If a modem is used to connect the PC, this must be indicated in the Twidosoft software.
To select a connection using Twidosoft, click "file", then "preferences".

Preferences

Default Program Editor

List

Ladder

List/Ladder Animation

Hex

Decimal

Ladder Information

1 line

3 lines (symbols AND addresses)

3 lines (symbols OR addresses)

Display Attributes

Symbols

Addresses

Automatic save

Save message

Close Ladder viewer on Edit Rung

Display Toolbars

Auto Line Validate

Automatic validation of configuration editor

Connections management

Connection:

COM1

OK

Cancel

Help

This screen allows you to select a connection or manage connections (creation, modification, etc.).
To use an existing connection, select it from those displayed in the drop-down menu.
If you have to add, modify or delete a connection, click once on "Manage connections"; a window opens displaying the list of connections and their properties.

Connections management

Name	Connection type	IP / Phone	P-Unit / Address	Baudrate	Parity	Stop Bits	Timeout	Break timeout
COM1	Serial	COM1	@	19200	None	1	5000	20
TCP/IP01	TCP/IP	192.163.1.101	Direct				3000	500
My Modem1	MODEM: TOSHIBA	0231858445		19200	None	1	5000	20

<

III

>

Add

Modify

Delete

Help

OK

In this case, 2 serial ports are displayed (Com1 and Com4), as well as a modem connection showing a TOSHIBA V.90 model configured to compose the number: 0231858445 (national call).

You can change the name of each connection for application maintenance purposes (COM1 or COM4 cannot be changed).

This is how you define and select the connection you wish to use for connecting your PC to a modem.

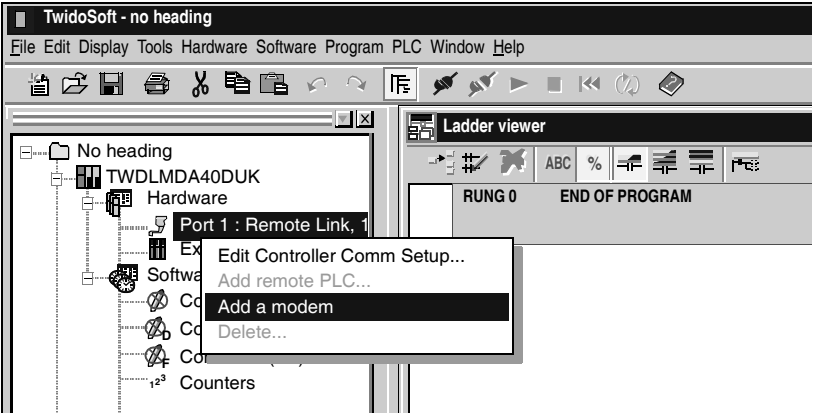
However, this is just part of the process for making an overall connection between the computer and the Twido controller.

The next step involves the Twido controller. The remote Twido must be connected to a modem.

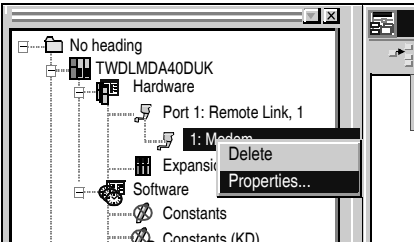
All modems need to be initialized to establish a connection. The Twido controller containing at least version V2.0 firmware is capable, on power-up, of sending an adapted string to the modem, if the modem is configured in the application.

Configuring the Modem

The procedure for configuring a modem in a Twido controller is as follows:



Once the modem is configured on port 1, the properties must be defined. Right-click on the modem to reveal the choice of "delete" or "properties". Clicking "properties" lets you either select a known modem, create a new modem, or modify a modem.

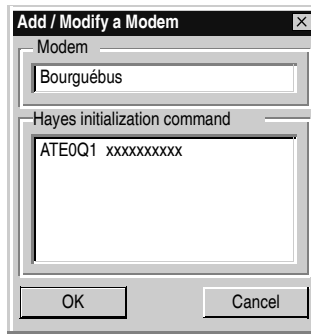


Note: The modem is completely managed by the Twido controller through port 1. This means you can connect a modem to communication port 2, but in this case all of the modem's operating modes and its initialization sequence must be performed manually, and cannot be performed in the same way as with communication port 1.

Next, select "properties", then:



You can select a previously-defined modem, or create a new one by clicking "...".



Then give the new profile a name and complete the Hayes initialization commands as described in the modem documentation.

In the image, "xxxxxx" represents the initialization sequence you must enter to prepare the modem for suitable communication, i.e. the baud rate, parity, stop bit, and receive mode.

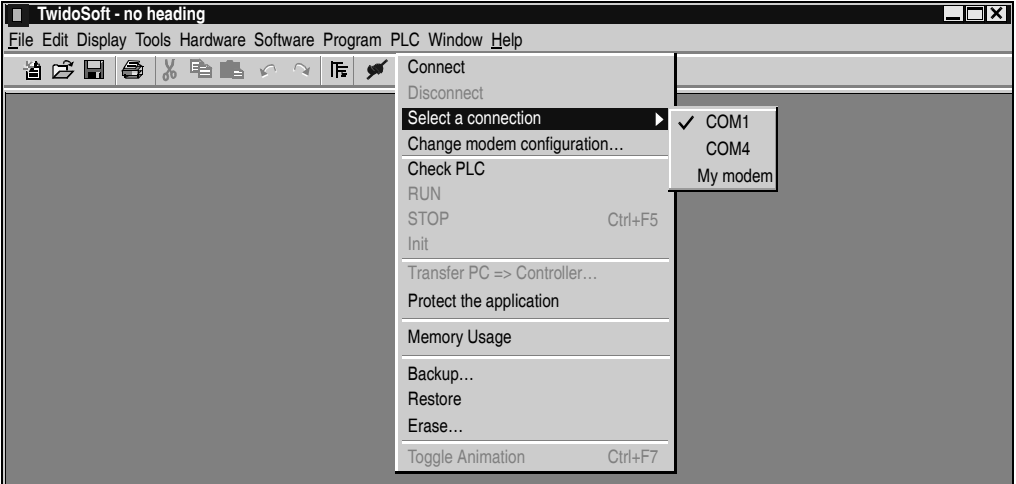
To complete the sequence, please refer to your modem documentation.

The maximum string length is: 127 characters.

When your application is complete, or at least when communication port 1 is fully described, transfer the application using a "point to point connection".

The Twido controller is now ready to be connected to a PC executing Twidosoft via modems.

Connection Sequence Once Twidosoft and the Twido controller are prepared, establish connection as follows:

Step	Action
1	Power-up the Twido controller and modem.
2	Start your computer and run Twidosoft.
3	Select the "PLC" menu, then "Select a connection", and select "My modem" (or the name you have given to your modem connection – see "creation of a connection":)
	
4	Connect TwidoSoft

Note: If you want to use your modem connection all the time, click "file", "preferences", and select "my modem" (or the name you have given it). Twidosoft will now memorize this preference.

Operating Modes The Twido controller sends the initialization string to the connected, powered-up modem. When a modem is configured in the Twido application, the controller first sends an "FF" command to establish whether the modem is connected. If the controller receives an answer, the initialization string is sent to the modem.

Internal, External and International Calls

If you are communicating with a Twido controller within your company premises, you can use just the line extension needed to dial, such as: 8445

Connections management

Name	Connection type	IP / Phone	P-Unit / Address	Baudrate	Parity	Stop Bits	Timeout	Break timeout
COM1	Serial	COM1	@	19200	None	1	5000	20
TCPIP01	TCP/IP	192.163.1.101	Direct				3000	500
My Modem1	MODEM: TOSHIBA	8445		19200	None	1	5000	20

Add

Modify

Delete

Help

OK

If you are using an internal switchboard to dial telephone numbers outside your company and you have to first press "0" or "9" before the number, use this syntax: 0,0231858445 or 9,0231858445

Connections management

Name	Connection type	IP / Phone	P-Unit / Address	Baudrate	Parity	Stop Bits	Timeout	Break timeout
COM1	Serial	COM1	@	19200	None	1	5000	20
TCPIP01	TCP/IP	192.163.1.101	Direct				3000	500
My Modem1	MODEM: TOSHIBA	0,0231858445		19200	None	1	5000	20

Add

Modify

Delete

Help

OK

For international calls, the syntax is: +19788699001, for example. And if you are using a switchboard: 0,+ 19788699001

Connections management

Name	Connection type	IP / Phone	P-Unit / Address	Baudrate	Parity	Stop Bits	Timeout	Break timeout
COM1	Serial	COM1	@	19200	None	1	5000	20
TCPIP01	TCP/IP	192.163.1.101	Direct				3000	500
My Modem1	MODEM: TOSHIBA	0,+19788699001		19200	None	1	5000	20

Add

Modify

Delete

Help

OK

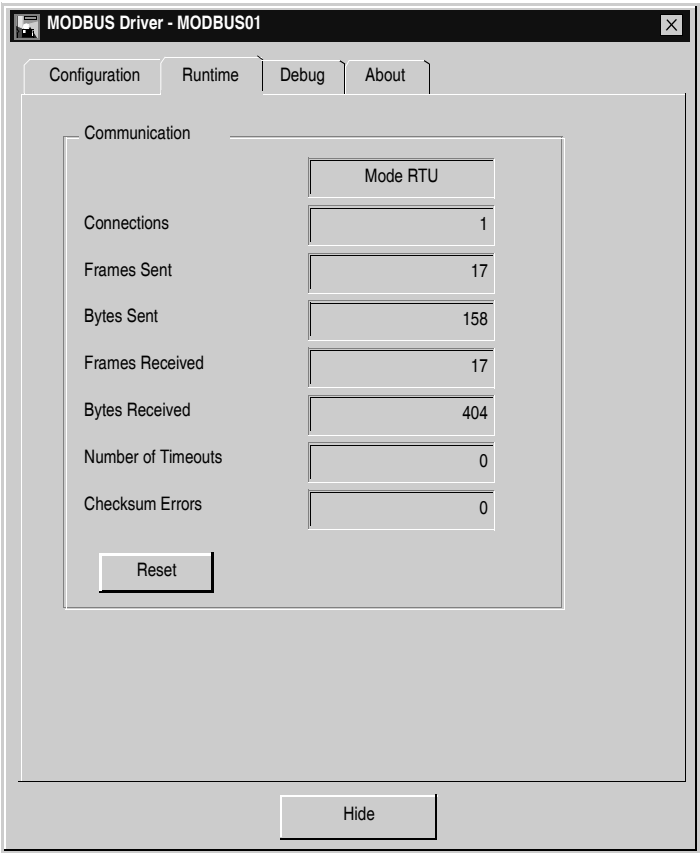
**Frequently
Asked Questions**

When your communication has been established for a few minutes, you can experience some communication errors. In this case, you must adjust the communication parameters.

Twidosoft uses a modbus driver to communicate via serial ports or internal modems. When communication starts, the modbus driver is visible in the toolbar. Double-click on the modbus driver icon to open the window. You now have access to the modbus driver parameters, and the "runtime" tab gives you information on the frames exchanged with the remote controller.

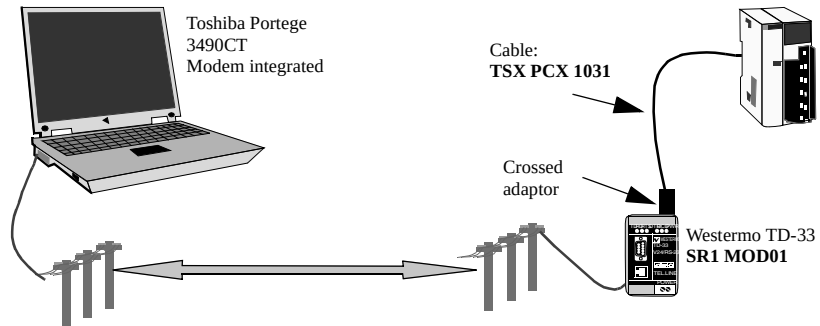
If the "Number of timeouts" increases or is other than 0, change the value using "Connection management", accessible using Twidosoft by clicking "File" then "Preferences" and "Connection management". Click on the "timeout" field, then click the modification button and enter a new, higher value. The default value is "5000", in milliseconds.

Try again with a new connection. Adjust the value until your connection stabilizes.



Examples

- **Example 1:** Twidosoft connected to a TWD LMDA 20DRT (Windows 98 SE).
 - PC: Toshiba Portege 3490CT running Windows 98,
 - Modem (internal on PC): Toshiba internal V.90 modem,
 - Twido Controller: TWD LMDA 20DRT version 2.0,
 - Modem (connected to Twido): Type Westermo TD-33 / V.90, reference SR1 MOD01, available from the new Twido catalog (September 03) (see *Appendix 2, p. 102*),
(North American customers only: The modem type that is available in your area is TD-33/V.90 US),
 - Cable: TSX PCX 1031, connected to Twido communication port 1, and an adaptor: 9 pin male / 9 pin male, in order to cross Rx and Tx during connection between the Westermo modem and the Twido controller (see *Appendix 1, p. 101*). You can also use the TSX PCX 1130 cable (RS485/232 conversion and Rx/Tx crossing).

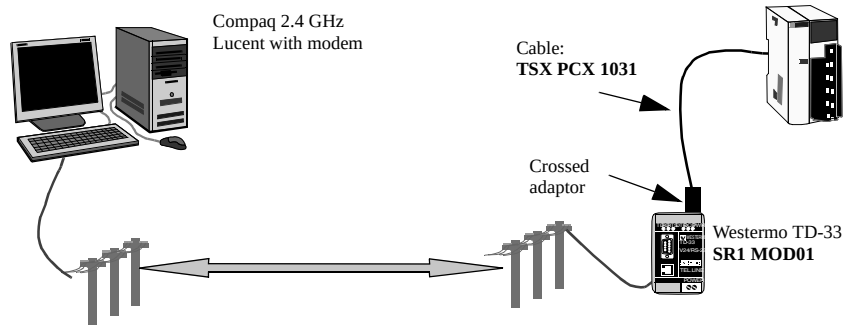


The first test involves using 2 analog telephone lines internal to the company, and not using the entire number – just the extension (hence only 4 digits for the internal Toshiba V.90 modem telephone number).

For this test, the connection parameters (Twidosoft menu "preferences" then "Connection management") were established with their default value, with a timeout of 5000 and break timeout of 20.

- **Example 2:** Twidosoft connected to TWD LMDA 20DRT (windows XP Pro)
 - PC: Compaq Pentium 4, 2.4GHz,
 - Modem: Lucent Win modem, PCI bus,
 - Twido Controller: TWD LMDA 20DRT version 2.0,
 - Modem (connected to Twido): Type WESTERMO TD-33 / V.90, reference SR1 MOD01, available from the new Twido catalog (September 03) (see *Appendix 2, p. 102*),
(North American customers only: The modem type that is available in your area is TD-33/V.90 US),

- Cable: TSX PCX 1031, connected to Twido communication port 1, and an adaptor: 9 pin male / 9 pin male, in order to cross Rx and Tx during connection between the Westermo modem and the Twido controller (see *Appendix 1*, p. 101). You can also use the TSX PCX 1130 cable (RS485/232 conversion and Rx/Tx crossing).

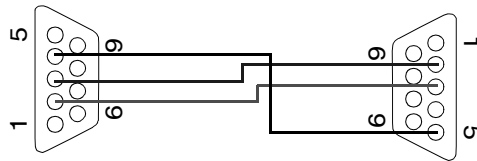


The first test involves using two analog telephone lines internal to the company, and not using the entire number – just the extension (hence only 4 digits for the internal Toshiba V.90 modem telephone number).

For this test, the connection parameters (Twidosoft menu "preferences" then "Connection management") were established with their default value, with a timeout of 5000 and break timeout of 20.

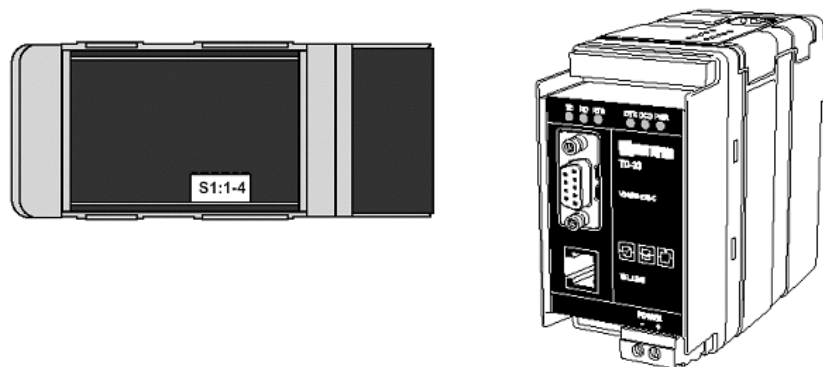
Appendix 1

Crossed adaptor for cable TSX PCX 1031 and Westermo TD-33 modem:

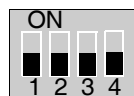


Appendix 2

Modem Westermo TD-33, Schneider reference number **SR1 MOD01⁽¹⁾**. This modem manages four DIP switches, which must all be set to **OFF**:



Factory Settings



Use stored configuration (speed & format etc)
Disable DTR Hotcall, Auto Band

Note:

1. Certain products may not be compatible and/or available in all areas. Please contact your local Schneider representative for availability.

Appendix 3

Wavecom WMOD2B modem, Schneider reference number **SR1 MOD02⁽¹⁾** double band (900/1800Hz):

**Note:**

1. Certain products may not be compatible and/or available in all areas. Please contact your local Schneider representative for availability.

Appendix 4

Reference numbers of the products used in this document:

- Twido product: TWD LMDA 20DRT,
- Twidosoft software: TWD SPU 1002 V10M,
- TSX PCX 1031 cable,
- TSX PCX 1130 cable,
- RTU modem: Westermo TD-33 / V90 **SR1 MOD01⁽¹⁾**,
- GSM modem: Wavecom WMOD2B **SR1 MOD02⁽¹⁾**.

Note:

1. Certain products may not be compatible and/or available in all areas. Please contact your local Schneider representative for availability.
-

Remote Link Communications

Introduction

The remote link is a high-speed master/slave bus designed to communicate a small amount of data between the master controller and up to seven remote (slave) controllers. Application or I/O data is transferred, depending on the configuration of the remote controllers. A mixture of remote controller types is possible, where some can be remote I/O and some can be peers.

Note: The master controller contains information regarding the address of a remote I/O. It does not know which specific controller is at the address. Therefore, the master cannot validate that all the remote inputs and outputs used in the user application actually exist. Take care that these remote inputs or outputs actually exist.

Note: The remote I/O bus and the protocol used is proprietary and no third party devices are allowed on the network.

CAUTION

UNEXPECTED EQUIPMENT OPERATION

- Be sure that there is only one master controller on a remote link and that each slave has a unique address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results.
- Be sure that all slaves have unique addresses. No two slaves should have the same address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results.

Failure to follow this instruction can result in injury or equipment damage.

Note: The remote link requires an EIA RS-485 connection and can only run on one communications port at a time.

Hardware Configuration

A remote link must use a minimum 3-wire EIA RS-485 port. It can be configured to use either the first or an optional second port if present.

Note: Only one communication port at time can be configured as a remote link.

The table below lists the devices that can be used:

Remote	Port	Specifications
TWDLCA10/16/24DRF, TWDLCA40DRF, TWDLMDA20/40DUK, TWDLMDA20/40DTK, TWDLMDA20DRT	1	Base controller equipped with a 3-wire EIA RS-485 port with a miniDIN connector.
TWDNOZ485D	2	Communication module equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485T	2	Communication module equipped with a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNAC485D	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485T	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a terminal. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDXCPODM	2	Operator Display expansion module equipped with a 3-wire EIA RS-485 port with a miniDIN connector or a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have a Communication expansion module.

Note: You can only check the presence and configuration (RS232 or RS485) of port 2 at power-up or reset by the firmware executive program.

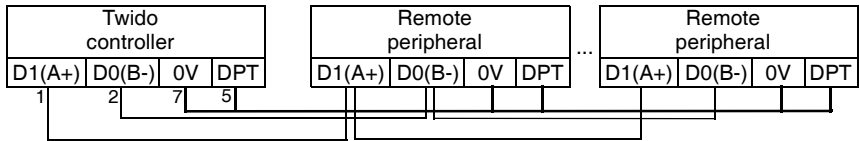
Cable
Connection to
Each Device

Note: The DPT signal on pin 5 must be tied to 0V on pin 7 in order to signify the use of remote link communications. When this signal is not tied to ground, the Twido controller as either the master or slave will default to a mode of attempting to establish communications with TwidoSoft.

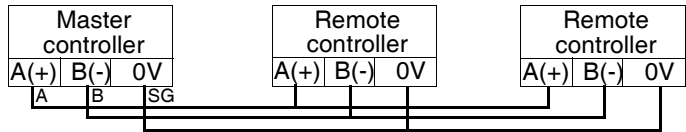
Note: The DPT to 0V connection is only necessary if you are connected to a base controller on Port 1.

The cable connections made to each remote device are shown below.

Mini-DIN connection



Terminal block connection



Software
Configuration

There must be only one master controller defined on the remote link. In addition, each remote controller must maintain a unique slave address. Multiple masters or slaves using identical addresses can either corrupt transmissions or create ambiguity.

⚠ CAUTION

UNEXPECTED EQUIPMENT OPERATION

Be sure that there is only one master controller on a remote link and that each slave has a unique address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results.

Failure to follow this instruction can result in injury or equipment damage.

Master Controller Configuration

The master controller is configured using TwidoSoft to manage a remote link network of up to seven remote controllers. These seven remote controllers can be configured either as remote I/Os or as peer controllers. The address of the master configured using TwidoSoft corresponds to address 0.

To configure a controller as a Master Controller, use TwidoSoft to configure port 1 or port 2 as remote links and select the address 0 (Master).

Then, from the "Add remote controller" window, you can specify the slave controllers either as remote I/O, or as peer controllers, as well as their addresses.

Remote Controller Configuration

A remote controller is configured using TwidoSoft, by setting port 1 or 2 as a remote link or by assigning the port an address from 1 to 7.

The table below summarizes the differences and constraints of each of these types of remote controller configurations:

Type	Application Program	Data Access
Remote I/O	No Not even a simple "END" statement RUN mode is coupled to the Master's.	%I and %Q Only the local I/O of the controller is accessible (and not its I/O extension).
Peer controller	Yes Run mode is independent of the Master's.	%INW and %QNW A maximum of 4 input words and 4 output words can be transmitted to and from each peer.

Remote Controller Scan Synchronization

The update cycle of the remote link is not synchronized with the master controller's scan. The communications with the remote controllers is interrupt driven and happens as a background task in parallel with the running of the master controller's scan. At the end of the scan cycle, the most up to date values are read into the application data to be used for the next program execution. This processing is the same for remote I/O and peer controllers.

Any controller can check for general link activity using system bit %S111. But to achieve synchronization, a master or peer will have to use system bit %S110. This bit is set to 1 when a complete update cycle has taken place. The application program is responsible for resetting this to 0.

The master can enable or disable the remote link using system bit %S112.

Controllers can check on the proper configuration and correct operation of the remote link using %S113. The DPT signal on Port 1 (used to determine if TwidoSoft is connected) is sensed and reported on %S100.

All these are summarized in the following table:

System Bit	Status	Indication
%S100	0	master/slave: DPT not active (TwidoSoft cable NOT connected)
	1	master/slave: DPT active (TwidoSoft cable connected)
%S110	0	master/slave: set to 0 by the application
	1	master: all remote link exchanges completed (remote I/O only) slave: exchange with master completed
%S111	0	master: single remote link exchange completed slave: single remote link exchange detected
	1	master: single remote link exchange in progress slave: single remote link exchange detected
%S112	0	master: remote link disabled
	1	master: remote link enabled
%S113	0	master/slave: remote link configuration/operation OK
	1	master: remote link configuration/operation error slave: remote link operation error

**Master
Controller
Restart**

If a master controller restarts, one of the following events happens:

- A cold start (%S0 = 1) forces a re-initialization of the communications.
- A warm start (%S1 = 1) forces a re-initialization of the communications.
- In Stop mode, the master continues communicating with the slaves.

**Slave Controller
Restart**

If a slave controller restarts, one of the following events happens:

- A cold start (%S0 = 1) forces a re-initialization of the communications.
- A warm start (%S1 = 1) forces a re-initialization of the communications.
- In Stop mode, the slave continues communicating with the master. If the master indicates a Stop state:
 - The remote I/Os apply a Stop state.
 - A peer controller continues in its current state.

**Master
Controller Stop**

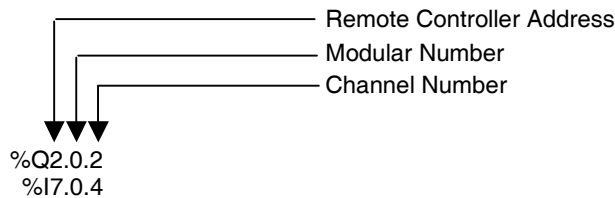
When the master controller enters Stop mode, all slave devices continue communicating with the master. When the master indicates a Stop is requested, then a remote I/O controller will Stop, but peer controllers will continue in their current Run or Stop state.

Remote I/O Data Access

The remote controller configured to be a remote I/O does not have or execute its own application program. The remote controller's base digital inputs and outputs are a simple extension of the master controller's. The application must only use the full three digit addressing mechanism provided.

Note: The module number is always zero for remote I/O.

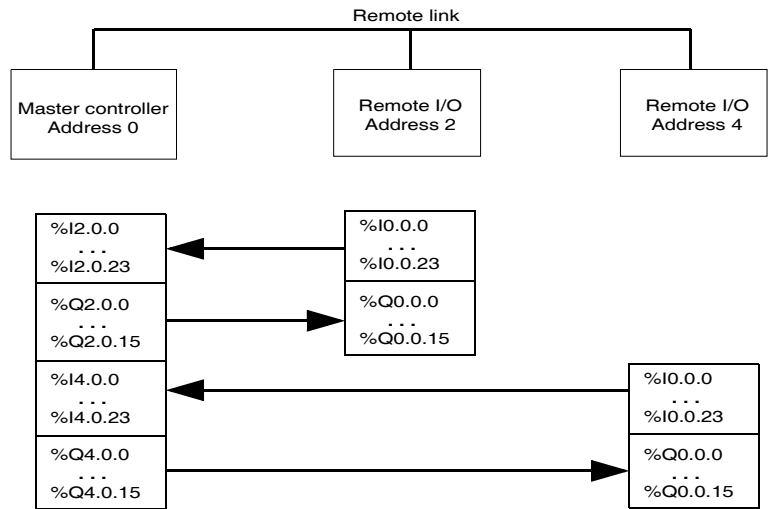
Illustration



To communicate with remote I/O, the master controller uses the standard input and output notation of %I and %Q. To access the third output bit of the remote I/O configured at address 2, instruction %Q2.0.2 is used. Similarly, to read the fifth input bit of the remote I/O configured at location 7, instruction %I7.0.4 is used.

Note: The master is restricted to accessing only the digital I/O that is part of the remote's local I/O. No analog or expansion I/O can be transferred, unless you use peer communications.

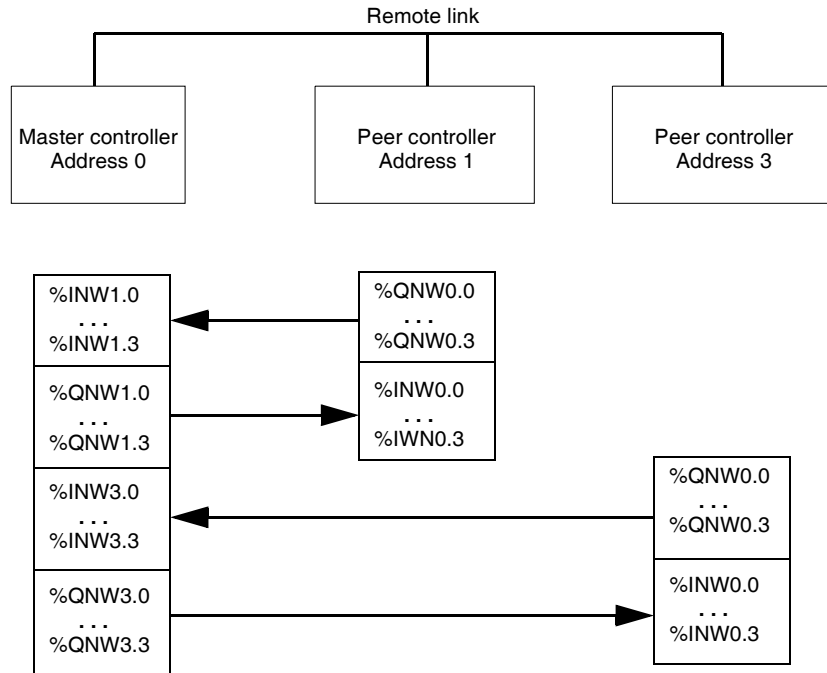
Illustration



Peer Controller Data Access

To communicate with peer controllers, the master uses network words %INW and %QNW to exchange data. Each peer on the network is accessed by its remote address "j" using words %INWj.k and %QNWj.k. Each peer controller on the network uses %INW0.0 to %INW0.3 and %QNW0.0 to %QNW0.3 to access data on the master. Network words are updated automatically when the controllers are in Run or Stop mode.

The example below illustrates the exchange of a master with two configured peer controllers.



There is no peer-to-peer messaging within the remote link. The master application program can be used to manage the network words, in order to transfer information between the remote controllers, in effect using the master as a bridge.

Status Information

In addition to the system bits explained earlier, the master maintains the presence and configuration status of remote controllers. This action is performed in system words %SW111 and %SW113. Either the remote or the master can acquire the value of the last error that occurred while communicating on the remote link in system word %SW112.

System Words	Use	
%SW111	Remote link status: two bits for each remote controller (master only)	
	x0-6	0-Remote controller 1-7 not present
		1-Remote controller 1-7 present
	x8-14	0-Remote I/O detected at Remote controller 1-7
		1-Peer controller detected at Remote controller 1-7
%SW112	Remote Link configuration/operation error code	
		0 - operations are successful
		1 - timeout detected (slave)
		2 - checksum error detected (slave)
		3 - configuration mismatch (slave)
%SW113	Remote link configuration: two bits for each remote controller (master only)	
	x0-6	0-Remote controller 1-7 not configured
		1-Remote controller 1-7 configured
	x8-14	0-Remote I/O configured as remote controller 1-7
		1-Peer controller configured as remote controller 1-7

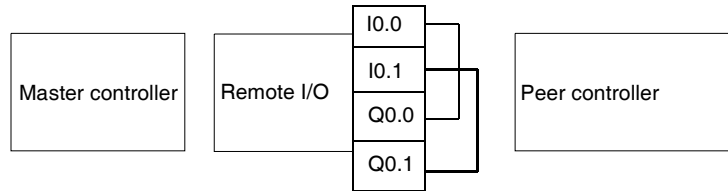
Remote Link Example

To configure a Remote Link, you must:

1. Configure the hardware.
2. Wire the controllers.
3. Connect the communications cable between the PC to the controllers.
4. Configure the software.
5. Write an application.

The diagrams below illustrate the use of the remote link with remote I/O and a peer controller.

Step 1: Configure the Hardware:

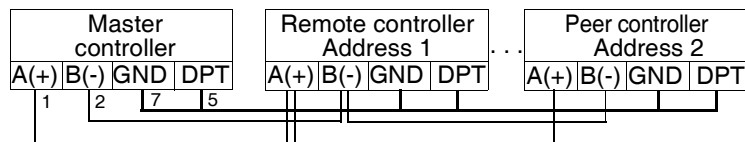


The hardware configuration is three base controllers of any type. Port 1 is used for two communication modes. One mode is to configure and transfer the application program with TwidoSoft. The second mode is for the Remote Link network. If available, an optional Port 2 on any of the controllers can be used, but a controller only supports a single Remote Link.

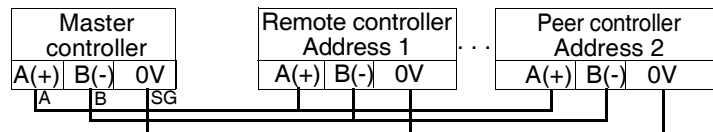
Note: In this example, the two first inputs on the Remote I/O are hard wired to the first two outputs.

Step 2: Wire the controllers

Mini-DIN connection

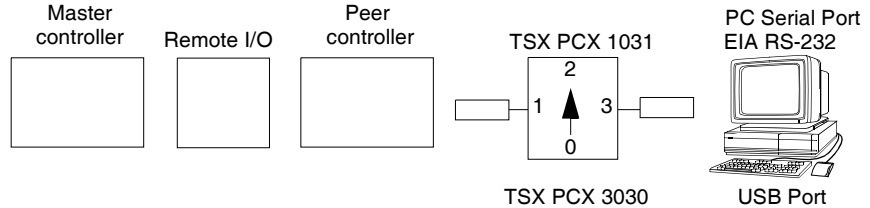


Terminal block connection



Connect the A(+) and B(-) signal wires together. And at each controller, the DPT signal is tied to ground. Although tying the signal to the ground is not required for use with a remote link on Port 2 (optional cartridge or communication module), it is good practice.

Step 3: Connect the Communications Cable between the PC and Controllers:



The TSXPCX1031 or TSX PCX 3030 multi-function programming cable is used to communicate with each of the three base controllers. Be sure that the cable is on switch position 2. In order to program each of the controllers, a point-to-point communication with each controller will need to be established. To establish this communication: connect to Port 1 of the first controller, transfer the configuration and application data, and set the controller to the run state. Repeat this procedure for each controller.

Note: The cable needs to be moved after each controller configuration and application transfer.

Step 4: Configure the Software:

Each of the three controllers uses TwidoSoft to create a configuration, and if appropriate, the application program.

For the master controller, edit the controller communication setup to set the protocol to "Remote Link" and the Address to "0 (Master)".

Controller comm. settings

Type: Remote link
Address: 0 (Master)

Configure the remote controller on the master by adding a "Remote I/O" at address "1" and a "Peer PLC" at address "2".

Add Remote Controllers

Controller Usage: Remote I/O
Remote Address: 1

Controller Usage: Peer controller
Remote Address: 2

For the controller configured as a remote I/O, verify that the controller communication setup is set to "Remote Link" and the address is set to "1".

Controller comm. settings

Type: Remote link
Address: 1

For the controller configured as peer, verify that the controller communication setup is set to "Remote Link" and the address is set to "2".

Controller comm. settings

Type: Remote link
Address: 2

Step 5: Write the applications:

For the Master controller, write the following application program:

```
LD 1

[%MW0 := %MW0 + 1]
[%QNW2.0 := %MW0]
[%MW1 := %INW2.0]

LD %I0.0
ST %Q1.00.0
LD %I1.0.0
ST %Q0.0

LD %I0.1
ST %Q1.0.1
LD %I1.0.1
ST %Q0.1
```

For the controller configured as a remote I/O, do not write any application program.
For the controller configured as peer, write the following application:

```
LD 1
[%QNW0.0 := %INW0.0]
```

In this example, the master application increments an internal memory word and communicates this to the peer controller using a single network word. The peer controller takes the word received from the master and echoes it back. In the master, a different memory word receives and stores this transmission.

For communication with the remote I/O controller, the master sends its local inputs to the remote I/O's outputs. With the external I/O hard wiring of the remote I/O, the signals are returned and retrieved by the master.

ASCII Communications

Introduction

ASCII protocol provides Twido controllers a simple half-duplex character mode protocol to transmit and/or receive data with a simple device. This protocol is supported using the EXCHx instruction and controlled using the %MSGx function block.

Three types of communications are possible with the ASCII Protocol:

- Transmission Only
- Transmission/Reception
- Reception Only

The maximum size of frames transmitted and/or received using the EXCHx instruction is 256 bytes.

Hardware Configuration

An ASCII link (see system bits %S103 and %S104 (See *System Bits (%S)*, p. 596)) can be established on either the EIA RS-232 or EIA RS-485 port and can run on as many as two communications ports at a time.

The table below lists the devices that can be used:

Remote	Port	Specifications
TWDLCA10/16/24DRF, TWDLCA40DRF, TWDLMDA20/40DTK, TWDLMDA20DRT	1	Base controller equipped with a 3-wire EIA RS-485 port with a miniDIN connector.
TWDNOZ232D	2	Communication module equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485D	2	Communication module equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485T	2	Communication module equipped with a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNAC232D	2	Communication adapter equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485D	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485T	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a terminal. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDXCPODM	2	Operator Display expansion module equipped with a 3-wire EIA RS-232 port with a miniDIN connector, a 3-wire EIA RS-485 port with a miniDIN connector and a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have a Communication expansion module.

Note: You can only check the presence and configuration (RS232 or RS485) of port 2 at power-up or reset by the firmware executive program.

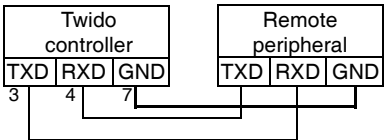
Nominal Cabling Nominal cable connections are illustrated below for both the EIA RS-232 and the EIA RS-485 types.

Note: If port 1 is used on the Twido controller, the DPT signal on pin 5 must be tied to 0V on pin 7. This signifies to the Twido controller that the communications through port 1 is ASCII and is not the protocol used to communicate with the TwidoSoft software.

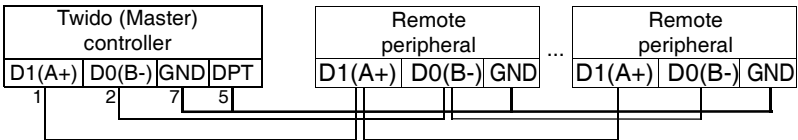
Cable connections to each device are illustrated below.

Mini-DIN connection

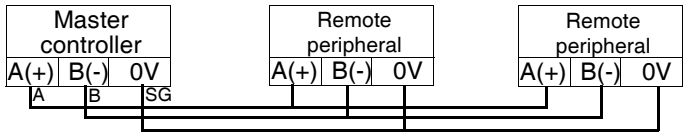
RS-232 EIA cable



RS-485 EIA cable



Terminal block connection



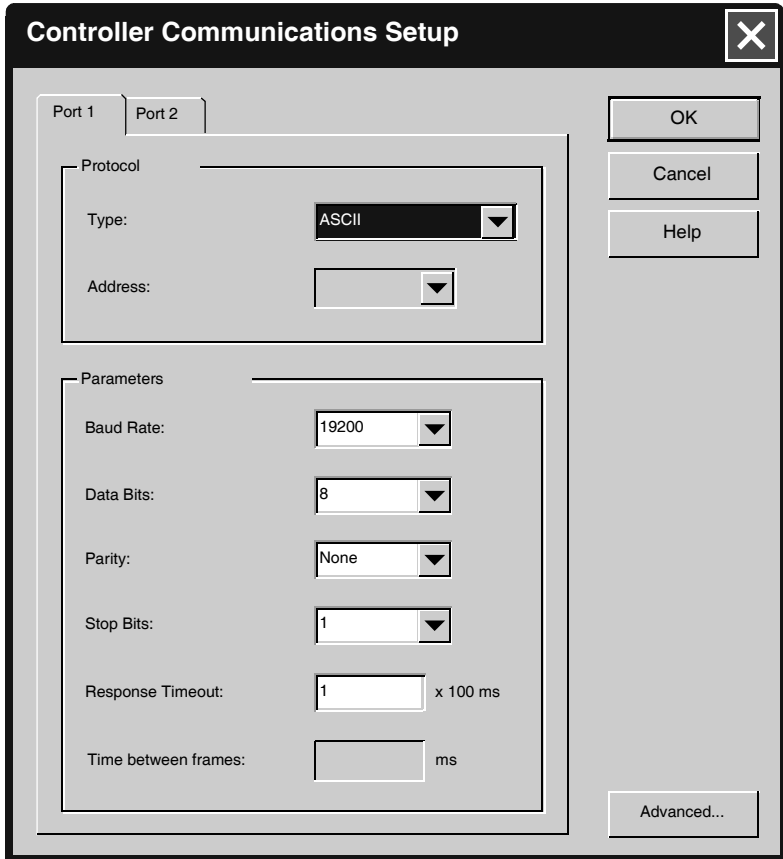
Software Configuration

To configure the controller to use a serial connection to send and receive characters using the ASCII protocol, you must:

Step	Description
1	Configure the serial port for ASCII using TwidoSoft.
2	Create in your application a transmission/reception table that will be used by the EXCHx instruction.

Configuring the Port

A Twido controller can use its primary port 1 or an optionally configured port 2 to use the ASCII protocol. To configure a serial port for ASCII:

Step	Action
1	Define any additional communication adapters or modules configured to the base.
2	Right click the port in the Application browser and select Edit Controller Comm Setup... Result: The following window opens. 
3	Select ASCII serial port type out of Protocol Type listbox.
4	Set the associated communication parameters.
5	Click Advanced button to set the advanced parameters.

**Configuring the
Transmission/
Reception table
for ASCII mode**

The maximum size of the transmitted and/or received frames is 256 bytes. The word table associated with the EXCHx instruction is composed of the transmission and reception control tables.

	Most significant byte	Least significant byte
Control table	Command	Length (transmission/reception)
	Reserved (0)	Reserved (0)
Transmission table	Transmitted Byte 1	Transmitted Byte 2
	...	...
	Transmitted Byte n+1	Transmitted Byte n
Reception table	Received Byte 1	Received Byte 2
	...	...
	Received Byte p+1	Received Byte p

Control table

The **Length** byte contains the length of the transmission table in bytes (250 max.), which is overwritten by the number of characters received at the end of the reception, if reception is requested.

The **Command** byte must contain one of the following:

- 0: Transmission only
- 1: Send/receive
- 2: Reception Only

**Transmission/
reception tables**

When in Transmit Only mode, the Control and Transmission tables are filled in prior to executing the EXCHx instruction, and can be of type %KW or %MW. No space is required for the reception of characters in Transmission only mode. Once all bytes are transmitted, %MSGx.D is set to 1, and a new EXCHx instruction can be executed. When in Transmit/Receive mode, the Control and Transmission tables are filled in prior to executing the EXCHx instruction, and must be of type %MW. Space for up to 256 reception bytes is required at the end of the Transmission table. Once all bytes are transmitted, the Twido controller switches to reception mode and waits to receive any bytes.

When in Reception only mode, the Control table is filled in prior to executing the EXCHx instruction, and must be of type %MW. Space for up to 256 reception bytes is required at the end of the Control table. Twido controller immediately enters the reception mode and waits to receive any bytes.

Reception ends when end of frame bytes used have been received, or the Reception table is full. In this case, an error (receive table overflowed) appears in the word %SW63 and %SW64. If a non-zero timeout is configured, reception ends when the timeout is completed. If a zero timeout value is selected, there is no reception timeout. Therefore to stop reception, %MSGx.R input must be activated.

**Message
Exchange**

The language offers two services for the communication:

- **EXCHx instruction:** to transmit/receive messages
- **%MSGx Function Block:** to control the message exchanges.

Twido controller uses the protocol configured for that port when processing an EXCHx instruction.

Note: Each communications port can be configured for different protocols or the same. The EXCHx instruction or %MSGx function block for each communications port is accessed by appending the port number (1 or 2).

**EXCHx
Instruction**

The EXCHx instruction allows the Twido controller to send and/or receive information to/from ASCII devices. The user defines a table of words (%MWi:L or %KW:L) containing control information and the data to be sent and/or received (up to 256 bytes in transmission and/or reception). The format for the word table is described earlier. A message exchange is performed using the EXCHx instruction:

Syntax: [EXCHx %MWi:L]

where: x = port number (1 or 2)

L = number of words in the control words and transmission and
reception tables

The Twido controller must finish the exchange from the first EXCHx instruction before a second can be launched. The %MSGx function block must be used when sending several messages.

The processing of the EXCHx list instruction occurs immediately, with any transmissions started under interrupt control (reception of data is also under interrupt control), which is considered background processing.

%MSGx Function Block

The use of the %MSGx function block is optional; it can be used to manage data exchanges. The %MSGx function block has three purposes:

- **Communications error checking**

The error checking verifies that the parameter L (length of the Word table) programmed with the EXCHx instruction is large enough to contain the length of the message to be sent. This is compared with the length programmed in the least significant byte of the first word of the word table.

- **Coordination of multiple messages**

To ensure the coordination when sending multiple messages, the %MSGx function block provides the information required to determine when transmission of a previous message is complete.

- **Transmission of priority messages**

The %MSGx function block allows current message transmissions to be stopped in order to allow the immediate sending of an urgent message.

The %MSGx function block has one input and two outputs associated with it:

Input/Output	Definition	Description
R	Reset input	Set to 1: re-initializes communication or resets block (%MSGx.E = 0 and %MSGx.D = 1).
%MSGx.D	Communication complete	0: Request in progress. 1: communication done if end of transmission, end character received, error, or reset of block.
%MSGx.E	Error	0: message length OK and link OK. 1: if bad command, table incorrectly configured, incorrect character received (speed, parity, and so on.), or reception table full.

Limitations

It is important to note the following limitations:

- Port 2 availability and type (see %SW7) is checked only at power-up or reset
- Any message processing on Port 1 is aborted when the TwidoSoft is connected
- EXCHx or %MSG can not be processed on a port configured as Remote Link
- EXCHx aborts active Modbus Slave processing
- Processing of EXCHx instructions is not re-tried in the event of an error
- Reset input (R) can be used to abort EXCHx instruction reception processing
- EXCHx instructions can be configured with a time out to abort reception
- Multiple messages are controlled via %MSGx.D

**Error and
Operating Mode
Conditions**

If an error occurs when using the EXCHx instruction, bits %MSGx.D and %MSGx.E are set to 1 and system word %SW63 contains the error code for Port 1, and %SW64 contains the error code for Port 2.

System Words	Use
%SW63	EXCH1 error code: 0 - operation was successful 1 - number of bytes to be transmitted is too great (> 250) 2 - transmission table too small 3 - word table too small 4 - receive table overflowed 5 - time-out elapsed 6 - transmission error 7 - bad command within table 8 - selected port not configured/available 9 - reception error 10 - cannot use %KW if receiving 11 - transmission offset larger than transmission table 12 - reception offset larger than reception table 13 - controller stopped EXCH processing
%SW64	EXCH2 error code: See %SW63.

**Consequence of
Controller
Restart on the
Communication**

If a controller restarts, one of the following events happens:

- A cold start (%S0 = 1) forces a re-initialization of the communications.
- A warm start (%S1 = 1) forces a re-initialization of the communications.
- In Stop, the controller stops all ASCII communications.

**ASCII Link
Example**

To configure an ASCII Link, you must:

1. Configure the hardware.
2. Connect the ASCII communications cable.
3. Configure the port.
4. Write an application.
5. Initialize the Animation Table Editor.

The diagram below illustrates the use of the ASCII communications with a Terminal Emulator on a PC.

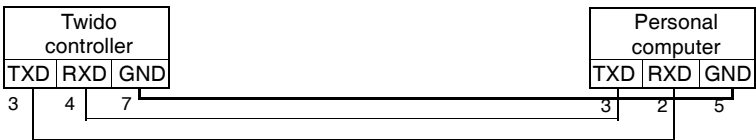
Step 1: Configure the Hardware:



The hardware configuration is two serial connections from the PC to a Twido controller with an optional EIA RS-232 Port 2. On a Modular controller, the optional Port 2 is a TWDNOZ232D or a TWDNAC232D in the TWDXCPODM. On the Compact controller, the optional Port 2 is a TWDNAC232D.

To configure the controller, connect the TSXPCX1031 cable (not shown) to Port 1 of the Twido controller. Next, connect the cable to the COM 1 port of the PC. Be sure that the switch is in position 2. Finally, connect the COM 2 port of the PC to the optional EIA RS-232 Port 2 on the Twido controller. The wiring schematic is provided in the next step.

Step 2: ASCII Communications Cable (EIA RS-232) Wiring Schematic:



The minimum number of wires used in an ASCII communications cable is 3. Cross the transmit and receive signals.

Note: On the PC side of the cable, additional connections (such as Data Terminal Ready and Data Set Ready) may be needed to satisfy the handshaking. No additional connections are required to satisfy the Twido controller.

Step 3: Port Configuration:

Hardware -> Add Option TWDNOZ232D	
Serial Port 2	
Protocol	ASCII
Address	
Baud Rate	19200
Data Bits	8
Parity	None
Stop Bit	1
Response Timeout (x100ms)	100
Time between frames (ms)	
Start character	
1st end character	65
2nd end character	
Stop on silence (ms)	
Stop on the number of received bytes	

Terminal Emulator on a PC	
Port:	COM2
Baud Rate:	19200
Data:	8 Bit
Parity:	None
Stop:	1 Bit
Flow control:	None

Use a simple Terminal Emulator application on the PC to configure the COM2 port and to ensure that there is no flow control.

Use TwidoSoft to configure the controller's port. First, the hardware option is configured. In this example, the TWDNOZ232D is added to the Modular base controller.

Second, the Controller Communication Setup is initialized with all of the same parameter settings as the Terminal Emulator on the PC. In this example, capital letter "A" is chosen for "1st end character", in order to terminate character reception. A 10 second timeout is chosen for "Response Timeout" parameter. Only one of these two parameters will be invoked, depending on whichever happens first.

Step 4: Write the application:

```
LD 1
[%MW10 := 16#0104 ]
[%MW11 := 16#0000 ]
[%MW12 := 16#4F4B ]
[%MW13 := 16#0A0D ]
LD 1
AND %MSG2.D
[EXCH2 %MW10:8]
LD %MSG2.E
ST %Q0.0
END
```


Use TwidoSoft to create an application program with three primary parts. First, initialize the Control and Transmission tables to use for the EXCH instruction. In this example, a command is set up to both send and receive data. The amount of data to send will be set to 4 bytes, as defined in the application, followed by the end of frame character used (in this case, the first end character "A"). Start and end characters do not display in the Animation table, where only data characters show up. Anyway, those characters are automatically transmitted or checked at reception (by %SW63 and %SW64), when used.

Next, check the status bit associated with %MSG2 and issue the EXCH2 instruction only if the port is ready. For the EXCH2 instruction, a value of 8 words is specified. There are 2 Control words (%MW10 and %MW11), 2 words to be used for transmit information (%MW12 and %MW13), and 4 words to receive data (%MW14 through %MW16).

Finally, the error status of the %MSG2 is sensed and stored on the first output bit on the local base controller I/O. Additional error checking using %SW64 could also be added to make this more accurate.

Step 5: Initialize the Animation Table Editor:

Address	Current	Retained Format
1 %MW10	0104	Hexadecimal
2 %MW11	0000	Hexadecimal
3 %MW12	4F4B	Hexadecimal
4 %MW13	0A0D	Hexadecimal
5 %MW14	TW	ASCII
6 %MW15	ID	ASCII
7 %MW16	O	ASCII

The final step is to download this application controller and run it. Initialize an Animation Table Editor to animate and display the %MW10 through %MW16 words. On the Terminal Emulator, characters "O- K - CR - LF - A" can be displayed as many times as the EXCH block response timeout has elapsed. On the Terminal Emulator, type in "T - W - I - D - O - A". This information is exchanged with Twido controller and displayed in the Animation Table Editor.

Modbus Communications

Introduction

The Modbus protocol is a master-slave protocol that allows for one, and only one, master to request responses from slaves, or to act based on the request. The master can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a message (response) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

CAUTION

UNEXPECTED EQUIPMENT OPERATION

- Be sure that there is only one Modbus master controller on the bus and that each Modbus slave has a unique address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results.
- Be sure that all Modbus slaves have unique addresses. No two slaves should have the same address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results.

Failure to follow this instruction can result in injury or equipment damage.

Hardware Configuration

A Modbus link can be established on either the EIA RS-232 or EIA RS-485 port and can run on as many as two communications ports at a time. Each of these ports can be assigned its own Modbus address, using system bit %S101 and system words %SW101 and %SW102 (See *System Bits (%S)*, p. 596). . (See also *System Words (%SW)*, p. 604)

The table below lists the devices that can be used:

Remote	Port	Specifications
TWDLCA10/16/24DRF, TWDLCA40DRF, TWDLMDA20/40DTK, TWDLMDA20DRT	1	Base controller supporting a 3-wire EIA RS-485 port with a miniDIN connector.
TWDNOZ232D	2	Communication module equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485D	2	Communication module equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485T	2	Communication module equipped with a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNAC232D	2	Communication adapter equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485D	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485T	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a terminal connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDXCPODM	2	Operator Display expansion module equipped with a 3-wire EIA RS-232 port with a miniDIN connector, a 3-wire EIA RS-485 port with a miniDIN connector and a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have a Communication expansion module.

Note: The presence and configuration (RS232 or RS485) of Port 2 is checked at power-up or at reset by the firmware executive program.

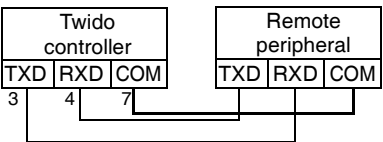
Nominal Cabling Nominal cable connections are illustrated below for both the EIA RS-232 and the EIA RS-485 types.

Note: If port 1 is used on the Twido controller, the DPT signal on pin 5 must be tied to the circuit common (COM) on pin 7. This signifies to the Twido controller that the communications through port 1 is Modbus and is not the protocol used to communicate with the TwidoSoft software.

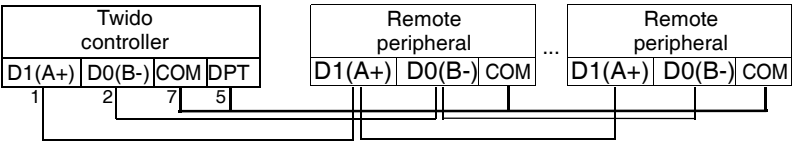
The cable connections made to each remote device are shown below.

Mini-DIN connection

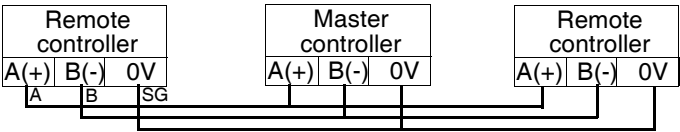
RS-232 EIA cable



RS-485 EIA cable



Terminal block connection



EIA RS-485 Line Polarization on TWDLCA•40DRF Controllers

There is no internal pre-polarization in TWDLCA•40DRF controllers. Therefore, external line polarization is required when connecting the TWDLCA•40DRF Modbus master controller to the EIA-485 Modbus network. (When there is no data activity on an EIA-485 balanced pair, the lines are not driven and, therefore, susceptible to external noise or interference. To ensure that its receiver stays in a constant state, when no data signal is present, the Modbus master device needs to bias the network via external line polarization.)

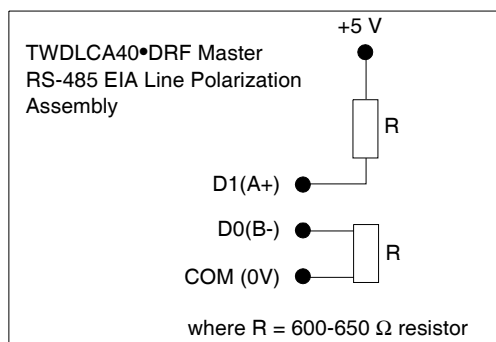
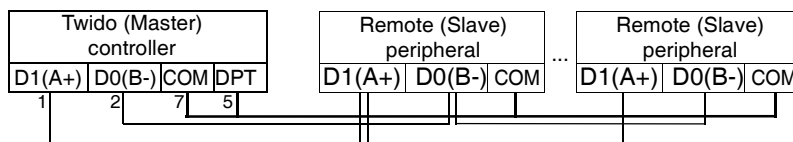
Note: EIA RS-485 external line polarization must be implemented on the Modbus Master controller only; you must not implement it on any slave device.

The external line polarization assembly on the TWDLCA•40DRF mini-DIN RS-485 EIA line shall consist in:

- One pull-up resistor to a 5V voltage on D1(A+) circuit,
- One pull-down resistor to the common circuit on D0(B-) circuit.

The following figure illustrates the external line polarization assembly on the TWDLCA•40DRF mini-DIN RS-485 EIA line:

Mini-DIN connection
RS-485 EIA cable



External polarization can be performed in any of two ways:

- Connecting externally the user-provided polarization assembly via mini-DIN fly cable. (Please refer to pin definition for connector.)
- Using a polarization tap (configured for 2-wire polarization) and polarization assembly (available soon from the catalog).

**Software
Configuration**

To configure the controller to use a serial connection to send and receive characters using the Modbus protocol, you must:

Step	Description
1	Configure the serial port for Modbus using TwidoSoft.
2	Create in your application a transmission/reception table that will be used by the EXCHx instruction.

**Configuring the
Port**

A Twido controller can use its primary port 1 or an optionally configured port 2 to use the Modbus protocol. To configure a serial port for Modbus:

Step	Action
1	Define any additional communication adapters or modules configured to the base.
2	Right-click on the port and click Edit Controller Comm Setup... and change serial port type to "Modbus".
3	Set the associated communication parameters.

Modbus Master

Modbus master mode allows the controller to send a Modbus query to a slave, and to wait for the response. The Modbus Master mode is only supported via the EXCHx instruction. Both Modbus ASCII and RTU are supported in Modbus Master mode. The maximum size of the transmitted and/or received frames is 250 bytes. Moreover, the word table associated with the EXCHx instruction is composed of the control, transmission and reception tables.

	Most significant byte	Least significant byte
Control table	Command	Length (Transmission/ Reception)
	Reception offset	Transmission offset
Transmission table	Transmitted Byte 1	Transmitted Byte 2
	...	...
	...	Transmitted Byte n
	Transmitted Byte n+1	
Reception table	Received Byte 1	Received Byte 2
	...	...
	...	Received Byte p
	Received Byte p+1	

Note: In addition to queries to individual slaves, the Modbus master controller can initiate a **broadcast** query to all slaves. The **command** byte in case of a broadcast query must be set to 00, while the **slave address** must be set to 0.

Control table

The **Length** byte contains the length of the transmission table (maximum 250 bytes), which is overwritten by the number of characters received at the end of the reception, if reception is requested.

This parameter is the length in bytes of the transmission table. If the Tx Offset parameter is equal to 0, this parameter will be equal to the length of the transmission frame. If the Tx Offset parameter is not equal to 0, one byte of the transmission table (indicated by the offset value) will not be transmitted and this parameter is equal to the frame length itself plus 1.

The **Command** byte in case of Modbus RTU request (except for broadcast) must always equal to 1 (Tx and Rx).

The **Tx Offset** byte contains the rank (1 for the first byte, 2 for the second byte, and so on) within the Transmission Table of the byte to ignore when transmitting the bytes. This is used to handle the issues associated with byte/word values within the Modbus protocol. For example, if this byte contains 3, the third byte would be ignored, making the fourth byte in the table the third byte to be transmitted.

The **Rx Offset** byte contains the rank (1 for the first byte, 2 for the second byte, and so on) within the Reception Table to add when transmitting the packet. This is used to handle the issues associated with byte/word values within the Modbus protocol. For example, if this byte contains 3, the third byte within the table would be filled with a ZERO, and the third byte was actually received would be entered into the fourth location in the table.

**Transmission/
reception tables**

When using either mode (Modbus ASCII or Modbus RTU), the Transmission table is filled with the request prior to executing the EXCHx instruction. At execution time, the controller determines what the Data Link Layer is, and performs all conversions necessary to process the transmission and response. Start, end, and check characters are not stored in the Transmission/Reception tables.

Once all bytes are transmitted, the controller switches to reception mode and waits to receive any bytes.

Reception is completed in one of several ways:

- timeout on a character or frame has been detected,
- end of frame character(s) received in ASCII mode,
- the Reception table is full.

Transmitted byte X entries contain Modbus protocol (RTU encoding) data that is to be transmitted. If the communications port is configured for Modbus ASCII, the correct framing characters are appended to the transmission. The first byte contains the device address (specific or broadcast), the second byte contains the function code, and the rest contain the information associated with that function code.

Note: This is a typical application, but does not define all the possibilities. No validation of the data being transmitted will be performed.
--

Received Bytes X contain Modbus protocol (RTU encoding) data that is to be received. If the communications port is configured for Modbus ASCII, the correct framing characters are removed from the response. The first byte contains the device address, the second byte contains the function code (or response code), and the rest contain the information associated with that function code.

Note: This is a typical application, but does not define all the possibilities. No validation of the data being received will be performed, except for checksum verification.

Modbus Slave

Modbus slave mode allows the controller to respond to standard Modbus queries from a Modbus master.

When TSXPCX1031 cable is attached to the controller, TwidoSoft communications are started at the port, temporarily disabling the communications mode that was running prior to the cable being connected.

The Modbus protocol supports two Data Link Layer formats: ASCII and RTU. Each is defined by the Physical Layer implementation, with ASCII using 7 data bits, and RTU using 8 data bits.

When using Modbus ASCII mode, each byte in the message is sent as two ASCII characters. The Modbus ASCII frame begins with a start character (':'), and can end with two end characters (CR and LF). The end of frame character defaults to 0x0A (line feed), and the user can modify the value of this byte during configuration. The check value for the Modbus ASCII frame is a simple two's complement of the frame, excluding the start and end characters.

Modbus RTU mode does not reformat the message prior to transmitting; however, it uses a different checksum calculation mode, specified as a CRC.

The Modbus Data Link Layer has the following limitations:

- Address 1-247
- Bits: 128 bits on request
- Words: 125 words of 16 bits on request

Message Exchange

The language offers two services for communication:

- **EXCHx instruction:** to transmit/receive messages
- **%MSGx Function Block:** to control the message exchanges.

The Twido controller uses the protocol configured for that port when processing an EXCHx instruction.

Note: Each communications port can be configured for different protocols or the same. The EXCHx instruction or %MSGx function block for each communications port is accessed by appending the port number (1 or 2).

**EXCHx
Instruction**

The EXCHx instruction allows the Twido controller to send and/or receive information to/from Modbus devices. The user defines a table of words (%MWi:L) containing control information and the data to be sent and/or received (up to 250 bytes in transmission and/or reception). The format for the word table is described earlier.

A message exchange is performed using the EXCHx instruction:

Syntax: [EXCHx %MWi:L]

where: x = port number (1 or 2)

L = number of words in the control words, transmission and reception tables

The Twido controller must finish the exchange from the first EXCHx instruction before a second can be launched. The %MSGx function block must be used when sending several messages.

The processing of the EXCHx list instruction occurs immediately, with any transmissions started under interrupt control (reception of data is also under interrupt control), which is considered background processing.

%MSGx Function Block

The use of the %MSGx function block is optional; it can be used to manage data exchanges. The %MSGx function block has three purposes:

- **Communications error checking**

The error checking verifies that the parameter L (length of the Word table) programmed with the EXCHx instruction is large enough to contain the length of the message to be sent. This is compared with the length programmed in the least significant byte of the first word of the word table.

- **Coordination of multiple messages**

To ensure the coordination when sending multiple messages, the %MSGx function block provides the information required to determine when transmission of a previous message is complete.

- **Transmission of priority messages**

The %MSGx function block allows current message transmissions to be stopped in order to allow the immediate sending of an urgent message.

The %MSGx function block has one input and two outputs associated with it:

Input/Output	Definition	Description
R	Reset input	Set to 1: re-initializes communication or resets block (%MSGx.E = 0 and %MSGx.D = 1).
%MSGx.D	Communication complete	0: request in progress. 1: communication done if end of transmission, end character received, error, or reset of block.
%MSGx.E	Error	0: message length OK and link OK. 1: if bad command, table incorrectly configured, incorrect character received (speed, parity, and so on.), or reception table full.

Limitations

It is important to note the following limitations:

- Port 2 presence and configuration (RS232 or RS485) is checked at power-up or reset
- Any message processing on Port 1 is aborted when the TwidoSoft is connected
- EXCHx or %MSG can not be processed on a port configured as Remote Link
- EXCHx aborts active Modbus Slave processing
- Processing of EXCHx instructions is not re-tried in the event of an error
- Reset input (R) can be used to abort EXCHx instruction reception processing
- EXCHx instructions can be configured with a time out to abort reception
- Multiple messages are controlled via %MSGx.D

**Error and
Operating Mode
Conditions**

If an error occurs when using the EXCHx instruction, bits %MSGx.D and %MSGx.E are set to 1 and system word %SW63 contains the error code for Port 1, and %SW64 contains the error code for Port 2.

System Words	Use
%SW63	EXCH1 error code: 0 - operation was successful 1 - number of bytes to be transmitted is too great (> 250) 2 - transmission table too small 3 - word table too small 4 - receive table overflowed 5 - time-out elapsed 6 - transmission 7 - bad command within table 8 - selected port not configured/available 9 - reception error 10 - can not use %KW if receiving 11 - transmission offset larger than transmission table 12 - reception offset larger than reception table 13 - controller stopped EXCH processing
%SW64	EXCH2 error code: See %SW63.

**Master
Controller
Restart**

If a master/slave controller restarts, one of the following events happens:

- A cold start (%S0 = 1) forces a re-initialization of the communications.
- A warm start (%S1 = 1) forces a re-initialization of the communications.
- In Stop mode, the controller stops all Modbus communications.

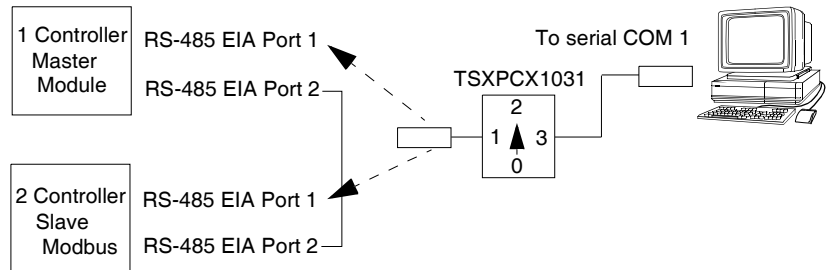
Modbus Link Example 1

To configure a Modbus Link, you must:

1. Configure the hardware.
2. Connect the Modbus communications cable.
3. Configure the port.
4. Write an application.
5. Initialize the Animation Table Editor.

The diagrams below illustrate the use of Modbus request code 3 to read a slave's output words. This example uses two Twido Controllers.

Step 1: Configure the Hardware:



The hardware configuration is two Twido controllers. One will be configured as the Modbus Master and the other as the Modbus Slave.

Note: In this example, each controller is configured to use EIA RS-485 on Port 1 and an optional EIA RS-485 Port 2. On a Modular controller, the optional Port 2 can be either a TWDNOZ485D or a TWDNOZ485T, or if you use TWDXCPODM, it can be either a TWDNAC485D or a TWDNAC485T. On a Compact controller, the optional Port 2 can be either a TWDNAC485D or a TWDNAC485T.

To configure each controller, connect the TSXPCX1031 cable to Port 1 of the controller.

Note: The TSXPCX1031 can only be connected to one controller at a time, on RS-485 EIA port 1 only.

Next, connect the cable to the COM 1 port of the PC. Be sure that the cable is in switch position 2. Download and monitor the application. Repeat procedure for second controller.

Step 2: Connect the Modbus Communications Cable:

Mini-DIN connection



Terminal block connection



The wiring in this example demonstrates a simple point to point connection. The three signals D1(A+), D0(B-), and COM(0V) are wired according to the diagram. If using Port 1 of the Twido controller, the DPT signal (pin 5) must be tied to circuit common (pin 7). This conditioning of DPT determines if TwidoSoft is connected. When tied to the ground, the controller will use the port configuration set in the application to determine the type of communication.

Step 3: Port Configuration:

Hardware -> Add Option TWDNOZ485-	Hardware -> Add Option TWDNOZ485-
Hardware => Controller Comm. Setting	Hardware => Controller Comm. Setting
Serial Port 2	Serial Port 2
Protocol Modbus	Protocol Modbus
Address 1	Address 2
Baud Rate 19200	Baud Rate 19200
Data Bits 8 (RTU)	Data Bits 8 (RTU)
Parity None	Parity None
Stop Bit 1	Stop Bit 1
Response Timeout (x100ms) 10	Response Timeout (x100ms) 100
Time between frames (ms) 10	Time between frames (ms) 10

In both master and slave applications, the optional EIA RS-485 ports are configured. Ensure that the controller's communication parameters are modified in Modbus protocol and at different addresses.

In this example, the master is set to an address of 1 and the slave to 2. The number of bits is set to 8, indicating that we will be using Modbus RTU mode. If this had been set to 7, then we would be using Modbus-ASCII mode. The only other default modified was to increase the response timeout to 1 second.

Note: Since Modbus RTU mode was selected, the "End of Frame" parameter was ignored.

Step 4: Write the application:

```
LD 1
[%MW0 := 16#0106 ]
[%MW1 := 16#0300 ]
[%MW2 := 16#0203 ]
[%MW3 := 16#0000 ]
[%MW4 := 16#0004 ]
LD 1
AND %MSG2.D
[EXCH2 %MW0:11]
LD %MSG2.E
ST %Q0.0
END
```

```
LD 1
[%MW0 := 16#6566 ]
[%MW1 := 16#6768 ]
[%MW2 := 16#6970 ]
[%MW3 := 16#7172 ]
END
```

Using TwidoSoft, an application program is written for both the master and the slave. For the slave, we simply write some memory words to a set of known values. In the master, the word table of the EXCHx instruction is initialized to read 4 words from the slave at Modbus address 2 starting at location %MW0.

Note: Notice the use of the RX offset set in %MW1 of the Modbus master. The offset of three will add a byte (value = 0) at the third position in the reception area of the table. This aligns the words in the master so that they fall correctly on word boundaries. Without this offset, each word of data would be split between two words in the exchange block. This offset is used for convenience.

Before executing the EXCH2 instruction, the application checks the communication bit associated with %MSG2. Finally, the error status of the %MSG2 is sensed and stored on the first output bit on the local base controller I/O. Additional error checking using %SW64 could also be added to make this more accurate.

Step 5: Initialize the animation table editor in the master:

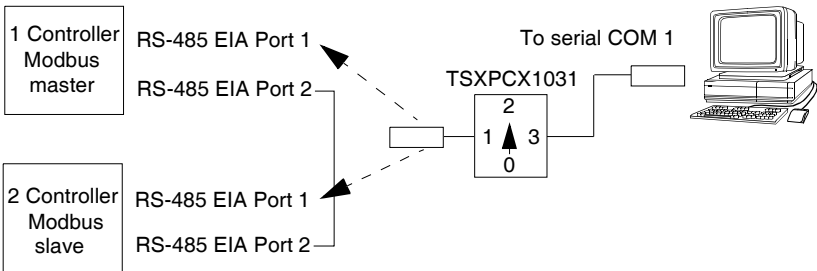
	Address	Current	Retained	Format
1	%MW5	0203	0000	Hexadecimal
2	%MW6	0008	0000	Hexadecimal
3	%MW7	6566	0000	Hexadecimal
4	%MW8	6768	0000	Hexadecimal
5	%MW9	6970	0000	Hexadecimal
6	%MW10	7172	0000	Hexadecimal

After downloading and setting each controller to run, open an animation table on the master. Examine the response section of the table to check that the response code is 3 and that the correct number of bytes was read. Also in this example, note that the words read from the slave (beginning at %MW7) are aligned correctly with the word boundaries in the master.

**Modbus Link
Example 2**

The diagram below illustrates the use of Modbus request 16 to write output words to a slave. This example uses two Twido Controllers.

Step 1: Configure the Hardware:



The hardware configuration is identical to the previous example.

Step 2: Connect the Modbus Communications Cable (RS-485):

Mini-DIN connection



Terminal block connection



The Modbus communications cabling is identical to the previous example.

Step 3: Port Configuration:

Hardware -> Add Option TWDNOZ485-	Hardware -> Add Option TWDNOZ485-
Hardware => Controller Comm. Setting Serial Port 2 Protocol Modbus Address 1 Baud Rate 19200 Data Bits 8 (RTU) Parity None Stop Bit 1 Response Timeout (x100ms) 10 Time between frames (ms) 10	Hardware => Controller Comm. Setting Serial Port 2 Protocol Modbus Address 2 Baud Rate 19200 Data Bits 8 (RTU) Parity None Stop Bit 1 Response Timeout (x100ms) 100 Time between frames (ms) 10

The port configurations are identical to those in the previous example.

Step 4: Write the application:

```
LD 1
[%MW0 := 16#010C ]
[%MW1 := 16#0007 ]
[%MW2 := 16#0210 ]
[%MW3 := 16#0010 ]
[%MW4 := 16#0002 ]
[%MW5 := 16#0004 ]
[%MW6 := 16#6566 ]
[%MW7 := 16#6768 ]
LD 1
AND %MSG2.D
[EXCH2 %MW0:11]
LD %MSG2.E
ST %Q0.0
END
```

```
LD 1
[%MW18 := 16#FFFF ]
END
```

Using TwidoSoft, an application program is created for both the master and the slave. For the slave, write a single memory word %MW18. This will allocate space on the slave for the memory addresses from %MW0 through %MW18. Without allocating the space, the Modbus request would be trying to write to locations that did not exist on the slave.

In the master, the word table of the EXCH2 instruction is initialized to read 4 bytes to the slave at Modbus address 2 at the address %MW16 (10 hexadecimal).

Note: Notice the use of the TX offset set in %MW1 of the Modbus master application. The offset of seven will suppress the high byte in the sixth word (the value 00 hexadecimal in %MW5). This works to align the data values in the transmission table of the word table so that they fall correctly on word boundaries.

Before executing the EXCH2 instruction, the application checks the communication bit associated with %MSG2. Finally, the error status of the %MSG2 is sensed and stored on the first output bit on the local base controller I/O. Additional error checking using %SW64 could also be added to make this more accurate.

Step 5:Initialize the Animation Table Editor:

Create the following animation table on the master:

Address	Current	Retained	Format
1 %MW0	010C	0000	Hexadecimal
2 %MW1	0007	0000	Hexadecimal
3 %MW2	0210	0000	Hexadecimal
4 %MW3	0010	0000	Hexadecimal
5 %MW4	0002	0000	Hexadecimal
6 %MW5	0004	0000	Hexadecimal
7 %MW6	6566	0000	Hexadecimal
8 %MW7	6768	0000	Hexadecimal
9 %MW8	0210	0000	Hexadecimal
10 %MW9	0010	0000	Hexadecimal
11 %MW10	0004	0000	Hexadecimal

Create the following animation table on the slave:

Address	Current	Retained	Format
1 %MW16	6566	0000	Hexadecimal
2 %MW17	6768	0000	Hexadecimal

After downloading and setting each controller to run, open an animation table on the slave controller. The two values in %MW16 and %MW17 are written to the slave. In the master, the animation table can be used to examine the reception table portion of the exchange data. This data displays the slave address, the response code, the first word written, and the number of words written starting at %MW8 in the example above.

Standard Modbus Requests

Introduction

These requests are used to exchange memory words or bits between remote devices. The table format is the same for both RTU and ASCII modes.

Format	Reference number
Bit	%Mi
Word	%MWi

Modbus Master: Read N Bits

The following table represents requests 01 and 02.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	03 (Reception offset)	00 (Transmission offset)
Transmission table	2	Slave@(1..247)	01 or 02 (Request code)
	3	Address of the first bit to read	
	4	N_1 = Number of bits to read	
Reception table (after response)	5	Slave@(1..247)	01 or 02 (Response code)
	6	00 (byte added by Rx Offset action)	N_2 = Number of data bytes to read = $[1+(N_1-1)/8]$, where $[]$ means integral part
	7	Value of the 1 st byte (value = 00 or 01)	Value of the 2 nd byte (if $N_1 > 1$)
	8	Value of the 3 rd byte (if $N_1 > 1$)	
	...		
	$(N_2/2)+6$ (if N_2 is even) $(N_2/2+1)+6$ (if N_2 is odd)	Value of the N_2^{th} byte (if $N_1 > 1$)	

(*) This byte also receives the length of the string transmitted after response

**Modbus Master:
Read N Words**

The following table represents requests 03 and 04.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	03 (Reception Offset)	00 (Transmission offset)
Transmission table	2	Slave@ (1..247)	03 or 04 (Request code)
	3	Address of the first word to read	
	4	N = Number of words to read	
Reception table (after response)	5	Slave@ (1..247)	03 or 04 (Response code)
	6	00 (byte added by Rx Offset action)	2*N (number of bytes read)
	7	First word read	
	8	Second word read (if N>1)	
	...		
	N+6	Word N read (if N>2)	

(*) This byte also receives the length of the string transmitted after response

Note: The Rx offset of three will add a byte (value = 0) at the third position in the reception table. This ensures a good positioning of the number of bytes read and of the read words' values in this table.

Modbus Master: This table represents Request 05.
Write Bit

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	00 (Reception offset)	00 (Transmission offset)
Transmission table	2	Slave@ (1..247)	05 (Request code)
	3	Address of the bit to write	
	4	Bit value to write	
Reception table (after response)	5	Slave@ (1..247)	05 (Response code)
	6	Address of the bit written	
	7	Value written	

(*) This byte also receives the length of the string transmitted after response

Note:

- This request does not need the use of offset.
- The response frame is the same as the request frame here (in a normal case).
- For a bit to write 1, the associated word in the transmission table must contain the value FF00H, and 0 for the bit to write 0.

Modbus Master: This table represents Request 06.
Write Word

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/ reception)	06 (Transmission length) (*)
	1	00 (Reception offset)	00 (Transmission offset)
Transmission table	2	Slave@(1..247)	06 (Request code)
	3	Address of the word to write	
	4	Word value to write	
Reception table (after response)	5	Slave@(1..247)	06 (Response code)
	6	Address of the word written	
	7	Value written	

(*) This byte also receives the length of the string transmitted after response

Note:

- This request does not need the use of offset.
- The response frame is the same as the request frame here (in a normal case).

Modbus Master: This table represents Request 15.
Write of N Bits

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	8 + number of bytes (transmission)
	1	00 (Reception Offset)	07 (Transmission offset)
Transmission table	2	Slave@ (1..247)	15 (Request code)
	3	Number of the first bit to write	
	4	N_1 = Number of bits to write	
	5	00 (byte not sent, offset effect)	N_2 = Number of data bytes to write = $[1+(N_1-1)/8]$, where $[]$ means integral part
	6	Value of the 1 st byte	Value of the 2 nd byte
	7	Value of the 3 rd byte	Value of the 4 th byte
	...		
	$(N_2/2)+5$ (if N_2 is even) $(N_2/2+1)+5$ (if N_2 is odd)	Value of the N_2^{th} byte	
		Slave@ (1..247)	15 (Response code)
		Address of the 1 st bit written	
Reception table (after response)		Address of bits written (= N_1)	

Note:

- The Tx Offset=7 will suppress the 7th byte in the sent frame. This also allows a good correspondence of words' values in the transmission table.

Modbus Master: This table represents Request 16.
Write of N Words

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	8 + (2*N) (Transmission length)
	1	00 (Reception offset)	07 (Transmission offset)
Transmission table	2	Slave@ (1..247)	16 (Request code)
	3	Address of the first word to write	
	4	N = Number of words to write	
	5	00 (byte not sent, offset effect)	2*N = Number of bytes to write
	6	First word value to write	
	7	Second value to write	
	...		
	N+5	N values to write	
Reception table (after response)	N+6	Slave@ (1..247)	16 (Response code)
	N+7	Address of the first word written	
	N+8	Address of words written (= N)	

Note: The Tx Offset = 7 will suppress the 5th MMSB byte in the sent frame. This also allows a good correspondence of words' values in the transmission table.

Transparent Ready Implementation Class (Twido Serial A05, Ethernet A15)

Overview

The following Modbus Function codes are supported by both serial Modbus and TCP/IP Modbus. For detailed information about Modbus protocol, please refer to document *Modbus Application Protocol* which is available at <http://www.modbus-ida.org>

Twido Supported Modbus Function Codes (MB FC)

The following table describes function codes supported by both Twido serial and TCP/IP Modbus:

Supported MB FC	Supported Sub-fc code	Function
1	—	Read multiple internal bits %M
2	—	Read multiple internal bits %M
3	—	Read multiple internal registers %MW
4	—	Read multiple internal registers %MW
5	—	Force single internal bit %M
6	—	Write single internal register %MW
8	00 only	Echo diagnostics
15	—	Write multiple internal bits %M
16	—	Write multiple internal registers %MW
23	—	Read/write multiple internal registers %MW
43	14	Read device identification (regular service)

Ethernet TCP/IP Communications Overview

Ethernet Features

The following information describes the Ethernet-capable features of the Twido TWDLCAE40DRF base controller.

The TWDLCAE40DRF base controller is an Ethernet-capable device that implements the Modbus Application Protocol (MBAP) over TCP/IP. Modbus TCP/IP provides peer-to-peer communications over the network in a client/server topology.

Frame Format

The Twido TWDLCAE40DRF compact controller supports the Ethernet II frame format only. It does not accommodate IEEE802.3 framing. Note that other PLCs available from Schneider Electric, such as the Premium and Quantum series support both Ethernet II and IEEE802.3 frame formats and are frame format selectable. Therefore, if you are planning to team up your Twido controller with Premium or Quantum PLCs, you should configure them as using Ethernet II frame format to allow for optimum compatibility.

TCP Connections

The TWDLCAE40DRF compact controller is a 4-simultaneous-channel device capable of communicating over a 100Base-TX Ethernet network. It implements 100Base-TX auto-negotiation and can work on a 10Base-T network as well. Moreover, it allows one marked IP connection, as configured in the TwidoSoft application program (see *Marked IP Tab, p. 169* for more details about Marked IP). The maximum number of server transactions supported by the Twido controller is 1 per TCP connection.

IP Address

TWDLCAE40DRF controllers implement BootP client support to obtain an IP address from a BootP server. For increased flexibility, you still have the ability to specify a static IP address through TwidoSoft programming software, along with defining the subnetwork and gateway IP addresses.

In addition, if the TWDLCAE40DRF controller fails to obtain a valid IP address from the BootP server (or if it detects a duplicate IP address when you assign a static IP address), the controller goes into fallback mode and uses the default IP address. Each TWDLCAE40DRF controller is assigned a unique MAC physical address (IEEE Global Address) permanently stored in the compact controller. The default IP address is derived from the controller's MAC address.

Note: When using the default IP address, BootP client service is closed.

**Modbus TCP
Client/Server**

A TWDLCAE40DRF controller can be both Modbus TCP/IP Client and Server depending on whether it is querying or answering a remote device, respectively. TCP messaging service is implemented via TCP port 502.

- The Modbus Server implements the Schneider Transparent Ready class messaging TR A15 standard.
 - Modbus Client is implemented via the EXCH3 instruction and %MSG3 function. You may program several EXCH3 instructions, however one EXCH3 only can be active at a time. The TCP connection is automatically negotiated by the compact controller as soon as the
The Modbus Client implements the Schneider Transparent Ready class messaging TR A10 standard.
-

Quick TCP/IP Setup Guide for PC-to-Controller Ethernet Communication

Scope This Quick TCP/IP Setup Guide is intended to provide Ethernet connectivity information and TCP/IP configuration information to rapidly setup communication between your PC running the TwidoSoft application and the Twido Controller over a stand-alone Ethernet network.

Checking the Current IP Settings of your PC The following procedure describes how to check the current IP settings of your PC. Also, this procedure is valid for all versions of the Windows operating system:

Step	Action
1	Select Run from the Windows Start menu.
2	Type " command " in the Open textbox of the Run dialog box. Result: The C:\WINDOWS\system32\command.com prompt appears.
3	Type " ipconfig " at the command prompt.
4	The Windows IP Configuration appears, and displays the following parameters: IP Address.....: Subnet Mask.....: Default Gateway.....: Note: The above IP settings cannot be changed directly at the command prompt. They are available for consultation only. If you plan to change the IP configuration of your PC, please refer to the following section.

Configuring the TCP/IP Settings of your PC

The following information will help configure the TCP/IP settings of your PC running the TwidoSoft application for programming and control of the Twido controller over the network. The procedure outlined below is workable on a PC equipped with a Windows XP operating system, and is intended as an example only. (Otherwise, for other operating systems, please refer to TCP/IP setup instructions outlined in the user's guide of the particular operating system installed on your PC.)

Step	Action
Note: If your PC is already installed and the Ethernet card is configured over the existing stand-alone network, you will not need to change the IP address settings (skip steps 1-6 and continue to the following section). Follow steps 1-6 of this procedure only if you intend to change the PC's TCP/IP settings.	
1	Select Control Panel > Network Connections from the Windows Start menu.
2	Right click on the Local Area Connection (the stand-alone network) on which you are planning to install the Twido controller, and select Properties .
3	Select TCP/IP from the list of network components installed, and click Properties . Note: If TCP/IP protocol is not among the list of installed components, please refer to the user's manual of your operating system to find out how to install the TCP/IP network component.
4	The TCP/IP Properties dialog box appears and displays the current TCP/IP settings of your PC, including IP Address and Subnet Mask . Note: On a stand-alone network, do not use the Obtain an IP address automatically option. The Specify an IP address radio-button must be selected, and the IP Address and Subnet Mask fields must contain valid IP settings.
5	Enter a valid static IP Address in dotted decimal notation. Over a stand-alone network, we suggest you to specify a Class-C network IP address (see <i>IP Addressing</i> , p. 160.). For example, 192.168.1.198 is a Class-C IP address. Note: The IP address you specify must be compatible with the network ID of the existing network. For example, if the existing network supports 192.168.1.xxx IP addresses (where 192.168.1 is the network ID, and xxx = 0-255 is the host ID), then you may specify 191.168.1.198 as a valid IP address for your PC. (Make sure the host ID 198 is unique over the network).
6	Enter a valid Subnet Mask in dotted decimal notation. If subnetting is not used on your Class-C network, we suggest you to specify a Class-C network default subnet mask such as 255.255.255.0.

Configuring the TCP/IP Settings of your Twido Controller

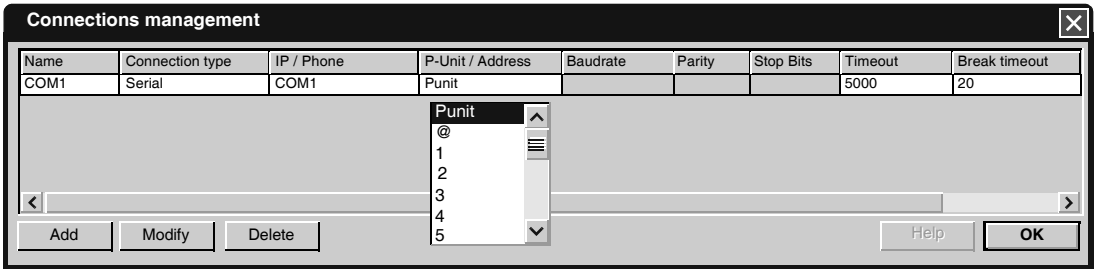
Once you have configured the TCP/IP settings of your PC hosting the TwidoSoft application, you will need to configure the TCP/P settings of the Twido controller you wish TwidoSoft to communicate with over the network, as described below:

Step	Action
1	Connect a serial cable (TSXPCX1031) from the PC running TwidoSoft to the Twido controller's RS-485 console port.
2	Launch the TwidoSoft application program on your PC.
3	Select a new Hardware from the TwisoSoft Application Brower and choose the TWDLCAE40DRF controller.
4	Select PLC > Select a connection from the TwidoSoft menu bar, and choose the COM1 port.
5	Double-click on the Ethernet Port icon in the TwisoSoft Application Browser (or select Hardware > Ethernet from the menu bar) to call up the Ethernet Configuration dialog box, as shown below:
6	From the IP Address Configure tab: • Select From a Server radio-button, to use BootP client support in order to automatically obtain a dynamic IP address from the server. (Go directly to Step 10.) Note: The TWDLCAE40DRF controller performs three, 200ms-interval retries to send BootP requests to the server. If no valid response is received, the controller uses the fallback default IP address. • Select Configured radio-button, and start configuring the static IP Address, Subnetwork mask and Gateway address fields as explained in steps 7-9. Note: At this stage, we are only dealing with the basic configuration of PC-to-controller communication over the Ethernet network. Therefore, you will not need to configure the Marked IP, Time out and Remote Devices tabs yet.

Step	Action
7	Enter a valid static IP Address for the Twido controller in dotted decimal notation. This IP address must be compatible with that of the PC's IP address that you have configured in the previous section. Note: The IP addresses of the Twido controller and the PC must share the same network ID. However, the Twido controller's host ID must be different from the PC's host ID, and unique over the network. For example, if the PC's Class-C IP address is 192.168.1.198, then a valid address for the Twido controller is 192.168.1.xxx (where 192.168.1 is the network ID, and xxx = 0-197, 199-255 is the host ID).
8	Enter a valid Subnetwork mask in dotted decimal notation. The Twido controller and the PC running TwidoSoft must be on the same network segment. Therefore, you must enter a subnet mask that is identical to that specified for the PC. Note: If subnetting is not used on your Class-C network, we suggest you to specify a Class-C network default subnet mask, such as 255.255.255.0.
9	Enter a valid Gateway address in dotted decimal notation. Note: If there is no gateway device on your stand-alone network, enter the Twido controller's own IP Address that you have just configured in step 6 in this field.
10	Click on OK to save the Ethernet configuration settings of your Twido controller.

Setting Up a New TCP/IP Connection in TwidoSoft

You will now set up a new TCP/IP connection in the TwidoSoft application. The new dedicated TCP/IP connection will allow the PC running TwidoSoft and the Twido controller to communicate over the Ethernet network.
Select **File** → **Preferences** from TwidoSoft menu bar to call up **Connexions management** dialog box:



Step	Action
1	Click the Add button in the Connections Management dialog box. Result: A new connection line is added. The new line displays suggested default connection settings. You will need to change these settings. Note: To set a new value in a field, you have two options: ● Select the desired field, then click the Modify button. ● Double click the desired field.
2	In the Name field, enter a descriptive name for the new connection. A valid name may contain up to 32 alphanumeric characters.
3	In the Connection Type field, click to unfold the dropdown list that includes: TCP/IP, Serial, Modem (if any) and USB (if any). Select TCP/IP as you are setting up a new Ethernet connection between your PC and an Ethernet-capable Twido controller.
4	In the IP / Phone field, enter a valid IP address which is the IP information of the Twido TWDLCAE40DRF controller you wish to connect to. IP Address: Enter the static IP address that you have specified for your Twido controller in a previous section.

Step	Action
5	The Punit / Address field can be filled in when IP / Phone has been selected. For a TCP/IP Type connection, default value is Direct. For a Serial Type connection, default value is Punit. When any of those is selected, next three fields (Baudrate, Parity and Stop Bits) are disabled. If you do not know the controller address, @ allows you to select it later, once the program has been downloaded. <i>(A window pops up before the first connection to let you choose the controller to which you transfer, with a 1-247 range, and 1 as the default address value.)</i>
6	Use the default settings in Timeout and Break timeout fields, unless you have specific timeout needs. (For more details, please refer to <i>Ethernet Connections Management</i>, p. 176.)
7	Click the OK button to save the new connection settings and close the Connections management dialog box. Result: The names of all newly-added connections are added to the dropdown list of connections in the File → Preferences dialog box or in the PLC → Select a connection menu.

Connecting your Controller to the Network

Overview

The following information describes how to install your TDWLCAE40DRF compact controller on your Ethernet network.

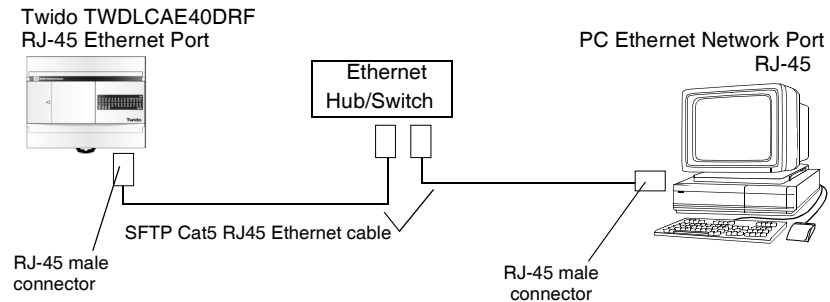
Determining the Appropriate IP Address Set

Consult your network administrator to determine if you must configure a new set of device IP, gateway and subnet mask addresses. If the administrator assigns new IP address parameters, you will need to enter this information manually in the TwidoSoft application. Follow the directions in the *TCP/IP Setup, p. 165* section hereafter.

Ethernet Network Connection

Note: Although direct cable connection (using a Ethernet crossover cable) is supported between the Twido TDWLCAE40DRF and the PC running the TwidoSoft programming software, we do not recommend it. Therefore, you should always favor a connection via a network Ethernet hub/switch.

The following figure shows a Twido network connection via an Ethernet hub/switch:



The Twido TDWLCAE40DRF features a RJ-45 connector to connect to the 100BASE-TX network Ethernet with auto negotiation. It can accommodate both 100Mbps and 10 Mbps network speeds.

Note: When connecting the Twido controller to a 100BASE-TX network, you should use at least a category 5 Ethernet cable.

IP Addressing

Overview

This section provides you with information on IP Address notation, subnet and gateway concepts as well.

IP Address

An IP address is a 32-bit quantity expressed in dotted decimal notation. It consists of four groups of numbers ranging in value from 0 to 255 and separated from one another by a dot. For example, 192.168.2.168 is an IP address in dotted decimal notation (note that this is a reserved IP address provided as an example only). On usual networks, IP addresses fall into three categories named Class A, B, and C networks. Classes can be differentiated according to the value of their first number which ranges as described in the following table:

First decimal group	IP class
0-127	Class A
128-191	Class B
192-223	Class C

IP Subnet Mask

An IP address consists of two parts, the network ID and the host ID. The subnet mask is used to split the network portion of the IP address to artificially create subnetworks with a larger number of host IDs. Thus, subnetting is used as a means of connecting multiple physical networks to logical networks. All devices on the same subnetwork share the same network ID.

All devices on the same subnetwork share the same network ID.

Note: If you are part of a large organization, then there is a good chance that subnetting is being implemented on your company's networks. Check with your network administrator to obtain adequate subnetting information when you are installing your new Twido controller on the existing network.

Gateway Address

The Gateway is the networking device also called router that provides to your network segment access to other network segments on your company's global network, access to the Internet or to a remote Intranet.

The gateway address uses the same dotted decimal notation format as the IP address described above.

Note: Check with your network administrator to obtain adequate gateway information when you are installing your new Twido controller on the existing network.

Assigning IP Addresses

Overview

This section provides you with information on how to determine which type of IP address you can assign to the Twido TWDLCAE40DRF controller that you wish to install on your network.

Installation on a Stand-alone Network

Your Twido TWDLCAE40DRF controller is intended for installation on a stand-alone Ethernet network.

Note: A network is called stand-alone when it is not linked to the Internet or a company's Intranet.

Obtaining an Address via BootP

BootP Served Address: If you choose **From a Server** in the **IP Address Configure** tab, the Twido controller will try to obtain an IP address from BootP server first.

BootP process expects a response from the BootP server. If no valid IP address is received following the BootP request transmission, Twido assumes the default IP configuration that is derived from a MAC address (see *MAC Address and Default IP Address of the Controller*, p. 161 below.)

MAC Address and Default IP Address of the Controller

MAC Address: Each Twido TWDLCAE40DRF controller has its own factory-set MAC address that is a worldwide-unique 48-bit address assigned to each Ethernet device.

Default IP Address: The default Ethernet interface IP address of the Twido controller is derived from its unique MAC address.

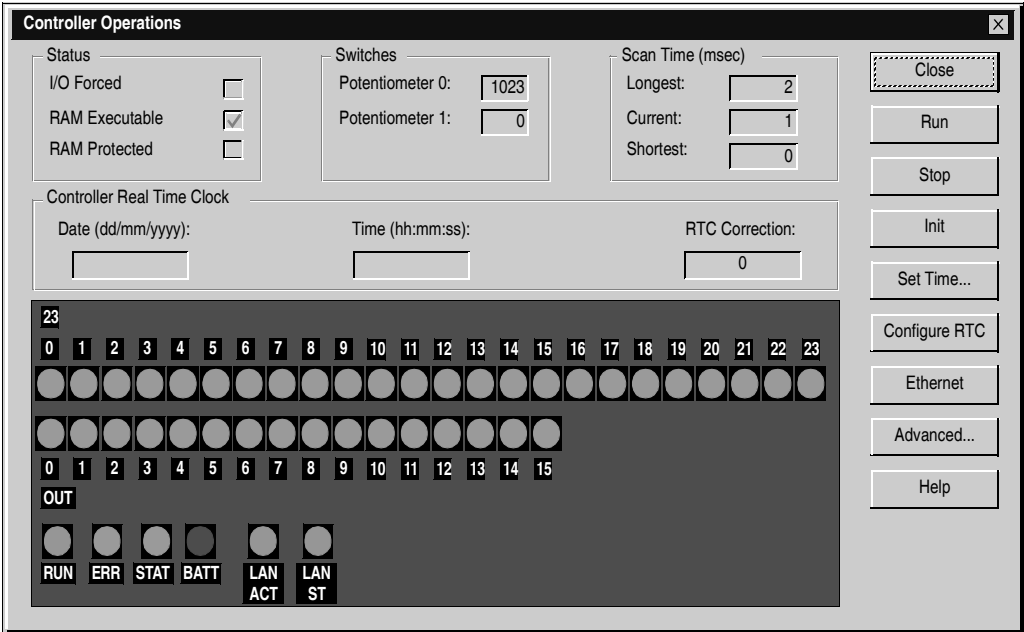
The default IP address expressed in dotted decimal notation is defined as follows: 085.016.xxx.yyy, where:

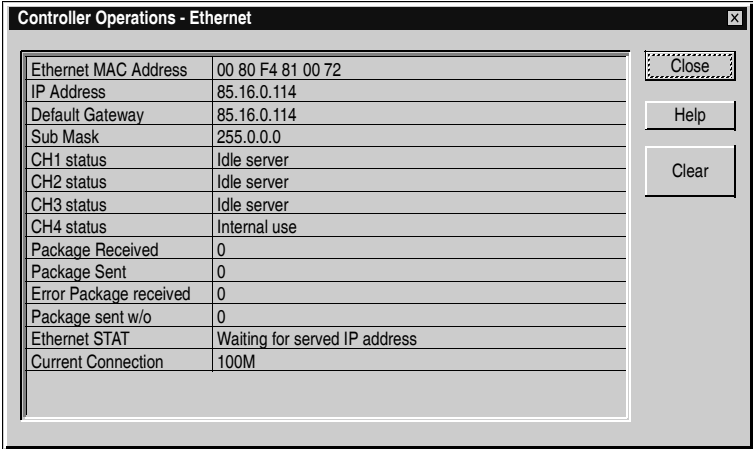
- 085.016. is a set header shared by all IP addresses derived from MAC address,
- xxx and yyy are last two numbers of the device MAC address.

For example, the IP address derived from MAC address 00.80.F4.81.01.11 is 085.016.001.17.

Checking the MAC Address and Current IP Address of the Controller

To check out the MAC address and the current IP address of your Twido controller, along with IP configuration settings (subnetwork mask and gateway addresses) and Ethernet connection status, follows these instructions:

Step	Action
1	In TwidoSoft application program, select PLC from the menu bar.
2	Select Check PLC from the menu items list. Result: The Controller Operations dialogbox appears, displaying the Twido LEDs on a soft front-panel, as shown in the figure below: 

Step	Action																												
3	Click the Ethernet button located in the right portion of the screen to access the connection parameters. Result: The Control Operations - Ethernet table appears, displaying MAC, current IP ,Subnet and Gateway information, as well as Ethernet connection information, as shown in the following figure: <table><tr><td>Ethernet MAC Address</td><td>00 80 F4 81 00 72</td></tr><tr><td>IP Address</td><td>85.16.0.114</td></tr><tr><td>Default Gateway</td><td>85.16.0.114</td></tr><tr><td>Sub Mask</td><td>255.0.0.0</td></tr><tr><td>CH1 status</td><td>Idle server</td></tr><tr><td>CH2 status</td><td>Idle server</td></tr><tr><td>CH3 status</td><td>Idle server</td></tr><tr><td>CH4 status</td><td>Internal use</td></tr><tr><td>Package Received</td><td>0</td></tr><tr><td>Package Sent</td><td>0</td></tr><tr><td>Error Package received</td><td>0</td></tr><tr><td>Package sent w/o</td><td>0</td></tr><tr><td>Ethernet STAT</td><td>Waiting for served IP address</td></tr><tr><td>Current Connection</td><td>100M</td></tr></table>	Ethernet MAC Address	00 80 F4 81 00 72	IP Address	85.16.0.114	Default Gateway	85.16.0.114	Sub Mask	255.0.0.0	CH1 status	Idle server	CH2 status	Idle server	CH3 status	Idle server	CH4 status	Internal use	Package Received	0	Package Sent	0	Error Package received	0	Package sent w/o	0	Ethernet STAT	Waiting for served IP address	Current Connection	100M
Ethernet MAC Address	00 80 F4 81 00 72																												
IP Address	85.16.0.114																												
Default Gateway	85.16.0.114																												
Sub Mask	255.0.0.0																												
CH1 status	Idle server																												
CH2 status	Idle server																												
CH3 status	Idle server																												
CH4 status	Internal use																												
Package Received	0																												
Package Sent	0																												
Error Package received	0																												
Package sent w/o	0																												
Ethernet STAT	Waiting for served IP address																												
Current Connection	100M																												
4	Note that the unique MAC address of the Twido controller is showing on the first row of the Ethernet table.																												
5	The IP information displayed in this table varies depending on the user-settings in the IP Configure tab of the Ethernet Configuration dialogbox (see <i>IP Address Configure Tab, p. 167</i>): ● if you selected From a Server in the IP Configure tab, the above table displays the default IP address (derived from MAC address) of the Twido controller, the default subnet and gateway as well. Note that the default IP address is used in fallback mode only, if no valid BootP served IP address can be obtained from the server. When one channel is used as UDP for BootP, the channel status shows Internal use. ● if you selected Configured from the IP Configure tab, the above table displays the current IP address, subnet and gateway settings that you have previously entered in the IP Configure tab. Note: The remaining fields provide information about the current status of the Ethernet connection. To find out more information, please refer to .																												

**Private IP
Addresses**

If your network is stand-alone (isolated from the Internet), you may therefore assign to your network node (Twido controller) any arbitrary IP address (as long as the IP address conforms to the IANA notation rule and it doesn't conflict with the IP address of another device already connected to the network).

Private IP addresses meet the need for arbitrary IP addressing over a stand-alone network. Note that addresses within this private address space will only be unique within the enterprise.

The following table outlines the private IP address space:

Network	Valid range for private IP addresses
Class A	10.0.0.0 -> 10.255.255.255
Class B	172.16.0.0 -> 172.31.255.255
Class C	192.168.0.0 -> 192.168.255.255

**Assigning an IP
Address to your
Controller**

Today's networks are rarely either totally isolated from the Internet or from the rest of the company's Ethernet network. Therefore, if you are installing and connecting your Twido base controller to an existing network, do not assign an arbitrary IP address without prior consulting with your network administrator. you should follow the directions outlined below when assigning an IP address to your controller.

Note: It is good practice to use Class-C IP addresses on stand-alone networks.

TCP/IP Setup

Overview

The following are detailed instructions on how to set up the Ethernet TCPI/IP configuration for your Twido TWDLCAE40DRF compact controller.

Note: TCP/IP setup can be performed when the TwidoSoft application program is in offline mode only

⚠ CAUTION

UNINTENDED EQUIPMENT OPERATION

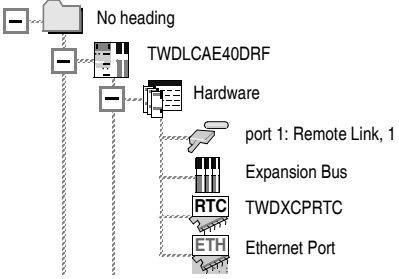
Having two devices with the same IP address can cause unpredictable operation of your network.

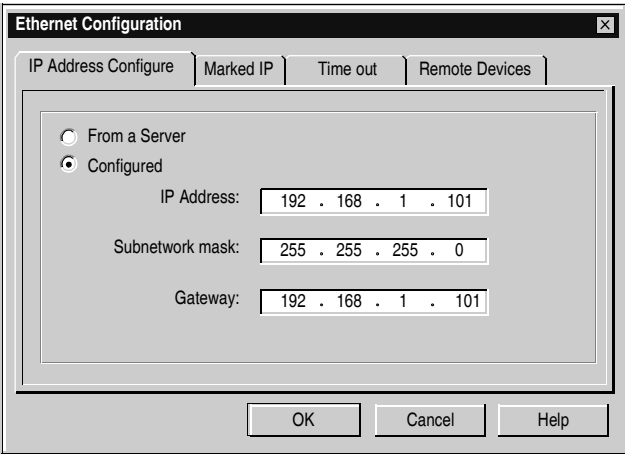
- Ensure that this device will receive a unique IP address.
- Always obtain your IP address from your system administrator to avoid the possibility of duplicate address.

Failure to follow this instruction can result in injury or equipment damage.

Calling up the Ethernet Configuration Dialogbox

The following steps detail how to call up the **Ethernet Configuration** dialogbox:

Step	Action
1	Open the Application Browser, as shown in the figure below. Result:  Note: Make sure an Ethernet-capable device such as TWDLCAE40DRF is selected as the current hardware, or otherwise the Ethernet Port hardware option will not appear.

Step	Action
2	Double-click on the Ethernet Port icon to bring up the Ethernet Configuration dialogbox, as shown below. Result:  Note: There are two alternate ways to call up the Ethernet Configuration screen: 1. Right-click on the Ethernet Port icon and select Edit from the popup list. 2. Select Hardware > Ethernet from the TwidoSoft menu bar.

TCP/IP Setup

The following sections detail how to configure the Twido TWDLCAE40DRF TCP/IP parameters by using the **IP Address Configure**, **Marked IP**, **Time out** and **Remote Devices** tabs.

IP Address Configure Tab

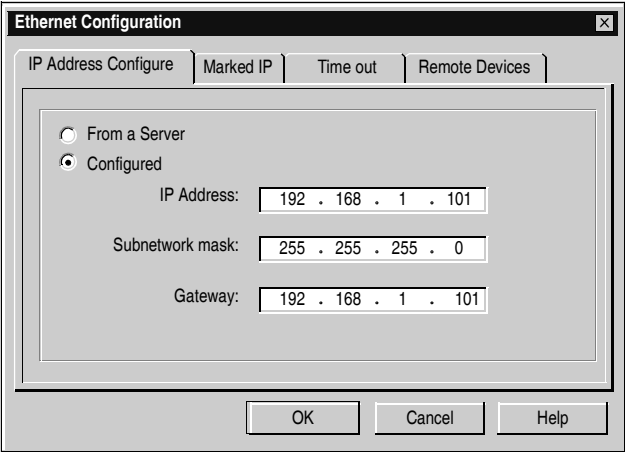
Overview

The following information describes how to configure the IP Address Configure tab of the Ethernet Configuration dialogbox.

Note: The IP address of the Twido controller can be configured when the TwidoSoft application program is in offline mode only

IP Address Configure tab

The following figure presents a sample screen of the IP Address Configure tab showing examples of IP, Subnet and Gateway addresses configured manually by the user:



Configuring the IP Address tab The following information describes how to configure the various fields in the IP Address Configure tab:

Field	Configuring
From a Server	Check this radio button if you do not wish to set the IP address of the Twido controller manually (the IP Address, Subnetwork mask and Gateway textboxes are grayed out). The Twido controller (BootP client) will then use the IP address automatically assigned by the server. The Twido controller will choose to use the default IP address (fallback state), if it cannot obtain a valid served IP address after the three retries at 200 ms intervals. (Note that the Twido controller periodically sends requests to the server at 15s intervals until it obtains a valid IP address.) The default Ethernet interface IP address is derived from its MAC address. (Note that the default IP address will not be changed automatically when any channel (excluding the channel for internal use) of PLC is active.) Note: To find out more information about BootP and MAC address, please refer to <i>Assigning IP Addresses</i>, p. 161.
Configured	Check this radio button to configure the IP, subnetwork and gateway addresses manually. Note: Consult with your network or system administrator to obtain valid IP parameters for your network.
IP Address	Enter the static IP address of your Twido in dotted decimal notation. Caution: For good device communication, the IP addresses of the PC running the TwidoSoft application and the Twido controller must share the same network ID. Note: To allow good communication over the network, each connected device must have a unique IP address. When connected to the network, the Twido controller runs a check for duplicate IP address. If a duplicate IP address is located over the network, the LAN ST LED of the Twido controller will emit 4 flashes periodically. You must then enter a new duplicate-free IP address in this field.
Subnetwork mask	Enter the valid subnet mask assigned to your controller by your network administrator. Please note that you cannot leave this field blank; you must enter a value. As default, the TwidoSoft application automatically computes and displays a default subnet mask based on the class IP that you have provided in the IP Address field above. Default subnet mask values, according to the category of the Twido network IP address, follow this rule: Class A network -> Default subnet mask: 255.0.0.0 Class B network -> Default subnet mask: 255.255.0.0 Class C network -> Default subnet mask: 255.255.255.0 Caution: For good device communication, the subnet mask configured on the PC running the TwidoSoft application and the Twido controller's subnet mask must match. Note: Unless your Twido controller has special need for subnetting, use the default subnet mask.
Gateway	Enter the IP address of the gateway. On the LAN, the gateway must be on the same segment as your Twido controller. This information typically is provided to you by your network administrator. Please note that no default value is provided by the application, and that you must enter a valid gateway address in this field. Note: If there is no gateway device on your network, simply enter your Twido controller's IP address in the Gateway field.

Marked IP Tab

Overview

The following information describes how to configure the Marked IP tab of the Ethernet Configuration dialogbox.

Note: The Marked IP can be configured when the TwidoSoft application program is in offline mode only.

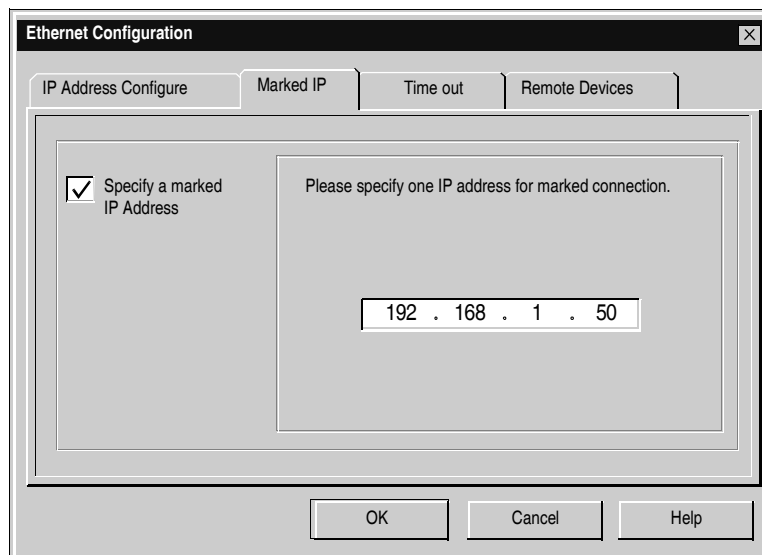
Definition of the Marked IP Function

This function allows you to reserve one of the four Ethernet TCP connection channels supported by your Twido controller for a particular client host designated as Marked IP.

Marked IP can ensure that one TCP channel is reserved and always available for communication with the specified remote device, even if the idle timeout is disabled (idle timeout is set to "0".)

Marked IP tab

The following figure presents a sample screen of the Marked IP tab showing an example of marked IP address entered by the user:



**Configuring the
Marked IP tab**

To configure the Marked IP tab, follow these steps:

Step	Action
1	Check the box labeled Specify a marked IP address to enable the Marked IP function. Note that Marked IP is disabled, as default. Result: The IP address box becomes active in the right portion of the frame, as shown in the previous figure.
2	Enter the IP address of the client host you wish to mark the IP in the provided IP address box. Note: There is no default value in this field. You must provide the IP address of the marked device, or otherwise uncheck the Specify a marked IP address box to disable this function.

Time out Tab

Overview

The following information describes how to configure the Time out tab of the Ethernet Configuration dialogbox.

Note: The Time out of the Twido controller can be configured when the TwidoSoft application program is in offline mode only.

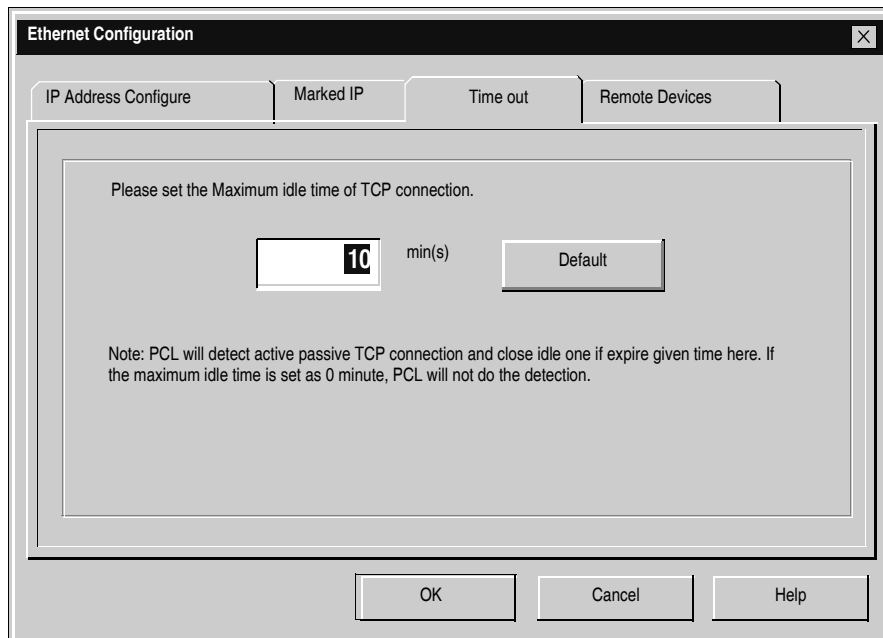
Definition of Time out

Time out applies an idle timeout to all current Ethernet TCP connections of the Twido controller. The idle timeout is the amount of time that any of the four Ethernet TCP connection channels may remain idle before the remote client host connection to this channel is dropped.

Note: The idle timer is reset whenever there is data traffic on the monitored connection channel.

Time out tab

The following figure presents a sample screen of the Time out tab showing the 10 min default value of the idle timer:



Configuring the Time out tab

To set the Idle timer, enter directly the elapsed time in minutes in the **min(s)** textbox, as shown in the previous figure.

Note:

1. The default elapsed time is 10 minutes. After you entering a value, to **reset** the configured elapsed time to 10 minutes, click on the **Default** button.
2. To **disable** the Time out function, set the elapsed time to **0**. The Twido controller no longer performs idle checks. As a result, the TCP connections stay up indefinitely.
3. The maximum idle time allowed to set is 255 minutes.

Remote Devices Tab

Overview

The following information describes how to configure the Remote Devices tab of the Ethernet Configuration dialogbox when you intend to use the EXCH3 instruction for the Twido controller to act as Modbus TCP/IP client.

Note: The Remote Devices tab of the Twido controller can be configured when the TwidoSoft application program is in offline mode only.

What You Should Know at First

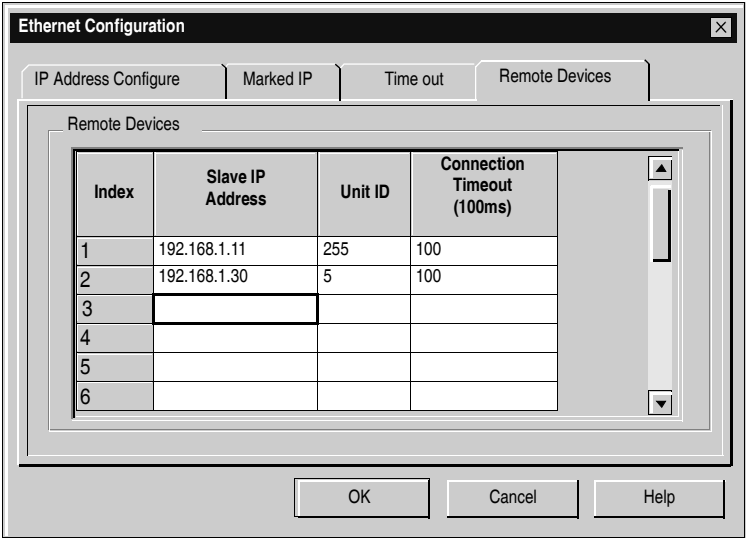
You do not need to configure the Remote Devices on any controller other than the controller that you want to use the Modbus TCP/IP client (legacy Modbus master) instruction (EXCH3).

Remote Devices Table

The Remote Devices table stores information about remote controllers (acting as Modbus TCP/IP servers) over the Ethernet network that can be queried by the Modbus TCP/IP client using the EXCH3 instruction. Therefore, you must configure the Remote Devices table properly so that the Modbus TCP/IP client controller can poll Modbus TCP/IP server controllers over the network.

Remote Devices tab

The following figure presents a sample screen of the Remote Devices tab configured on the Twido controller acting as Modbus TCP/IP client:



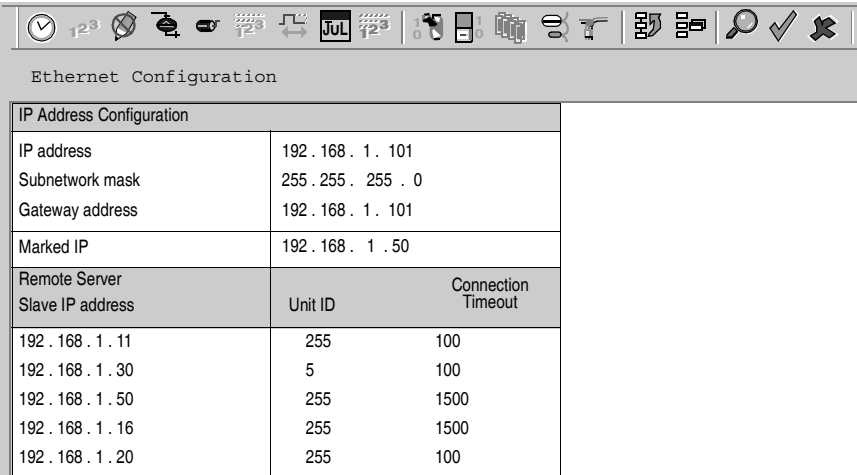
Configuring the Remote Devices tab

The following information describes how to configure the various fields in the Remote Devices tab:

Field	Configuring
Index	This is a read-only field that contains the MBAP Index associated with the Ethernet network IP address of the remote device (Modbus TPC/IP server specified in the Slave IP Address field). The MBAP Index is called by the EXCH3 instruction as one of the function's arguments to identify which remote controller specified in the table is being queried by the Modbus TCP/IP client. Note: You may specify up to 16 different remote devices indexed from 1 to 16 in this table.
Slave IP Address	Enter the IP address of the remote device (Modbus TCP/IP server) controller in this field. Note: You must configure the slave IP addresses starting at Index 1 and in growing index number, in a consecutive manner. For example, configuring slave IPs of index 1 then 3 is not allowed, for you must first configure the entry indexed 2 prior to index 3.
Unit ID	Enter the Modbus Unit ID (or Protocol Address) in this field. A valid Unit ID can range from 0 to 255. The default setting is 255. A Unit ID (other than 255) makes communications with a remote device across a Modbus bridge or gateway possible. If the target device is another Twido controller or a legacy Modbus device installed on another bus - serial link address via a gateway, then you may set the Unit ID of that remote device, accordingly. In the field, you should set the Slave IP as the gateway or bridge IP address, and the Unit ID as the Modbus serial link address of your target device.
Connection Timeout (100 ms)	Specify the elapsed time in units of 100 ms that the Twido controller will keep trying to establish a TCP connection with the remote device. If the connection is still not established after Timeout, the Twido controller will give up trying, until the next connection request by an EXCH3 instruction. A valid timeout setting can range from 0 to 65535 (which translates to 0 to 6553.5 s). The default setting is 100.

Viewing the Ethernet Configuration

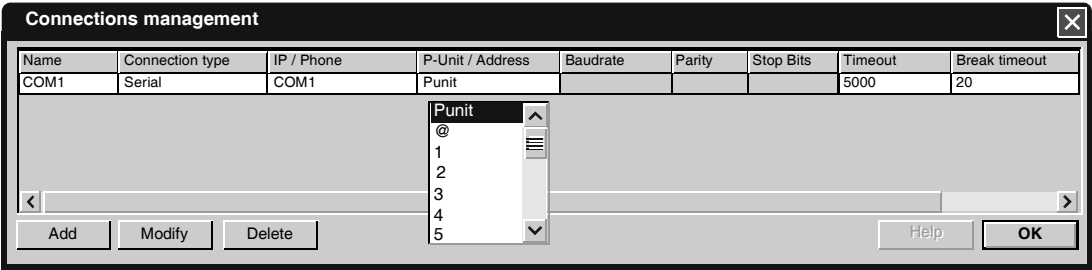
Overview	You may use the TwidoSoft Configuration Editor to view the current Ethernet configuration of the Twido controller.
Viewing the Ethernet Configuration	To view the current Ethernet configuration settings using the Configuration Editor, follow these instructions:

Step	Action
1	Select Program > Configuration Editor from the TwidoSoft menu bar.
2	Click on the shortcut labeled ETH in the Configuration Editor taskbar or double click on the Ethernet Port shortcut in the Application Browser.
3	The Ethernet TCP/IP Configuration parameters appear in a table as shown in the figure below: 
4	At this stage, if you have just made changes to your Twido's Ethernet TCP/IP configuration settings, you may still decide to keep the changes or to discard them and restore the previous configuration, as explained below: ● Select Tools > Accept Changes from the TwidoSoft menu bar, to keep the changes you have made to the TCP/IP Ethernet configuration. ● Select Tools > Cancel Changes to discard the changes and restore the previous TCP/IP Ethernet configuration settings. ● Select Tools > Edit... to return to the Ethernet Configuration dialogbox and modify the TCP/IP configuration settings. ● Select PLC > Transfer PC=>PLC... to download the complete PLC configuration file into the Twido controller.

Ethernet Connections Management

Overview The following information describes how to configure/add/delete/select a PC-to-controller Ethernet TCP/IP connection.

Setting Up a New TCP/IP Connection To set up an Ethernet TCP/IP connection between your PC running the TwidoSoft application and a TWDLCAE40DRF controller installed on your network, follow these instructions.
Select **File** → **Preferences** from TwidoSoft menu bar to call up **Connexions management** dialog box:



Step	Action
1	Click the Add button in the Connections Management dialog box. Result: A new connection line is added. The new line displays suggested default connection settings. You will need to change these settings. Note: To set a new value in a field, you have two options: ● Select the desired field, then click the Modify button. ● Double click the desired field.
2	In the Name field, enter a descriptive name for the new connection. A valid name may contain up to 32 alphanumeric characters.
3	In the Connection Type field, click to unfold the dropdown list that includes: TCP/IP, Serial, Modem (if any) and USB (if any). Select TCP/IP as you are setting up a new Ethernet connection between your PC and an Ethernet-capable Twido controller.
4	In the IP / Phone field, enter a valid IP address which is the IP information of the Twido TWDLCAE40DRF controller you wish to connect to. IP Address: Enter the static IP address that you have specified for your Twido controller in a previous section.

Step	Action
5	The Punit / Address field can be filled in when IP / Phone has been selected. For a TCP/IP Type connection, default value is Direct. For a Serial Type connection, default value is Punit. When any of those is selected, next three fields (Baudrate, Parity and Stop Bits) are disabled . If you do not know the controller address, @ allows you to select it later, once the program has been downloaded. (A window pops up before the first connection to let you choose the controller where you transfer to, with a 1-247 range, and 1 as the default address value.)
6	In the Timeout field, enter a timeout value in milliseconds (ms) for establishing a connection with the Twido controller. After timeout has elapsed and the PC has failed to connect to the controller, the TwidoSoft application will give up trying to establish a connection. To resume a new attempt for connection, select PLC → Select a connection from TwidoSoft menu bar. Note: Default Timeout value is 500 ms. Maximum Timeout value is 65535 x 100 ms (6553.5 s).
7	The Break timeout value is the maximum elapsed time allowed between a Modbus TCP/IP query and the reception of the response frame. If Break timeout is exceeded without receiving the requested response frame, the TwidoSoft application breaks the connection between the PC and the controller. Note: Default Break timeout value is 20 ms. You must set a non-zero value.
8	Click the OK button to save the new connection settings and close the Connections management dialog box. Result: The names of all newly-added connections are added to the dropdown list of connections in the File → Preferences dialog box or in the PLC → Select a connection menu.

Modifying and Deleting a TCP/IP Connection

Existing Ethernet TCP/IP connections can be deleted or have their parameters modified, as follows:

- To delete a connection from the Ethernet management dialog box, select a connection Name, then click the **Delete** button. Note that, after deletion, all connection parameters are permanently lost.
- To modify the parameters of an existing connection, select the desired field, and click the **Modify** button. Then, you can enter a new value in the selected field.

Ethernet LED Indicators

Overview

Two Ethernet communications LED indicators are located on the LED panel, at the front panel of the TWDLCAE40DRF controller and on the soft front-panel accessible via the **PLC > Check PLC** path in the TwidoSoft application as well. They are label as follows:

- LAN ACT
- LAN ST

The Ethernet LEDs provide continuous monitoring of the Ethernet port connections status and diagnostics.

LED Status

The following table describes the status of both **LAN ACT** and **LAN ST** Ethernet LED indicators.

LED	State	Color	Description
LAN ACT	Off	-	No Ethernet signal on RJ-45 port.
	Steady	Green	10BASE-TX link beat signal to indicate a 10 Mbps connection.
	Blinking		Data packets sent or received over the 10BASE-TX connection.
	Steady	Amber	100BASE-TX link beat signal to indicate a 100 Mbps connection.
	Blinking		Data packets sent or received over the 100BASE-TX connection.

LED	State	Color	Description
LAN ST	Steady	Green	Base controller is powered on. Ethernet port is ready to communicate over the network.
	Fast flashing		Ethernet initialization at power-up.
	2 Flashes, long off		No valid MAC address.
	3 Flashes, long off		Any of three possible causes: ● No link beat detected. ● Ethernet network cable is not plugged correctly or faulty cable. ● Network device (hub/switch) is faulty or not properly configured.
	4 Flashes, long off		Duplicate IP address detected over the network. (To remedy this situation, try assigning a different IP address to your Twido controller.)
	6 Flashes, long off		Using a valid converted default IP address; FDR safe-mode.
	9 Flashes, long off		Ethernet hardware failure.

TCP Modbus Messaging

Overview

You may use TCP Modbus messaging to allow the Modbus TCP Client (Master controller) to send and receive Ethernet messages to and from the Modbus TCP Server (Slave controller). As TCP Modbus is a peer-to-peer communications protocol, a Twido Ethernet-capable controller can be both Client and Server depending on whether it is querying or answering requests, respectively.

Message Exchange over the Ethernet Network

Ethernet messaging is handled by the EXCH3 instruction and the %MSG3 function block: Routing to an Ethernet host or via a gateway is supported by EXCH3, as well.

- **EXCH3 instruction:** to transmit/receive messages
- **%MSG3 Function Block:** to control the message exchanges.

EXCH3 Instruction

The EXCH3 instruction allows the Twido controller to send and/or receive information to/from Ethernet network nodes. The user defines a table of words (%MWi:L) containing control information and the data to be sent and/or received (up to 128 bytes in transmission and/or reception). The format for the word table is described in the following section.

A message exchange is performed using the EXCH3 instruction:

Syntax: [EXCH3 %MWi:L]

where: L = number of words in the control words, transmission and reception tables

The Twido controller must finish the exchange from the first EXCH3 instruction before a second can be launched. The %MSG3 function block must be used when sending several messages.

The processing of the EXCH3 list instruction occurs immediately, with any transmissions started under interrupt control (reception of data is also under interrupt control), which is considered background processing.

Note: Usage of the EXCH3 instruction is the same as EXCHx (where x = 1 or 2) used with legacy Modbus. Instruction syntaxes are also identical. However, there is one major difference in the information carried by Byte1 of the transmission and reception tables. While Byte1 of the legacy Modbus conveys the serial link address of the slave controller, Byte1 of the TCP Modbus carries the **Index** number of the Modbus TCP client controller. The Index number is specified and stored in the Remote Devices table of the TwidoSoft Ethernet Configuration (for more details see *Remote Devices Tab*, p. 173).

**EXCH3 Word
Table**

The maximum size of the transmitted and/or received frames is 128 bytes (note that this limitation applies to the TCP Modbus client only, while the TCP Modbus server supports the standard Modbus PDU length of 256 bytes). Moreover, the word table associated with the EXCH3 instruction is composed of the control, transmission and reception tables, as described below:

	Most significant byte	Least significant byte
Control table	Command	Length (Transmission/ Reception)
	Reception Offset	Transmission Offset
Transmission table	Transmitted Byte 1 (Index as specified in the Remote Device Table of the TwidoSoft Ethernet Configuration dialogbox.)	Transmitted Byte 2 as Modbus serial
	...	Transmitted Byte n
	Transmitted Byte n+1	
Reception table	Received Byte 1 (Index as specified in the Remote Device Table of the TwidoSoft Ethernet Configuration dialogbox.)	Received Byte 2 as Modbus serial
	...	Received Byte p
	Received Byte p+1	

%MSG3 Function Block

The use of the %MSG3 function is identical to that of %MSGx used with legacy Modbus. %MSG3 is used to manage data exchanges by providing:

- Communications error checking
- Coordination of multiple messages
- Transmission of priority messages

The %MSGx function block has one input and two outputs associated with it:

Input/Output	Definition	Description
R	Reset input	Set to 1: re-initializes communication or resets block (%MSGx.E = 0 and %MSGx.D = 1).
%MSGx.D	Communication complete	0: request in progress. 1: communication done if end of transmission, end character received, error, or reset of block.
%MSGx.E	Error	0: message length OK and link OK. 1: if bad command, table incorrectly configured, incorrect character received (speed, parity, and so on.), or reception table full.

EXCH3 Error Code

When an error occurs with the EXCH3 instruction:

- bits **%MSG3.D** and **%MSG3.E** are set to **1**, and
- the Ethernet communication **error code** is recorded into system word **%SW65**.

The following table presents the EXCH3 error code:

EXCH3 Error Code (recorded into System Word %SW65)	
Standard error codes common to all EXCHx (x = 1, 2, 3):	
0 - operation was successful	
1 - number of bytes to be transmitted is too great (> 128)	
2 - transmission table too small	
3 - word table too small	
4 - receive table overflowed	
5 - time-out elapsed (Note that error code 5 is void with the EXCH3 instruction and replaced by the Ethernet-specific error codes 109 and 122 described below.)	
6 - transmission	
7 - bad command within table	
8 - selected port not configured/available	
9 - reception error	
10 - can not use %KW if receiving	
11 - transmission offset larger than transmission table	
12 - reception offset larger than reception table	
13 - controller stopped EXCH processing	
Ethernet-specific error codes for EXCH3:	
101 - no such IP address	
102 - the TCP connection is broken	
103 - no socket available (all connection channels are busy)	
104 - network is down	
105 - network cannot be reached	
106 - network dropped connection on reset	
107 - connection aborted by peer device	
108 - connection reset by peer device	
109 - connection time-out elapsed	
110 - rejection on connection attempt	
111 - host is down	
120 - unknown index (remote device is not indexed in configuration table)	
121 - fatal (MAC, Chip, Duplicated IP)	
122 - receiving timed-out elapsed after data was sent	
123 - Ethernet initialization in progress	

Built-In Analog Functions

7

At a Glance

Subject of this Chapter

This chapter describes how to manage the built-in analog channel and potentiometers.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Analog potentiometer	186
Analog Channel	188

Analog potentiometer

Introduction

Twido controllers have:

- An analog potentiometer on TWDLC•A10DRF, TWDLC•A16DRF controllers and on all modular controllers (TWDLMDA20DTK, TWDLMDA20DUK, TWDLMDA20DRT, TWDLMDA40DTK and TWDLMDA40DUK,
- Two potentiometers on the TWDLC•A24DRF and TWDLCA•40DRF controllers.

Programming

The numerical values, from 0 to 1023 for analog potentiometer 1, and from 0 to 511 for analog potentiometer 2, corresponding to the analog values provided by these potentiometers are contained in the following two input words:

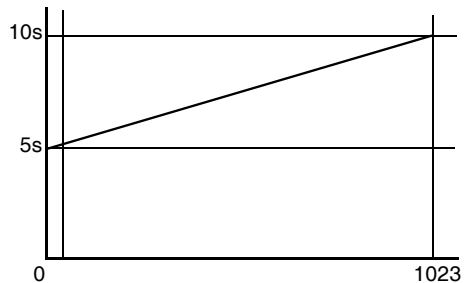
- %IW0.0.0 for analog potentiometer 1 (on left)
- %IW0.0.1 for analog potentiometer 2 (on right)

These words can be used in arithmetic operations. They can be used for any type of adjustment, for example, presetting a time-delay or a counter, adjusting the frequency of the pulse generator or machine preheating time.

Example

Adjusting the duration of a time-delay from 5 to 10 s using analog potentiometer 1:

For this adjustment practically the entire adjustment range of analog potentiometer 1 from 0 to 1023 is used.

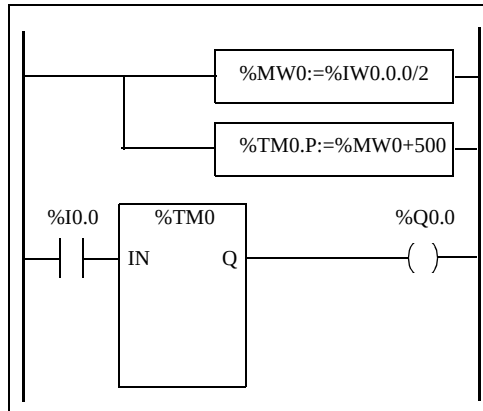


The following parameters are selected at configuration for the time-delay block %TM0:

- Type TON
- Timebase: 10 ms

The preset value of the time-delay is calculated from the adjustment value of the potentiometer using the following equation $\%TM0.P := (\%IW0.0.0/2)+500$.

Code for the above example:



```
LD      1
[%MW0:=%IW0.0.0/2]
[%TM0.P:=%MW0+500]
BLK     %TM0
LD      %I0.0
IN
OUT_BLK
LD      Q
ST      %Q0.0
END_BLK
.....
```

Analog Channel

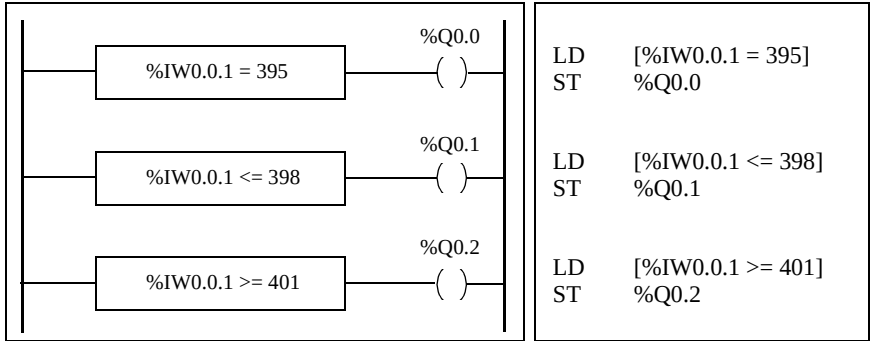
Introduction All Modular controllers (TWDLMDA20DTK, TWDLMDA20DUK, TWDLMDA20DRT, TWDLMDA40DTK, and TWDLMDA40DUK) have a built-in analog channel. The voltage input ranges from 0 to 10 V and the digitized signal from 0 to 511. The analog channel takes advantage of a simple averaging scheme that takes place over eight samples.

Principle An analog to digital converter samples an input voltage from 0 to 10 V to a digital value from 0 to 511. This value is stored in system word %IW0.0.1. The value is linear through the entire range, so that each increment is approximately 20 mV (10 V/512). Once the system detects value 511, the channel is considered saturated.

Programming Example **Controlling the temperature of an oven:** The cooking temperature is set to 350°C. A variation of +/- 2.5°C results in tripping of output %Q0.0 and %Q0.2, respectively. Practically all of the possible setting ranges of the analog channel from 0 to 511 is used in this example. Analog setting for the temperature set points are:

Temperature (°C)	Voltage	System Word %IW0.0.1
0	0	0
347.5	7.72	395
350	7.77	398
352.5	7.83	401
450	10	511

Code for the above example:



Managing Analog Modules



At a Glance

Subject of this Chapter This chapter provides an overview of managing analog modules for Twido controllers.

What's in this Chapter? This chapter contains the following topics:

Topic	Page
Analog Module Overview	190
Addressing Analog Inputs and Outputs	191
Configuring Analog Inputs and Outputs	192
Analog Module Status Information	198
Example of Using Analog Modules	199

Analog Module Overview

Introduction

In addition to the built-in 10-bit potentiometer and 9-bit analog channel, all the Twido controllers that support expansion I/O are also able to configure and communicate analog I/O modules.

These analog modules are:

Name	Points	Signal Range	Encoding
TWDAMI2HT	2 In	0 - 10 Volts or 4 - 20 mA	12 Bit
TWDAMO1HT	1 Out	0 - 10 Volts or 4 - 20 mA	12 Bit
TWDAMM3HT	2 In, 1 Out	0 - 10 Volts or 4 - 20 mA	12 Bit
TWDALM3LT	2 In, 1 Out	0 - 10 Volts, Inputs Th or PT100, Outputs 4 - 20 mA	12 Bit
TWDAVO2HT	2 Out	+/- 10 Volts	11 Bit + sign
TWDAMI4LT	4 In	0 - 10 Volts, 0 - 20 mA, NI or PT 3-wire sensors	12 Bit
TWDAMI8HT	8 In	0 - 10 Volts or 0 - 20 mA	10 Bit
TWDARI8HT	8 In	NTC or PTC sensors	10 Bit

Operating Analog Modules

Input and output words (%IW and %QW) are used to exchange data between the user application and any of the analog channels. The updating of these words is done synchronously with the controller scan during RUN mode.

CAUTION

UNEXPECTED START-UP OF DEVICES

When the controller is set to STOP, the analog output is set to its fall-back position. As is the case with digital output, the default setpoint is zero.

Failure to follow this instruction can result in injury or equipment damage.

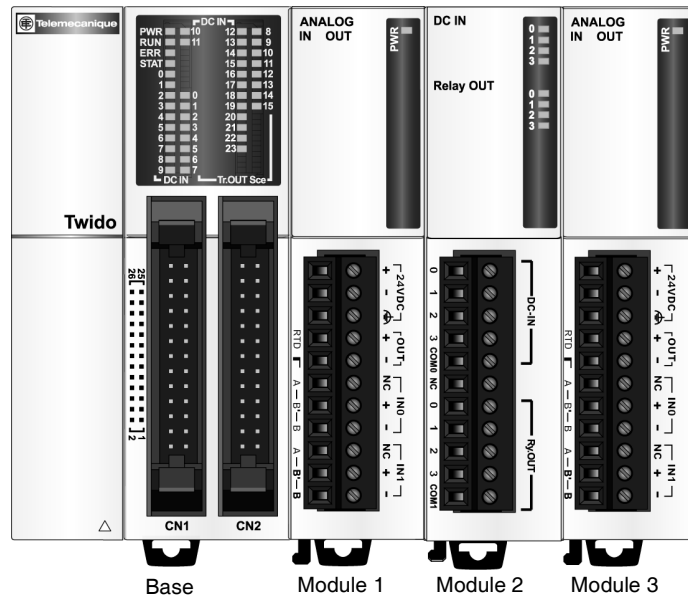
Addressing Analog Inputs and Outputs

Introduction

Addresses are assigned to the analog channels depending on their location on the expansion bus.

Example of Addressing Analog I/O

In this example, a TWDLMDA40DUK has a built-in analog-adjusted 10-bit potentiometer, a 9-bit built-in analog channel. On the expansion bus are the following: a TWDAMM3HT analog module, a TWDDMM8DRT input/output digital relay module, and a second TWDAMM3HT analog module are configured.



The table below details the addressing for each output.

Description	Base	Module 1	Module 2	Module 3
Potentiometer 1	%IW0.0.0			
Built-in analog channel	%IW0.0.1			
Analog input channel 1		%IW0.1.0		%IW0.3.0
Analog input channel 2		%IW0.1.1		%IW0.3.1
Analog output channel 1		%QW0.1.0		%QW0.3.0
Digital input channels			%I0.2.0 - %I0.2.3	
Digital output channels			%Q0.2.0 - %Q0.2.3	

Configuring Analog Inputs and Outputs

Introduction

This section provides information on configuring analog module's inputs and outputs.

Configuring Analog I/O

The **Configure Module** dialog box is used to manage the parameters of the analog modules.

You access it via the Application Browser or the **Hardware** menu.

In the Application Browser	In the Hardware menu
1. Select a module.	1. Select Configure a module .
2. Right-click Configure to directly open the Configure Module - (Module ref. and position) dialog box.	2. Choose a module from the Configure Module - Choose Module dialog box
	3. Adjust the parameters from the Configure Module - (Module ref. and position) dialog box that opens.

Note: You can only modify the parameters offline, when you are not connected to a controller.

Title bar and Contents

The Title bar displays the module reference and its position on the expansion bus. The upper part of the dialog box shows a **Description** zone.

A table shows: **Address, Symbol, Type, Range, Minimum, Maximum** and **Units**

- In TWDAMI4LT and TWIDAMI8HT, the table is preceded by an **Input type** list box.
- In TWDAVO2HT and TWDAMI8HT, the **Type** column is replaced by a **Used** column with check boxes.
- In TWDARI8HT, each channel (0-7) is configured individually within a tab, in which you can choose either the **Chart** or **Formula** configuration method. The table can be seen in a **Recap** tab.

Description

The **Description** zone displays a short summary of the module.

Address

Each row of the spreadsheet represents either an input or output channel of the module.

The addresses of each of these are identified in the following table, where "i" is the location of the module on the expansion bus.

Module Name	Address
TWDALM3LT	2 Inputs (%IW _i .0, %IW _i .1), 1 Output (%QW _i .0)
TWDAMM3HT	2 Inputs (%IW _i .0, %IW _i .1), 1 Output (%QW _i .0)
TWDAMI2HT	2 Inputs (%IW _i .0, %IW _i .1)
TWDAMO1HT	1 Output (%QW _i .0)
TWDAVO2HT	2 Outputs (%QW _i .0, %QW _i .1)
TWDAMI4LT	4 Inputs (%IW _i .0 to %IW _i .3)
TWDAMI8HT	8 Inputs (%IW _i .0 to %IW _i .7)
TWDARI8HT	8 Inputs (%IW _i .0 to %IW _i .7)

Symbol

This is a read-only display of a symbol, if assigned, for the address.

Input type and/or Type

This identifies the mode of a channel. The choices depend on the channel and type of module.

For the TWDAMO1HT, TWDAMM3HT and TWDALM3LT, you can configure the single output channel type as:

Type
Not used
0 - 10 V
4 – 20 mA

For the TWDAMI2HT and TWDAMM3HT, you can configure the two input channel types as:

Type
Not used
0 - 10 V
4 – 20 mA

For the TWDALM3LT, you can configure the two input channel types as:

Type
Not used
Thermocouple K
Thermocouple J
Thermocouple T
PT 100

For the TWDAVO2HT, there is no type to adjust.

For the TWDAMI4LT, you can configure the four input types as:

Input type	Type
Voltage	Not used 0-10 V
Current	Not used 0-20 mA
Temperature	Not used PT 100 PT 1000 NI 100 NI 1000

For the TWDAMI8HT, you can configure the eight input types as:

Input type
0 - 10 V
0 - 20 mA

For the TWDARI8HT, you can configure each input channel (0-7) individually, from the **Operation** field in the lower part of the window. Directly choose a **Mode**, and a **Range**, if needed. You can then view a summary of all information in the Recap tab, with a **Type** column showing:

Type
Not used
NTC / CTN
PTC / CTP

CAUTION

EQUIPMENT DAMAGE

If you have wired your input for a voltage measurement, and you configure TwidoSoft for a current type of configuration, you may permanently damage the analog module. Ensure that the wiring is in agreement with the TwidoSoft configuration.

Failure to follow this instruction can result in injury or equipment damage.

Range

This identifies the range of values for a channel. The choices depend on the specific type of channel and module.

Once the **Type** is configured, you can set the corresponding **Range**. A table shows the **Minimum** and **Maximum** values accepted - either fixed or user-defined - together with the **Unit**, if needed.

Range (NTC sensors)	Minimum	Maximum	Units	I/O Analog Modules
Normal	0	4095	None	TWDALM3LT TWDAMO1HT TWDAMM3HT TWDAMI2HT TWDAMI4LT
	-2048	2047		TWDAVO2HT
	0	1023		TWDAMI8HT TWDARI8HT

Range (NTC sensors)	Minimum	Maximum	Units	I/O Analog Modules
Custom	User defined with a min. of -32768	User defined with a max. of 32767	None	All I/O Analog Modules
Celsius	-1000	5000	0.1°C	TWDALM3LT
	Dynamically updated by TwidoSoft according to user-defined parameters			TWDARI8HT
	-2000	6000		TWDAMI4LT (Pt sensor)
	-500	1500		TWDAMI4LT (Ni sensor)
Fahrenheit	-1480	9320	0.1°F	TWDALM3LT
	Dynamically updated by TwidoSoft according to user-defined parameters			TWDARI8HT
	-3280	11120		TWDAMI4LT (Pt sensor)
	-580	3020		TWDAMI4LT (Ni sensor)
Resistance	100	10000	Ohm	TWDARI8HT
	74	199		TWDAMI4LT (Ni100)
	742	1987		TWDAMI4LT (Ni1000)
	18	314		TWDAMI4LT (Pt100)
	184	3138		TWDAMI4LT (Pt1000)

Chart or Formula Method

In TWDARI8HT, each channel (0-7) is configured individually within a tab. Check the **Used** box then choose between **Chart** and **Formula** configuration methods.

- **Chart (graphical) method**

(**R1**, **T1**) and (**R2**, **T2**) correspond to float format coordinates of two points belonging to the curve.

R1(default 8700) and **R2** (default 200) values are expressed in Ohms.

T1 (default 233.15) and **T2** (default 398.15) values can have their unit set in the **Unit** list box: **Kelvin** (default), **Celsius** or **Fahrenheit**.

Note: Changing the temperature unit after setting the **T1** and **T2** values will not automatically recalculate **T1** and **T2** values with the new unit.

- **Formula method**

Provided you know **Rref**, **Tref** and **B** parameters, you can use this method to define sensor characteristics.

Rref (default 330) is expressed in Ohms.

B is default 3569 (min. 1, max. 32767).

Tref (default 298.15) can have its unit set in the **Unit** list box: **Kelvin** (default), **Celsius** or **Fahrenheit**.

Here is a table of corresponding min./max. **Tref** values between units:

Unit	Min. value	Max. value
Kelvin	1	650
Celsius	-272	376
Fahrenheit	-457	710

In both Chart and Formula windows, you can import values from another channel in the currently configured channel:

1. Select a channel number out of the **Channel No** box.
2. Press the **Import values** button.

Some error or warning messages can be associated with these windows.

Note: If you start setting values then decide to switch from Chart to Formula or from Formula to Chart, a warning message pops up, explaining that it will revert to default values and that any modified values will be lost.

Analog Module Status Information

Status Table

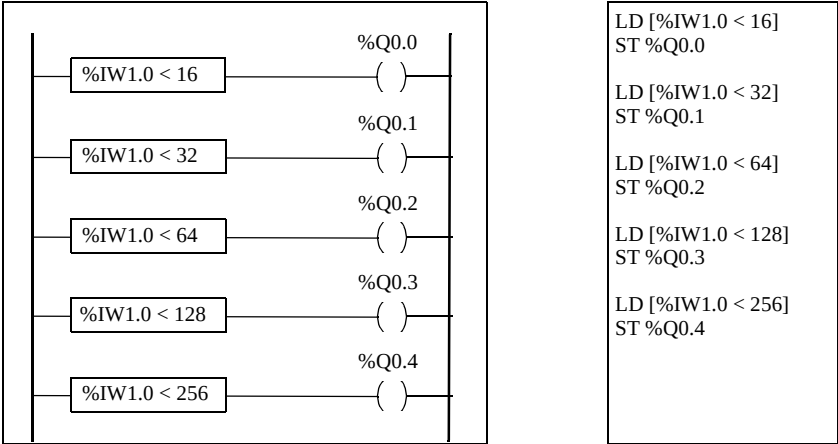
The following table has the information you need to monitor the status of Analog I/O modules.

System Word	Function	Description
%SW80	Base I/O Status	Bit [0] Channels in normal operation (for all its channels) Bit [1] Module under initialization (or of initializing information of all channels) Bit [2] Hardware failure (external power supply failure, common to all channels) Bit [3] Module configuration fault Bit [4] Converting data input channel 0 in progress Bit [5] Converting data input channel 1 in progress Bit [6] Input thermocouple channel 0 not configured Bit [7] Input thermocouple channel 1 not configured Bit [8] Not used Bit [9] Unused Bit [10] Analog input data channel 0 over range Bit [11] Analog input data channel 1 over range Bit [12] Incorrect wiring (analog input data channel 0 below current range, current loop open) Bit [13] Incorrect wiring (analog input data channel 1 below current range, current loop open) Bit [14] Unused Bit [15] Output channel not available
%SW81	Expansion I/O Module 1 Status: Same definitions as %SW80	
%SW82	Expansion I/O Module 2 Status: Same definitions as %SW80	
%SW83	Expansion I/O Module 3 Status: Same definitions as %SW80	
%SW84	Expansion I/O Module 4 Status: Same definitions as %SW80	
%SW85	Expansion I/O Module 5 Status: Same definitions as %SW80	
%SW86	Expansion I/O Module 6 Status: Same definitions as %SW80	
%SW87	Expansion I/O Module 7 Status: Same definitions as %SW80	

Example of Using Analog Modules

Introduction This section provides an example of using Analog modules available with Twido.

Example: analog input This example compares the analog input signal with five separate threshold values. A comparison of the analog input is made and a bit is set on the base controller if it is less than or equal to the threshold.



Example: analog output

The following program uses an analog card in slot 1 and 2. The card used in slot 1 has a 10-volt output with a "normal" range:



```
LD 1
[%QW0.1.0:=4095
LD 1
[%QW0.2.0:=%MW0
```

- Example of output values for %QW1.0=4095 (normal case):
The following table shows the output voltage value according to the maximum value assigned to %QW1.0:

	numerical value	analog value (volt)
Minimum	0	0
Maximum	4095	10
Value 1	100	0.244
Value 2	2460	6

- Example of output values for a customized range (minimum = 0, maximum = 1000):
The following table shows the output voltage value according to the maximum value assigned to %QW1.0:

	numerical value	analog value (volt)
Minimum	0	0
Maximum	1000	10
Value 1	100	1
Value 2	600	6

Installing the AS-Interface V2 bus

9

At a Glance

Subject of this Chapter

This chapter provides information on the software installation of the AS-Interface Master module TWDNOI10M3 and its slaves.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Presentation of the AS-Interface V2 bus	202
General functional description	203
Software set up principles	206
Description of the configuration screen for the AS-Interface bus	207
Configuration of the AS-Interface bus	209
Description of the debug screen	215
Modification of Slave Address	218
Updating the AS-Interface bus configuration in online mode	220
Automatic addressing of an AS-Interface V2 slave	225
How to insert a slave device into an existing AS-Interface V2 configuration	226
Automatic replacement of a faulty AS-Interface V2 slave	227
Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus	228
Programming and diagnostics for the AS-Interface V2 bus	229
AS-Interface V2 bus interface module operating mode:	233

Presentation of the AS-Interface V2 bus

Introduction

The AS-Interface Bus (Actuator Sensor-Interface) allows the interconnection on a single cable of sensor devices/actuators at the lowest level of automation. These sensors/actuators will be defined in the documentation as **slave devices**.

To implement the AS-Interface application you need to define the physical context of the application into which it will be integrated (expansion bus, supply, processor, modules, AS-Interface slave devices connected to the bus) then ensure its software implementation.

This second aspect will be carried out from the different TwidoSoft editors:

- either in local mode,
- or in online mode.

AS-Interface V2 Bus

The AS-interface Master module **TWDNOI10M3** includes the following functionalities:

- M3 profile: This profile includes all the functionalities defined by the AS-Interface V2 standard, but does not support the S7-4 analog profiles
- One AS-Interface channel per module
- Automatic addressing for the slave with the address 0
- Management of profiles and parameters
- Protection from polarity reversion on the bus inputs

The AS-Interface bus then allows:

- Up to 31 standard address and 62 extended address slaves
- Up to 248 inputs and 186 outputs
- Up to 7 analog slaves (Max of four I/O per slave)
- A cycle time of 10 ms maximum

A maximum of 2 AS-Interface Master modules can be connected to a Twido modular controller, a TWDLC•A24DRF or a TWDLCA•40DRF compact controller.

General functional description

General Introduction

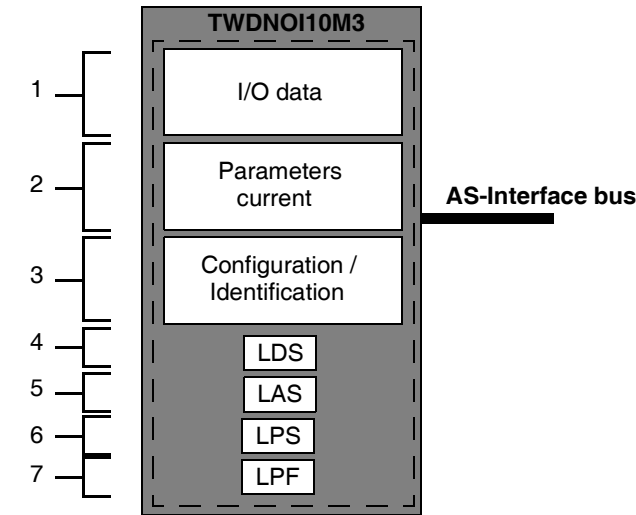
For the AS-Interface configuration, TwidoSoft software allows the user to:

- Manually configure the bus (declaration of slaves and assignment of addresses on the bus)
- Adapt the configuration according to what is present on the bus
- Acknowledge the slave parameters
- Control bus status

For this reason, all data coming from or going to the AS-Interface Master are stored in specific objects (words and bits).

**AS-Interface
Master Structure**

The AS-Interface module includes data fields that allow you to manage the lists of slaves and the images of input / output data. This information is stored in volatile memory.
The figure below shows **TWDNOI10M3** module architecture.



Key:

Address	Item	Description
1	I/O data (IDI, ODI)	Images of 248 inputs and 186 outputs of AS-Interface V2 bus.
2	Current parameters (PI, PP)	Image of the parameters of all the slaves.
3	Configuration/ Identification (CDI, PCD)	This field contains all the I/O codes and the identification codes for all the slaves detected.
4	LDS	List of all slaves detected on the bus.
5	LAS	List of slaves activated on the bus.
6	LPS	List of slaves provided on the bus and configured via TwidoSoft.
7	LPF	List of slaves having a device fault.

Structure of Slave Devices

The standard address slaves each have:

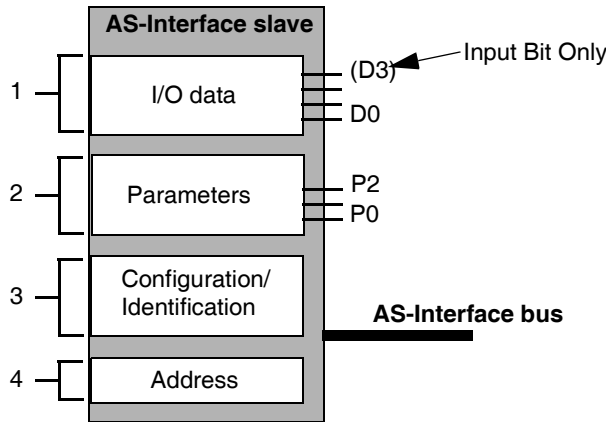
- 4 input/output bits
- 4 parametering bits

The slaves with extended addresses each have:

- 4 input/output bits (the last bit is reserved for entry only)
- 3 parametering bits

Each slave has its own address, profile and sub-profile (defines variables exchange).

The figure below shows the structure of an extended address slave:



Key:

Address	Item	Description
1	Input/output data	Input data is stored by the slave and made available for the AS-Interface master. Output data is updated by the master module.
2	Parameters	The parameters are used to control and switch internal operating modes to the sensor or the actuator.
3	Configuration/ Identification	This field contains: • the code which corresponds to I/O configuration, • the slave identification (ID) code, • the slave identification codes (ID1 and ID2).
4	Address	Physical address of slave.
Note: The operating parameters, address, configuration and identification data are saved in a non-volatile memory.		

Software set up principles

At a Glance

To respect the philosophy adopted in TwidoSoft, the user should adopt a step-by-step approach when creating an AS-Interface application.

Set up principle

The user must know how to functionally configure his AS-Interface bus (See *How to insert a slave device into an existing AS-Interface V2 configuration*, p. 226).

The following table shows the different software implementation phases of the AS-Interface bus.

Mode	Phase	Description
Local	Declaration of module	Choice of the slot for the AS-Interface Master module TWDNOI10M3 on the expansion bus.
	Configuration of the module channel	Choice of "master" modes.
	Declaration of slave devices	Selection for each device: • of its slot number on the bus, • of the type of standard or extended address slave.
	Confirmation of configuration parameters	Confirmation at slave level.
	Global confirmation of the application	Confirmation of application level.
Local or connected	Symbolization (optional)	Symbolization of the variables associated with the slave devices.
	Programming	Programming the AS-Interface V2 function.
Connected	Transfer	Transfer of the application to the PLC.
	Debugging	Debugging the application with the help of: • the debug screen, used on the one hand to display slaves (address, parameters), and on the other, to assign them the desired addresses, • diagnostic screens allowing identification of errors.

Note: The declaration and deletion of the AS-Interface Master module on the expansion bus is the same as for another expansion module. However, once two AS-Interface Master modules have been declared on the expansion bus, TwidoSoft will not permit another one to be declared.

Precautions Prior to Connection

Before connecting (via the software) the PC to the controller and to avoid any detection problem:

- Ensure that no slave is physically present on the bus with address 0
- Ensure that 2 slaves are not physically present with the same address.

Description of the configuration screen for the AS-Interface bus

At a Glance

The configuration screen of the AS-Interface master module gives access to the parameters associated with the module and the slave devices. It can be used to display and modify parameters in offline mode.

Illustration of Offline Mode

Illustration of the configuration screen in offline mode:

Configure Module - TWDNOI10M3 [Position 1]

Description

Master AS-Interface expansion module

Configuration

AS-interface configuration

Std/A Slaves		/B Slaves
	00	
XVBC21A	01	
	02	
	03	ASI20MT4IE
	04	
	05	INOUT24/12
	06	
WXA36	07	
	08	
	09	
	10	
	11	
	12	
	13	
	14	
	15	
	16	

Slave 1A

Characteristics

Profile: IO 7 ID f ID1 f ID2 f

Comment: XVB illuminated column base

Parameters

☒ Bits ☐ Decimal

0	☒	Blink e1	2	☒	Blink e3
1	☒	Blink e2	3	☒	Blink e4

Inputs/Outputs

Inputs	Number	Outputs	Number
1	%IA1.1A.0	1	%QA1.1A.0
2	%IA1.1A.1	2	%QA1.1A.1

Master mode

☒ Set data exchange active

☐ Network down

☒ Automatic addressing

OK Cancel Help

Description of the Screen in Offline Mode

This screen groups all data making up the bus in three blocks of information:

Blocks	Description
AS-interface configuration	Bus image desired by the user: view of standard and extended address setting slaves expected on the bus. Move the cursor down the vertical bar to access the following addresses. Grayed out addresses correspond to addresses not available here for slave configuration. If, for example, a new standard address setting slave is declared with the address 1A, the address 1B is automatically grayed out.
Slave xxA/B	Configuration of the selected slave: ● Characteristics: IO code, ID code, ID1 and ID2 codes (profiles), and comments on the slave, ● Parameters: list of parameters (modifiable), in binary (4 check boxes) or decimal (1 check box) form, at the discretion of the user, ● Inputs/Outputs: list of available I/Os and their respective addresses.
Master mode	Activation or deactivation is possible for the two functionalities available for this AS-Interface module (for example, automatic addressing). "Network down" allows you to force the AS-Interface bus to enter the offline mode. "Automatic addressing" mode is checked by default. Note: The "Data exchange activation" function is not yet available.

The screen also includes 3 buttons:

Buttons	Description
OK	Used to save the AS-Interface Bus configuration visible on the configuration screen Then return to the main screen. The configuration can then be transferred to the Twido controller.
Cancel	Returns to the main screen without acknowledging the changes in progress.
Help	Opens a Help window on-screen.

Note: Changes in the configuration screen can only be made in offline mode.

Configuration of the AS-Interface bus

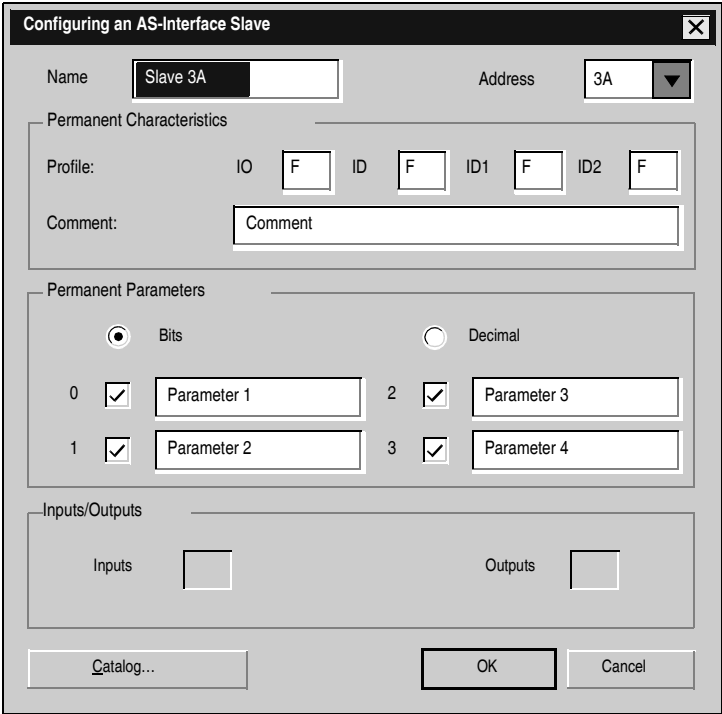
Introduction

AS-Interface bus configuration takes place in the configuration screen in local mode. Once the AS-Interface Master and the master modes have been selected, configuration of the AS-Interface bus consists of configuring the slave devices.

Procedure for Declaring and Configuring a Slave

Procedure for creating or modifying a slave on the AS-Interface V2 bus:

Step	Action
1	On the desired address cell (not grayed out) in the bus image: ● Double click: access to step 3 OR ● Right click: Result:
	Note: A shortcut menu appears. This is used to: ● Configure a new slave on the bus ● Modify the configuration of the desired slave ● Copy (or Ctrl+C), cut (or Ctrl+X), paste a slave (or Ctrl+V) ● Delete a slave (or Del)

Step	Action
2	In the shortcut menu, select: • "New" to create a new slave: A slave configuration screen is displayed; the "Address" field shows the selected address, the "Profile" fields are set to F by default and all other fields in the screen are blank. • "Open" to create a new slave or to modify the configuration of the selected slave. For a new slave, a new screen for configuring the slave is displayed, the "Address" field shows the selected address, the "Profile" fields are set to F by default and all other fields in the screen are blank. For a modification, the slave configuration screen is displayed with fields containing the values previously defined for the selected slave. Illustration of a Configuration Screen for a New Slave:  The screenshot shows a window titled "Configuring an AS-Interface Slave". It has a close button in the top right. The "Name" field contains "Slave 3A". The "Address" dropdown is set to "3A". Under "Permanent Characteristics", the "Profile" fields for IO, ID, ID1, and ID2 are all set to "F". There is a "Comment" text field. Under "Permanent Parameters", the "Bits" radio button is selected. There are four checked checkboxes for parameters 0, 1, 2, and 3, each with a text field labeled "Parameter 1" through "Parameter 4". Under "Inputs/Outputs", there are "Inputs" and "Outputs" text fields. At the bottom are "Catalog...", "OK", and "Cancel" buttons.
3	In the slave configuration screen that is then displayed, enter or modify: • the name of the new profile (limited to 13 characters), • a comment (optional). Or click "Catalog..." and select a slave from the pre-configured AS-Interface profile family.

Step	Action
4	Enter: • the IO code (corresponds to the input/output configuration), • the ID code (identifier), (plus ID1 and for an extended type). Note: The "Inputs" and "Outputs" fields show the number of input and output channels. They are automatically implemented when the IO code is entered.
5	For each parameter define: • the system's acknowledgement (box checked in "Bits" view, or decimal value between 0 and 15 in "Decimal" view), • a name that is more meaningful than "Parameter X" (optional). Note: The selected parameters are the image of permanent parameters to be provided to the AS-Interface Master.
6	If needed, modify "Address" (within the limit of available addresses on the bus), by clicking the up/down arrows to the left of the address (access is then given to authorized addresses) or by entering the address using the keyboard.
7	Confirm the slave configuration by clicking on the "OK" button. The result is the check that: • the IO and ID are authorized, • the slave address is authorized (if keyboard entry is used) according to the ID code ("bank" /B slaves are only available if the ID code is equal to A). If an error occurs, an error message warns the user (for example: "The slave cannot have this address") and the screen is displayed again with the initial values (in the profile or address, depending on the error).

Note: The software limits the number of analog slave declarations to 7.

Note: About the Schneider AS-Interface catalog: when you click Catalog, you can create and configure slaves in "Private family" (other than those in the Schneider AS-Interface catalog).

**AS-Interface
Catalog**

The Catalog button can be used to facilitate configuration of slaves on the bus. When you use a slave from the Schneider family, use this button to simplify and speed up configuration. Clicking on "Catalog" in the window "Configure an AS-Interface slave" opens the following window:

AS-Interface Catalog

Families of AS-Interface profiles:

6: Illuminated columns

AS-Interface Catalog: Illuminated columns

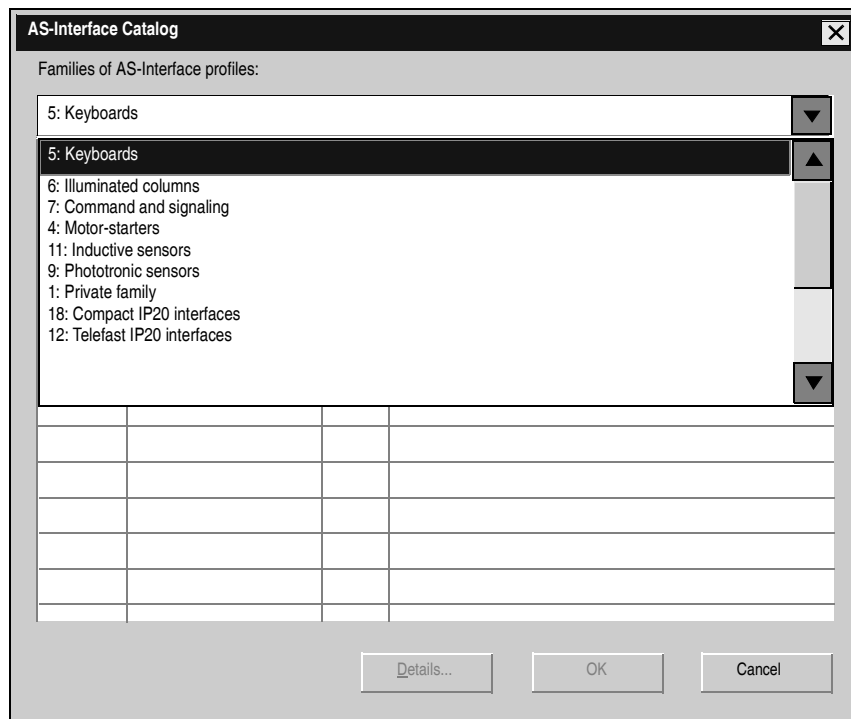
Profile	AS-Interface Name	@	Comment
7.F.F.F	XVBC21A	std	XVB illuminated column base.
8.F.F.F	XVA-S102	std	XVA illuminated column base.

Details...

OK

Cancel

The drop-down menu gives you access to all the families of the Schneider AS-Interface catalog:



When you have chosen your family, the list of corresponding slaves appears. Click on the required slave and validate by clicking "OK"

Note: You can display the characteristics of a slave by clicking "Details".

Note: You can add and configure slaves that are not part of the Schneider catalog. Simply select the private family and configure the new slave.

Description of the debug screen

At a Glance

When the PC is **connected** to the controller (after uploading the application to the controller), the "Debug" tab appears to the right of that of "Configuration"; it allows the debug screen to be accessed.

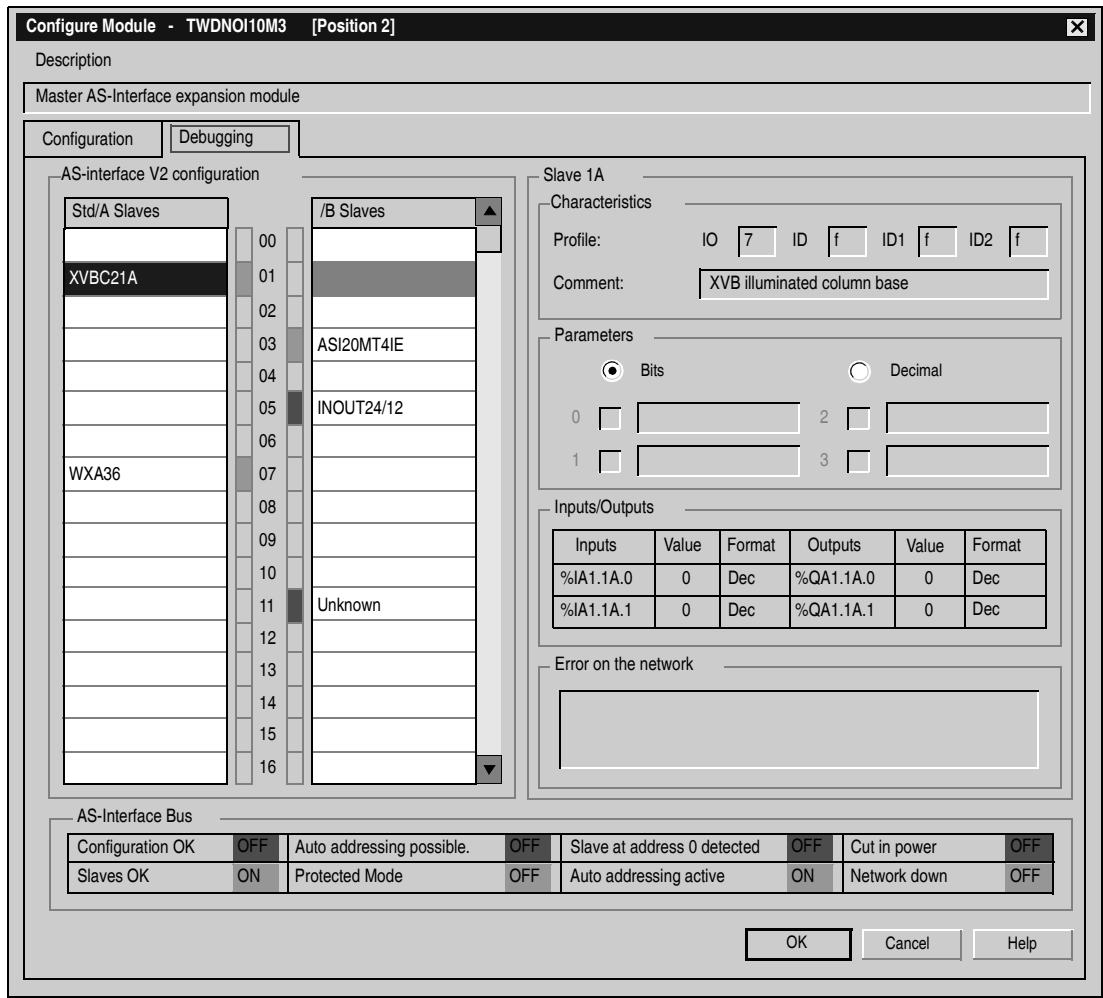
The debug screen dynamically provides an image of the physical bus that includes the:

- List of expected slaves (entered) during configuration with their name, and the list of detected slaves (with unknown names, but otherwise expected),
- Status of the AS-Interface module and the slave devices,
- Image of the profile, parameters and input/output values of the selected slaves.

It also enables the user:

- To obtain diagnostics of the slaves on which an error has occurred (See *Displaying Slave Status*, p. 217),
 - To modify the address of a slave in online mode (See *Modification of Slave Address*, p. 218),
 - To transmit the image of the slaves to the configuration screen (See *Updating the AS-Interface bus configuration in online mode*, p. 220),
 - To address all the slaves with the desired addresses (during the first debugging).
-

Illustration of the "Debug" Screen The illustration of the debug screen (in online mode only) looks like this:



Description of the Debug Screen

The "Debug" screen provides the same information as the configuration screen (See *Description of the Screen in Offline Mode, p. 208*).

The differences are listed in the following table:

Schedule	Description
AS-interface V2 configuration	Image of the physical bus. Includes slave status: ● Green indicator lamp: the slave with this address is active. ● Red indicator lamp: an error has occurred on the slave at this address, and the message informs you of the error type in the "Error on the network" window.
Slave xxA/B	Image of the configuration of the selected slave: ● Characteristics: image of the profile detected (grayed out, non-modifiable), ● Parameters: image of the parameters detected. The user can select only the parameter display format, ● Inputs/Outputs: the input/output values detected are displayed, non-modifiable.
Error on the network	Informs you of the error type, if an error has occurred on the selected slave.
AS-Interface Bus	Information resulting from an implicit "Read Status" command. ● Shows bus status: for example, "Configuration OK = OFF" indicates that the configuration specified by the user does not correspond to the physical configuration of the bus, ● Shows the authorized functionalities for the AS-Interface Master module: for example, "Automatic addressing active = ON" indicates that the automatic addressing Master mode is authorized.

Displaying Slave Status

When the indicator lamp associated with an address is red, there is an error on the slave associated with this address. The "Error on the network" window then provides the diagnostics of the selected slave.

Description of errors:

- The profile specified by the user by the configuration of a given address does not correspond to the actual profile detected for this address on the bus (diagnostics: "Profile error"),
- A new slave, not specified at configuration, is detected on the bus: a red indicator lamp is then displayed for this address and the slave name displayed is "Unknown" (diagnostics: "Slave not projected"),
- Peripheral fault, if the slave detected supports it (diagnostics: "Peripheral fault"),
- A configured profile is specified but no slave is detected for this address on the bus (diagnostics: "Slave not detected").

Modification of Slave Address

At a Glance

From the debug screen, the user can modify the address of a slave in online mode.

Modification of Slave Address

The following table shows the procedure for modifying a slave address:

Step	Description
1	Access the "Debug" screen.
2	Select a slave in the "AS-interface V2 Configuration" zone.
3	Drag and drop the slave to the cell corresponding to the desired address. Illustration: Dragging and dropping slave 3B to address 15B

ConfigurationDebugging

AS-interface V2 configuration

Std/A Slaves

XVBC21A

WXA36

0001020304050607080910111213141516

/B Slaves

ASI20MT41E

INOUT24/12

ASI20MT41E

Unknown

Step	Description																																																						
Result:	All the slave parameters are automatically checked to see if the operation is possible.																																																						
Illustration of result:	ConfigurationDebugging AS-interface V2 configuration <table><tr><th>Std/A Slaves</th><th></th><th>/B Slaves</th></tr><tr><td></td><td>00</td><td></td></tr><tr><td>XVBC21A</td><td>01</td><td></td></tr><tr><td></td><td>02</td><td></td></tr><tr><td></td><td>03</td><td>ASI20MT41E</td></tr><tr><td></td><td>04</td><td></td></tr><tr><td></td><td>05</td><td>INOUT24/12</td></tr><tr><td></td><td>06</td><td></td></tr><tr><td>WXA36</td><td>07</td><td></td></tr><tr><td></td><td>08</td><td></td></tr><tr><td></td><td>09</td><td></td></tr><tr><td></td><td>10</td><td></td></tr><tr><td></td><td>11</td><td>Unknown</td></tr><tr><td></td><td>12</td><td></td></tr><tr><td></td><td>13</td><td></td></tr><tr><td></td><td>14</td><td></td></tr><tr><td></td><td>15</td><td>Unknown</td></tr><tr><td></td><td>16</td><td></td></tr></table> After performing this operation, the diagnostics for the slave at address 3B indicate "slave not detected" meaning that the slave expected at this address is no longer there. By selecting the address 15B, the profile and the parameters of the moved slave can be re-located, but the name of the slave remains unknown as it was not expected at this address.	Std/A Slaves		/B Slaves		00		XVBC21A	01			02			03	ASI20MT41E		04			05	INOUT24/12		06		WXA36	07			08			09			10			11	Unknown		12			13			14			15	Unknown		16	
Std/A Slaves		/B Slaves																																																					
	00																																																						
XVBC21A	01																																																						
	02																																																						
	03	ASI20MT41E																																																					
	04																																																						
	05	INOUT24/12																																																					
	06																																																						
WXA36	07																																																						
	08																																																						
	09																																																						
	10																																																						
	11	Unknown																																																					
	12																																																						
	13																																																						
	14																																																						
	15	Unknown																																																					
	16																																																						

Note: The profile and parameters of a slave are not associated with a name. Several slaves with different names can have the same profiles and parameters.

Updating the AS-Interface bus configuration in online mode

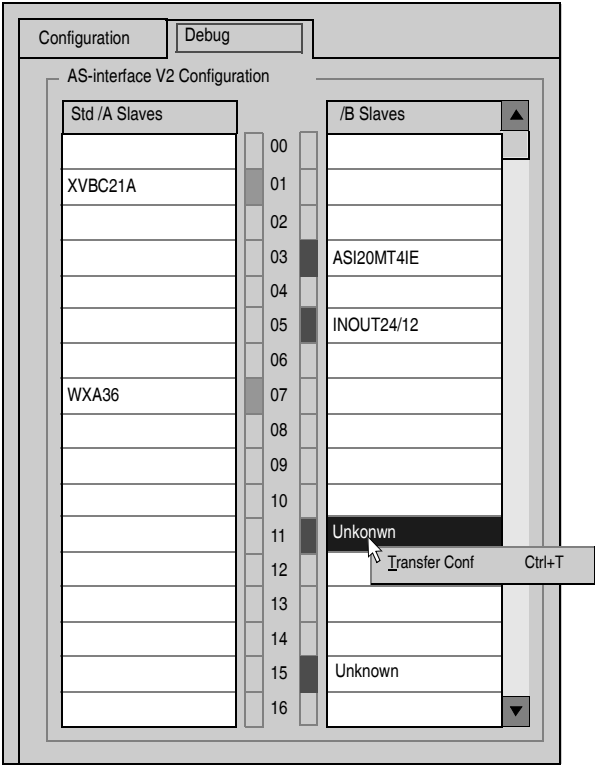
At a Glance

In online mode, no modification of the configuration screen is authorized and the physical configuration and software configuration can be different. Any difference in profile or parameters for a configured or non-configured slave can be taken into account in the configuration screen; in fact, it is possible to transmit any modification to the configuration screen before transferring the new application to the controller. The procedure to follow in order to take the physical configuration into account is the following:

Step	Description
1	Transfer of the desired slave configuration to the configuration screen.
2	Acceptance of the configuration in the configuration screen.
3	Confirmation of the new configuration.
4	Transfer of the application to the module.

Transfer of a Slave Image to the Configuration Screen.

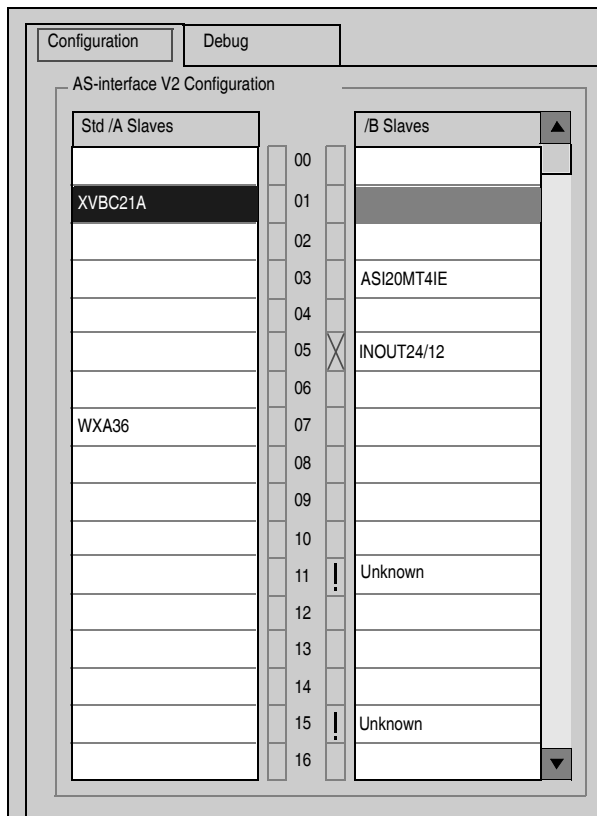
In the case when a slave that is not specified in the configuration is detected on the bus, an "Unknown" slave appears in the "AS-interface V2 Configuration zone" of the debug screen. The following table describes the procedure for transferring the image of the "Unknown" slave to the configuration screen:

Step	Description
1	Access the "Debug" screen.
2	Select the desired slave in the "AS-interface V2 Configuration" zone.
3	Right click on the mouse to select "Transfer Conf". Illustration:  Result: The image of the selected slave (image of the profile and parameters) is then transferred to the configuration screen.
4	Repeat the operation for each of the slaves whose image you would like to transfer to the configuration screen.

Return to the Configuration Screen

When the user returns to the configuration screen, all the new slaves (unexpected) which have been transferred are visible.

Illustration of the configuration screen following the transfer of all slaves:



Key:

- The cross signifies that there are differences between the image of the profile of the transferred slave, and the profile initially desired in the configuration screen.
- The exclamation mark signifies that a new profile was added to the configuration screen.

Explanation:

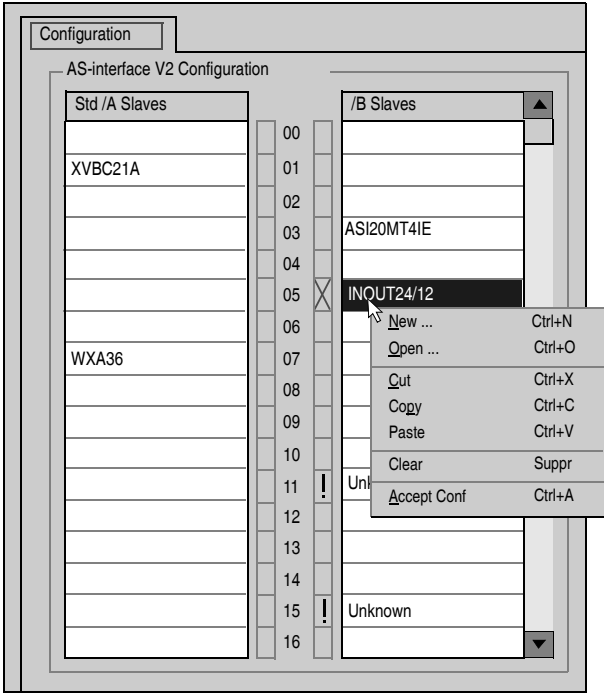
The configuration screen always shows the permanent image of the desired configuration (this is why the slave is still present as 3B in spite of the change of address (See *Modification of Slave Address*, p. 218)), completed by the current image of the bus.

The profiles and parameters of the expected slaves displayed correspond to those which were expected. The profiles and parameters of the unknown slaves displayed correspond to the images of those detected.

Procedure for Transferring the Definitive Application to the Module

Before transferring a new application to the module, the user can, for each slave, accept the detected profile and parameters (transferred to the configuration screen) or modify the configuration "manually" (See *Procedure for Declaring and Configuring a Slave*, p. 210).

The following table describes the steps to follow to confirm and transfer the definitive configuration to the module:

Step	Action
1	Via the software, disconnect the PC from the module. Note: No modification can be carried out in the configuration screen if the PC is connected to the module.
2	Right click on the desired slave.
3	2 choices: ● Select "Accept Conf" to accept the detected profile of the selected slave. Illustration:  For each of the slaves marked with a cross, a message will warn the user that this operation will overwrite the initial profile (displayed on-screen) of the slave. ● Select the other choices in the right click menu to configure the selected slave manually.

Step	Action
4	Repeat the operation for each of the desired slaves in the configuration.
5	Press the "OK" button to confirm and create the new application. Result: Automatic return to the main screen.
6	Transfer the application to the module.

Automatic addressing of an AS-Interface V2 slave

At a Glance

Each slave on the AS-Interface bus must be assigned (via configuration) a unique physical address. This must be the same as the one declared in TwidoSoft.

TwidoSoft software offers an automatic slave addressing utility so that an AS-Interface console does not have to be used.

The automatic addressing utility is used for:

- replacing a faulty slave,
- inserting a new slave.

Procedure

The table below shows the procedure for setting the **Automatic addressing** parameter.

Step	Action
1	Access the AS-Interface V2 master module's configuration screen.
2	Click on the Automatic addressing check box found in the Master mode zone. Result: The Automatic addressing utility will be activated (box checked) or disabled (box not checked). Note: By default, the Automatic addressing parameter has been selected in the configuration screen.

How to insert a slave device into an existing AS-Interface V2 configuration

At a Glance

It is possible to insert a device into an existing AS-Interface V2 configuration without having to use the pocket programmer.

This operation is possible once:

- the **Automatic addressing** utility of configuration mode is active (See *Automatic addressing of an AS-Interface V2 slave*, p. 225),
- a single slave is absent in the physical configuration,
- the slave which is to be inserted is specified in the configuration screen,
- the slave has the profile expected by the configuration,
- the slave has the address 0 (A).

The AS-Interface V2 module will therefore automatically assign to the slave the value predefined in the configuration.

Procedure

The following table shows the procedure for making the automatic insertion of a new slave effective.

Step	Action
1	Add the new slave in the configuration screen in local mode.
2	Carry out a configuration transfer to the PLC in connected mode.
3	Physically link the new slave with address 0 (A) to the AS-Interface V2 bus.

Note: An application can be modified by carrying out the above manipulation as many times as necessary.
--

Automatic replacement of a faulty AS-Interface V2 slave

Principle

When a slave has been declared faulty, it can be automatically replaced with a slave of the same type.

This happens without the AS-Interface V2 bus having to stop, and without requiring any manipulation since the configuration mode's **Automatic addressing** utility is active (See *Automatic addressing of an AS-Interface V2 slave*, p. 225).

Two options are available:

- The replacement slave is programmed with the same address using the pocket programmer, and has the same profile and sub-profile as the faulty slave. It is thus automatically inserted into the list of detected slaves (LDS) and into the list of active slaves (LAS),
 - The replacement slave is blank (address 0 (A), new slave) and has the same profile as the faulty slave. It will automatically assume the address of the replaced slave, and will then be inserted into the list of detected slaves (LDS) and the list of active slaves (LAS).
-

Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus

At a Glance

This page presents the details relating to the addressing of digital or analog I/Os of slave devices.

To avoid confusion with Remote I/Os, new symbols are available with an AS-Interface syntax: %IA for example.

Illustration

Reminder of the principles of addressing:

%	IA, QA, IWA, QWA	x	.	n	.	i
Symbol	Type of object	Expansion module address		slave address		Channel no.

Specific Values

The table below gives specific values to AS-Interface V2 slave objects:

Part	Values	Comment
IA	-	Image of the physical digital input of the slave.
QA	-	Image of the physical digital output of the slave.
IWA	-	Image of the physical analog input of the slave.
QWA	-	Image of the physical analog output of the slave.
x	1 to 7	Address of AS-Interface module on the expansion bus.
n	0A to 31B	Slot 0 cannot be configured.
i	0 to 3	-

Examples

The table below shows some examples of I/O addressing:

I/O object	Description
%IWA4.1A.0	Analog input 0 of slave 1A of the AS-Interface module situated in position 4 on the expansion bus.
%QA2.5B.1	Digital output 1 of slave 5B of the AS-Interface module situated in position 2 on the expansion bus.
%IA1.12A.2	Digital input 2 of slave 12A of the AS-Interface module situated in position 1 on the expansion bus.

Implicit Exchanges

The objects described below are exchanged implicitly, in other words they are exchanged automatically on each PLC cycle.

Programming and diagnostics for the AS-Interface V2 bus

Explicit Exchanges

Objects (words and bits) associated with the AS-Interface bus contribute data (for example: bus operation, slave status, etc.) and additional commands to carry out advanced programming of the AS-Interface function.

These objects are exchanged explicitly between the Twido controller and the AS-Interface Master by the expansion bus:

- At the request of the program user by way of the instruction: ASI_CMD (see "Presentation of the ASI_CMD" instruction below)
- Via the debug screen or the animation table.

Reserved Specific System Words

System words reserved in the Twido controller for the AS-Interface Master modules enable you to determine the status of the network: %SW73 is reserved for the first AS-Interface expansion module, and %SW74 for the second. Only the first 5 bits of these words are used; they are read-only.

The following table shows the bits used:

System Words	Bit	Description
%SW73 and %SW74	0	system status (= 1 if configuration OK, otherwise 0)
	1	data exchange (= 1 data exchange is enabled, 0 if in mode Data Exchange Off (See <i>AS-Interface V2 bus interface module operating mode</i> ; p. 233))
	2	system stopped (= 1 if the Offline (See <i>Offline Mode</i> , p. 233) mode is enabled, otherwise 0)
	3	ASI_CMD instruction terminated (= 1 if terminated, 0 if in progress)
	4	ASI_CMD error instruction (= 1 if there is an error in the instruction, otherwise 0)

Example of use (for the first AS-Interface expansion module):

Before using an ASI_CMD instruction, the %SW73:X3 bit must be checked to see whether an instruction is not in progress: check that %SW73:X3 = 1.

To ascertain whether the instruction has then correctly executed, check that the %SW73:X4 bit equals 0.

Presentation of the ASI_CMD Instruction

For each user program, the ASI_CMD instruction allows the user to program his network and obtain the slave diagnostics. The instruction parameters are passed by internal words (memory words) %MWx.

The syntax of the instruction is as follows:

ASI_CMD_n %MW_x:l

Legend:

Symbol	Description
n	Address of AS-Interface expansion module (1 to 7).
x	Number of the first internal word (memory word) passed in parameter (0 to 254).
l	Length of the instruction in number of words (2).

Using the ASI_CMD Instruction

The following table describes the action of the ASI_CMD instruction according to the value of the parameters %MW(x), and %MW(x+1) when necessary. For slave diagnostics requests, the result is returned in %MW(x+1).

%MWx	%MWx+1	Action
1	0	Exits Offline mode.
1	1	Switches to Offline mode.
2	0	Prohibits the exchange of data between the Master and its slaves (enters Data Exchange Off mode).
2	1	Authorizes the exchange of data between the Master and its slaves (exits Data Exchange Off mode).
3	Reserved	-
4	Result	Reads the list of active slaves (LAS table) with addresses from 0A to 15A (1 bit per slave).
5	Result	Reads the list of active slaves (LAS table) with addresses from 16A to 31A (1 bit per slave).
6	Result	Reads the list of active slaves (LAS table) with addresses from 0B to 15B (1 bit per slave).
7	Result	Reads the list of active slaves (LAS table) with addresses from 16B to 31B (1 bit per slave).
8	Result	Reads the list of detected slaves (LDS table) with addresses from 0A to 15A (1 bit per slave).
9	Result	Reads the list of detected slaves (LDS table) with addresses from 16A to 31A (1 bit per slave).
10	Result	Reads the list of detected slaves (LDS table) with addresses from 0B to 15B (1 bit per slave).
11	Result	Reads the list of detected slaves (LDS table) with addresses from 16B to 31B (1 bit per slave).

%MWx	%MWx+1	Action
12	Result	Reads the list of peripheral faults on slaves (LPF table) with addresses 0A to 15A (1 bit per slave).
13	Result	Reads the list of peripheral faults on slaves (LPF table) with addresses 16A to 31A (1 bit per slave).
14	Result	Reads the list of peripheral faults on slaves (LPF table) with addresses 0B to 15B (1 bit per slave).
15	Result	Reads the list of peripheral faults on slaves (LPF table) with addresses 16B to 31B (1 bit per slave).
16	Result	Reads bus status. See the results details in the next paragraph.

Note: Bus status is updated on each PLC scan.. But the result of the ASI_CMD bus reading instruction is available only at the end if the following PLC scan.

Details of the results of the ASI_CMD instruction to read bus status

In the case when bus status is read by the ASI_CMD instruction (value of the %MWx parameter is equal to 16), the format of the result in the %MWx+1 word is as follows:

%MWx+1		Designation (1=OK, 0=NOK)
least significant	bit 0	Configuration OK
	bit 1	LDS.0 (slave present with address 0)
	bit 2	Auto addressing active
	bit 3	Auto addressing available
	bit 4	Configuration Mode active
	bit 5	Normal operation active
	bit 6	APF (power supply problem)
	bit 7	Offline ready
most significant	bit 0	Peripheral fault
	bit 1	Data exchange active
	bit 2	Offline Mode
	bit 3	Normal mode (1)
	bit 4	Communication fault with the AS-Interface Master
	bit 5	ASI_CMD instruction in progress
	bit 6	ASI_CMD instruction error

Details of the results of the ASI_CMD instruction to read slave status

In the case of slave diagnostics by ASI_CMD instruction (%MWx value between 4 and 15), the slaves' status is returned in the bits (1=OK) of the %MWx+1 word. The following table gives the detail of the results according to the value of the %MWx word:

%MWx	%MWx+1															
value	most significant byte								least significant byte							
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
4, 8, 12	15A	14A	13A	12A	11A	10A	9A	8A	7A	6A	5A	4A	3A	2A	1A	0A
5, 9, 13	31A	30A	29A	28A	27A	26A	25A	24A	23A	22A	21A	20A	19A	18A	17A	16A
6, 10, 14	15B	14B	13B	12B	11B	10B	9B	8B	7B	6B	5B	4B	3B	2B	1B	0B
7, 11, 15	31B	30B	29B	28B	27B	26B	25B	24B	23B	22B	21B	20B	19B	18B	17B	16B

To read whether slave 20B is active, the ASI_CMD instruction must be executed with the %MWx internal word having a value of 7. The result is returned in the %MWx+1 internal word; the status of slave 20B is given by the value of bit 4 of the least significant byte: If bit 4 is equal to 1, then slave 20B is active.

Programming Examples for the ASI_CMD Instruction

To force the AS-Interface Master (positioned at 1 on the expansion bus) to switch to Offline mode:

```
LD 1
[%MW0 := 16#0001 ]
[%MW1 := 16#0001 ]
LD %SW73:X3          //If no ASI_CMD instruction is in progress, then continue
[ASI_CMD1 %MW0:2]    //to force the switch to Offline mode
```

To read the table of slaves active for addresses 0A to 15A:

```
LD 1
[%MW0 := 16#0004 ]
[%MW1 := 16#0000    //optional]
LD %SW73:X3          //If no ASI_CMD instruction is in progress, then continue
[ASI_CMD1 %MW0:2]    //to read the LAS table for addresses 0A to 15A
```

AS-Interface V2 bus interface module operating mode:

At a Glance

The AS-Interface bus interface module TWDNOI10M3 has three operating modes, each of which responds to particular needs. These modes are:

- Protected mode,
- Offline mode,
- Data Exchange Off mode.

Using the ASI_CMD (See *Presentation of the ASI_CMD Instruction, p. 230*) instruction in a user program allows you to enter or exit these modes.

Protected Mode

The protected operating mode is the mode generally used for an application which is running. It assumes that the AS-Interface V2 module is configured in TwidoSoft. This:

- continually checks that the list of detected slaves is the same as the list of expected slaves,
- monitors the power supply.

In this mode, a slave will only be activated if it has been declared in the configuration and been detected.

At power up or during the configuration phase, the Twido controller forces the AS-Interface module into protected mode.

Offline Mode

When the module is put into Offline mode, it first resets all the slaves present to zero and stops exchanges on the bus. When in Offline mode, the outputs are forced to zero.

In addition to using the PB2 button on the TWDNOI10M3 AS-Interface module, Offline mode can also be accessed via the software by using the ASI_CMD (See *Programming Examples for the ASI_CMD Instruction, p. 232*) instruction, which also allows you to exit the mode and return to protected mode.

Data Exchange Off Mode

When the Data Exchange Off mode is engaged, exchanges on the bus continue to function, but data is no longer refreshed.

This mode can only be accessed by using the ASI_CMD (See *Using the ASI_CMD Instruction, p. 230*) instruction.

Installing and Configuring the CANopen Fieldbus

10

A a Glance

Subject of this Chapter

This chapter describes how to install and configure the TWDNCO1M CANopen master module and its slave devices on the CANopen fieldbus.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
10.1	CANopen Fieldbus Overview	237
10.2	Implementing the CANopen Bus	251

10.1

CANopen Fieldbus Overview

At a Glance

Subject of this Section

This section is intended to provide you with general knowledge about the CANopen fieldbus technology and to introduce CAN-specific terminology that will be used throughout the remainder of this chapter.

What's in this Section?

This section contains the following topics:

Topic	Page
CANopen Knowledge Base	238
About CANopen	239
CANopen Boot-Up	242
Process Data Object (PDO) Transmission	245
Access to Data by Explicit Exchanges (SDO)	247
"Node Guarding" and "Life Guarding"	248
Internal Bus Management	250

CANopen Knowledge Base

Introduction	The following explanations of technical terms and acronyms are helpful for understanding the basic knowledge of CANopen network communication.
EDS file	EDS (Electronic Data Sheet) An EDS file describes the communication properties of a device on the CAN network (baudrates, transmission types, I/O offer, ...). It is provided by the device manufacturer. It is used in the configuration tool to configure a node (like a driver in an operating system).
PDO	PDO (Process Data Object) CANopen frame containing I/O data. We distinguish between: • Transmit-PDOs (TPDOs with data provided by a node) and • Receive PDOs (RPDOs with data to be consumed by a node). The transmission direction is always seen from a node's point of view. A PDO does not necessarily contain the whole data image of a node (for both TPDO and RPDO). Normally, analog input data and discrete input data are divided onto different TPDOs. The same is true for outputs.
SDO	SDO (Service Data Object) CANopen frames containing parameters. SDOs are typically used to read parameters from or write parameters to drives while the application is running.
COB-ID	COB-ID (Communication Object Identifier) Each CANopen frame starts with a COB-ID working as the Identifier in the CAN frame. During the configuration phase each node is receiving the COB-ID(s) for the frame(s) he is the provider or the consumer.

About CANopen

Introduction

CANopen is a standard fieldbus protocol for industrial control systems. It is particularly well suited to real-time PLCs, as it provides an effective, low-cost solution for integrated and transportable industrial applications.

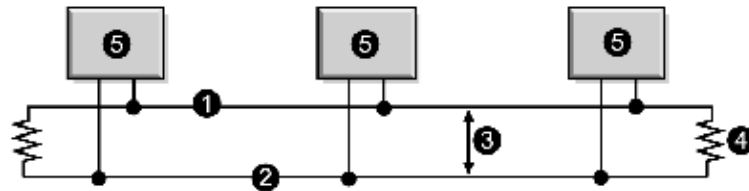
The CANopen Protocol

The CANopen protocol was created as a subset of CAN. By defining profiles, it is able to be even more specifically adapted to use with standard industrial components. CANopen is a CiA (CAN in Automation) standard which was taken up very rapidly as soon as it was made available on the market. In Europe, CANopen is now recognized as the industry standard for industrial systems based on a CAN design.

Physical Layer

CAN uses a differentially driven two-wire bus line (common return). A CAN signal is the difference between the voltage levels of the CAN-high and CAN-low wires. (See figure below.)

The following diagram shows the components of the physical layer of a two-wire CAN bus:



- 1 CAN-high wire
- 2 CAN-low wire
- 3 potential difference between CAN-high/CAN-low signals
- 4 120Ω resistance jack
- 5 node

The bus wires can be routed in parallel, twisted or shielded form in accordance with electromagnetic compatibility requirements. A single line structure minimizes reflection.

CANopen Profiles

The communication profile

The CANopen profile family is based on a "communication profile", which specifies the main communication mechanisms and their description (DS301).

The device profile

The most important types of devices used in industrial automation are described in the "Device profiles". They also define device functionalities.

Examples of the standard devices described are:

- digital and analog input/output modules (DS401),
- motors (DS402),
- control devices (DSP403),
- closed loop controllers (DSP404),
- PLCs (DS405),
- encoders (DS406).

Device Configuration via the CAN Bus

The possibility of configuring devices via the CAN bus is one of the basic principles of the autonomy required by manufacturers (for each profile family).

General Specifications for CANopen Profiles

CANopen is a set of profiles for CAN systems with the following specifications:

- open bus system,
- real-time data exchange without protocol overload,
- modular design with possibility of resizing,
- interoperability and interchangeability of devices,
- supported by a large number of international manufacturers,
- standardized network configuration,
- access to all device parameters,
- synchronization and circulation of cyclical process data and/or event-driven data (possibility of short system response times).

CANopen Product Certification

All manufacturers offering CANopen-certified products on the market are members of the CiA group. As an active member of the CiA group, Schneider Electric Industries SAS develops its products in compliance with the standardization recommendations set by this association.

CAN Standards

CANopen specifications are defined by the CiA group and can be accessed (subject to some restrictions) on the group site at <http://www.can-cia.com>. The sourcecodes for master and slave devices are available from the various suppliers.

Note: To find out more about CANopen standard specifications and mechanisms, please visit CiA's home page (<http://www.can-cia.de/>).

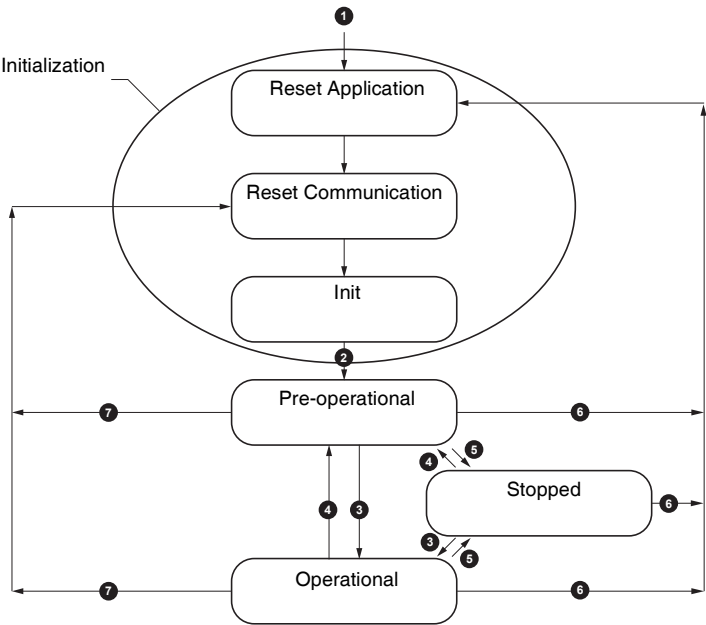
**Communication
on a CANopen
Network**

The communication profile is based on CAL services and protocols. It provides the user with access to two types of exchange: SDO and PDO. On power up, the device enters an initialization phase then goes into pre-operational state. At this stage, only SDO communication is authorized. After receiving a startup command, the device switches to the operational state. PDO exchanges can then be used, and SDO communication remains possible.

CANOpen Boot-Up

Boot-up Procedure

The minimum device configuration specifies a shortened boot procedure. This procedure is illustrated in the following diagram:



Legend

Number	Description
1	Module power up
2	After initialization, the module automatically goes into PRE-OPERATIONAL state.
3	NMT service indication: START REMOTE NODE
4	NMT service indication: PRE-OPERATIONAL
5	NMT service indication: STOP REMOTE NODE
6	NMT service indication: RESET NODE
7	NMT service indication: RESET COMMUNICATION

Active CANopen Objects depending on State Machine

The crosses in the table below indicate which CANopen objects are active for which states of the state machine.

	Initialization	Pre-operational	Operational	Stopped
PDO object			X	
SDO object		X	X	
Emergency		X	X	
Boot-Up	X		X	
NMT		X	X	X

Reset Application

The device goes into "Reset Application" state:

- after the device starts up,
- or by using the "Reset Node" Network management (NMT) service.

In this state, the device profile is initialized, and all the device profile information is reset to default values. When initialization is complete, the device automatically goes into the "Reset Communication" state.

Reset Communication

The device goes into the "Reset Communication" state:

- after the "Reset Application" state,
- or by using the "Reset Communication" Network management (NMT) service.

In this state, all the parameters (standard value, depending on the device configuration) of the supported communication objects (objects pertaining to device identification such as device type, heartbeat, etc.: 1000H - 1FFFFH) are saved in the object directory. The device then automatically goes into the "Init" state.

Init

The device goes into "Init" mode after being in the "Reset Communication" state.

This state enables you to:

- define the required communication objects (SDO, PDO, Emergency),
- install the corresponding CAL services
- configure the CAN-Controller.

Initialization of the device is complete and the device automatically goes into the "Pre-Operational" state.

Note: The TWDNCO1M CANopen master module does not support SYNC mode.

Pre-Operational The device goes into "Pre-Operational" state:

- after the "Init" state,
- on receiving the "Enter Pre-Operational" NMT indication if it was in Operational state.

When the device is in this state, its configuration can be modified. However, only SDOs can be used to read or write device-related data.

When configuration is complete, the device goes into one of the following states on receiving the corresponding indication:

- "Stopped" on receiving the "STOP REMOTE NODE" NMT indication,
- "Operational" on receiving the "START REMOTE NODE" NMT indication.

Stopped The device goes into the "Stopped" state on receiving the "Node stop" indication (NMT service) if it was in "Pre-Operational" or "Operational" state.

In this state, the device cannot be configured. No service is available to read and write device-related data (SDO). Only the slave monitoring function ("Node guarding") remains active.

Operational The device goes into the "Operational" state if it was in the "Pre-Operational" state on receiving the "Start Remote Node" indication.

When the CANopen network is started using the "Node start" NMT services in "Operational" state, all device functionalities can be used. Communication can use PDOs or SDOs.

Note: Modifications to the configuration in "Operational" mode may have unexpected consequences and should therefore only be made in "Pre-Operational" mode.
--

Process Data Object (PDO) Transmission

Definition of PDO PDOs are objects which provide the communication interface with process data and enable them to be exchanged in real time. A CANOpen device's PDO set describes the implicit exchanges between this device and its communication partners on the network.
The exchange of PDOs is authorized when the device is in "Operational" mode.

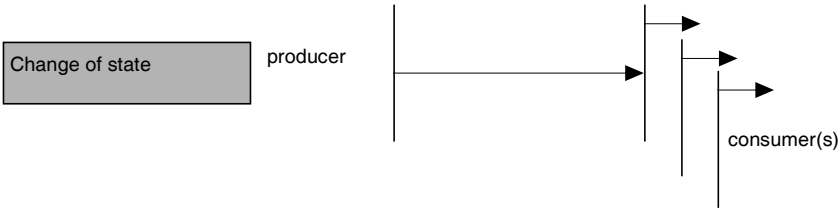
Types of PDO There are two types of PDO:

- PDOs transmitted by the device (often labeled:Transmit PDO or Tx-PDO or TPDO),
- PDOs received by the device (often labeled:Receive PDO or Rx-PDO or RPDO).

PDO Producers and Consumers PDOs are based on a "Producer / Consumer" model. The device which sends out a PDO is called the "producer" while one that receives it is known as the "consumer". Thus, writing an output to the TWDNCO1M master module sends a TPDO associated with the master, which contains the value of the output to be updated. In this case, the master is the PDO "producer" (while the slave device is the PDO "consumer".
In contrast, an input is updated by the transmission of a RPDO by the master module which is then the "consumer".

PDO Transmission Mode In addition to data to be transported, it is possible to configure the type of exchange for each PDO.
PDO can be exchanged by the TWDNCO1M master module in the following transmission mode:

Mode number	Mode type	Mode name
254 or 255	Asynchronous	Change of state



**Change of state
(Modes 254
and 255)**

"Change of state" corresponds to the modification of an input value (event control). Immediately after the change, the data are sent onto the bus. Event control makes it possible to make optimal use of bus bandwidth, as only the modification is transmitted, rather than the whole process image. This makes it possible to achieve a very short response time, as when an input value is modified, it is not necessary to await the next request from the master.

When selecting "change of state" PDO transmission, you should however bear in mind that it is probable that a number of events may occur at the same time, generating delays whilst waiting for a lower priority PDO to be transmitted to the bus. You should also avoid a situation where continual modification of an input with a high-priority PDO blocks the bus (this is known as a "babbling idiot").

Note: As a general rule, you should only choose to use PDO transmission with analog input modules if the Delta mode (object 6426H) or the inhibit time (objects 1800H to 1804H, sub-index 3) are set to avoid a bus overload.

Access to Data by Explicit Exchanges (SDO)

What is an SDO?	Service Data Objects (SDO) allow a device's data to be accessed by using explicit requests. The SDO service is available when the device is in "Operational" or "Pre-Operational" state.
Types of SDO	There are two types of SDO: • read SDOs (Download SDO), • write SDOs (Upload SDO).
Client/Server Model	The SDO protocol is based on a 'Client / Server' model. <i>For a Download SDO</i> The client sends a request indicating the object to be read. The server return the data contained within the object. <i>For an Upload SDO</i> The client sends a request indicating the object to be written to and the desired value. After the object has been updated, the server returns a confirmation message. <i>For an unprocessed SDO</i> In both cases, if an SDO was not able to be processed, the server returns an error code (abort code).

"Node Guarding" and "Life Guarding"

Definition of Life-Time	The "Life time" parameter is calculated as follows: Life Time = Guard Time x Life Time Factor The object 100CH contains the "Guard Time" parameter expressed in milliseconds. The object 100DH contains the "Life Time Factor" parameter.
Activation of Monitoring	If one of these two parameters is set to "0" (default configuration) the module does not perform monitoring (no "Life Guarding"). To activate monitoring over time, you must at least enter the value 1 in the object 100DH and specify a time in ms in the object 100CH.
Guarantee of Reliable Operation	To guarantee reliable operation, it is advisable to enter a "Life time factor" of 2. If not, should a delay occur (for example due to processing of messages of the highest priority or internal processing on the "Node Guarding") master, the module switches into "Pre-Operational" state without generating an error.
Importance of Monitoring	These two monitoring mechanisms are particularly important to the CANopen system, given that devices do not usually operate in event-controlled mode.
Slave Monitoring	Monitoring is performed in the following way:

Phase	Description
1	The master sets "Remote Frames" (remote transmit requests) on the "Guarding COB-IDs" of the slaves to be monitored.
2	The slaves concerned respond by sending the "Guarding" message. It contains the "Status Code" of the slave and the "Toggle Bit", which must change after each message.
3	The master compares the "Status" and "Toggle Bit" information: If they are not in the state expected by the NMT master or if no response is received, the master considers that an error has occurred on the slave.

**Master
Monitoring**

If the master requests "Guarding" messages on a strictly cyclical basis, the slave can detect a master failure.

If the slave does not receive a request from the master within the defined "Life Time" interval (Guarding error), it considers the a master failure has occurred ("Watchdog" function).

In this case, the corresponding outputs go into the error state and the slave switches back into "Pre-Operational" mode.

Note: The "Remote" request from the master obtains a response, even if there are no values entered in the "Guard Time" and "Life Time Factor" objects. Time monitoring is only activated when the values in the two objects are greater than 0. Typical values for the "Guard Time" parameter are between 250 ms and 2 seconds.

**"Guarding"
Protocol**

The value of the "Toggle Bit" (t) sent in the first "Guarding" message is "0". Then, the bit changes ("toggles") in each subsequent "Guarding" message, which makes it possible to indicate if a message has been lost.

The bus head indicates its network state (s) in the seven remaining bits:

Network state	Response
Stopped	0x04 or 0x84
Pre-operational	0x7F or 0xFF
Operational	0x05 or 0x85

Internal Bus Management

Switching the Internal Bus to the "Stop" State

The internal bus automatically switches from the "Stop" to the "Run" state when the communication module switches from the "Pre-operational" to the "Operational" state.

When the internal bus switches to the "Stop" state all the expansion module outputs are set to zero.

The communication module outputs are maintained in their current state.

Configuration of Expansion Modules

The internal bus is used to update the configuration of the discrete and analog expansion module parameters.

The parameters are sent to the communication module when the bus is in the "Stop" state.

These new configuration parameters are acknowledged when the bus goes into the "Run" state.

10.2 Implementing the CANopen Bus

Overview

Introduction This section describes how to implement the CANopen fieldbus on the Twido PLC system, using the TWDNCO1M CANopen master module.

What's in this Section? This section contains the following topics:

Topic	Page
Overview	252
Hardware Setup	253
Configuration Methodology	254
Declaration of CANopen Master	256
Network CANopen Slave Declaration	257
CANopen Objects Mapping	261
CANopen Objects Linking	265
CANopen Objects Symbolization	267
Addressing PDOs of the CANopen master	269
Programming and diagnostics for the CANopen fieldbus	270

Overview

Hardware and Software Requirements

The following hardware and software is required to implement a CANopen bus on your Twido PLC system:

Hardware	Requirements
Twido PLC compact or modular base controller	Compact base: ● TWDLC•24DRF ● TWDLCA•40DRF Modular base: ● TWDLMDA20*** ● TWDLMDA40***
CANopen master	1 CANopen master module: TWDNCO1M
CANopen slave devices	16 CANopen slaves maximum
CANopen connectors and cables	
Programming cable for the Twido PLC	

Software	Requirements
Twido PLC configuration software	TwidoSoft V3.0 or higher

CANopen Implementation Procedure

The following procedure will guide you through the installation, configuration and use of your CANopen network:

Step	Description
1	Hardware Setup
2	Configuration Methodology
3	Declaration of the CANopen Master
4	Network CANopen Slave Declaration
5	CANopen Objects Mapping
6	CANopen Objects Linking
7	CANopen Objects Symbolization
8	Network CANopen Diagnostics
The following sub-sections will provide a detailed description of each step of this procedure.	

Hardware Setup

Installing the TWDNCO1M Master Module

Install the TWDNCO1M master module on a Twido PLC system (DIN-rail or panel mounting) and connect it to the Twido PLC internal bus (for more details, see TwdoHW - Installing an expansion module). Follows these steps:

Step	Action	Description
1	Installation Preparation	Consult the <i>Twido Programmable Controllers Hardware Reference Guide (TWD USE 10AE)</i> for instructions on: ● correct mounting positions for Twido modules, ● adding and removing Twido components from a DIN rail, ● direct mounting on a panel surface, ● minimum clearances for modules in a control panel.
2	Mounting the TWDNCO1M Module	Install the TWDNCO1M master module on a DIN rail or panel. For more details, see TwdoHW - Installing an expansion module.
3	Module Connection to the Twido PLC's Bus	Connect the CANopen master module to the Twido PLC internal bus (for more details, see TwdoHW - Installing an expansion module).
4	CANopen Wiring and Connections	Follow the wiring and connections directions outlined in CANopen Wiring and Connections to connect the CAN bus power supply and signal lines.

Configuration Methodology

Overview

The CANopen configuration is performed via the CANopen Configuration tool available on TwidoSoft V3.0 or higher.

Note:

1. CANopen network, master and slave configuration, as well as configuration of communication parameters is performed only in Offline mode.
2. No change to the CANopen configuration is allowed in Online mode.
3. In Online mode, only certain parameters can be adjusted, such as %IWC and %QWC PDO addressing parameters.

Configuration Methodology

The following table describes the different software implementation phases of the CANopen bus:

Mode	Phase	Description
Local	Declaration of the TWDNCO1M module	Choose an available slot number to install the TWDNCO1M master module on the Twido expansion bus.
	Configuration of the CANopen network	Configure the CANopen network by: - importing EDS files of all slave device to the network catalog, - adding the slave devices from the catalog to the CANopen network.
	PDO mapping	Perform the mapping of TPDOs and RPDOs objects of each slave device declared on the network.
	PDO Linking	Link each slave PDO to the corresponding master module PDO.
Local or connected	Symbolization (optional)	Symbolization of the variables associated with the slave devices.
	Programming	Programming the CANopen function.
Connected	Transfer	Transfer of the application to the PLC.
	Debugging	Debugging the application with the help of: - the debug screen, used on the one hand to display slaves (address, parameters), and on the other, to assign them the desired addresses, - diagnostic screens allowing identification of errors.

Note: The declaration and deletion of the TWDNCO1M CANopen master module on the expansion bus is the same as for any other expansion module. However, only one CANopen master module is allowed on the Twido expansion bus. The TwidoSoft user interface program will not permit any other CANopen module to be added.

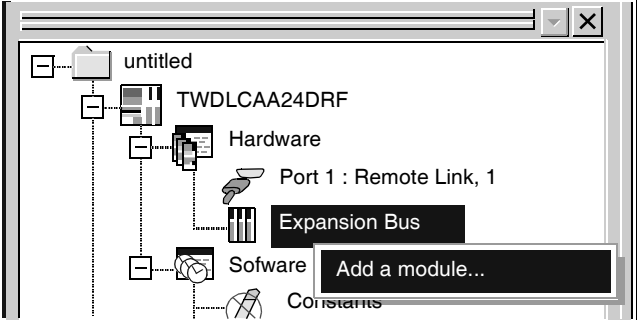
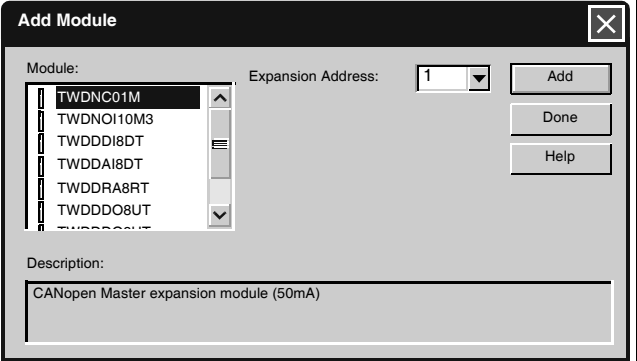
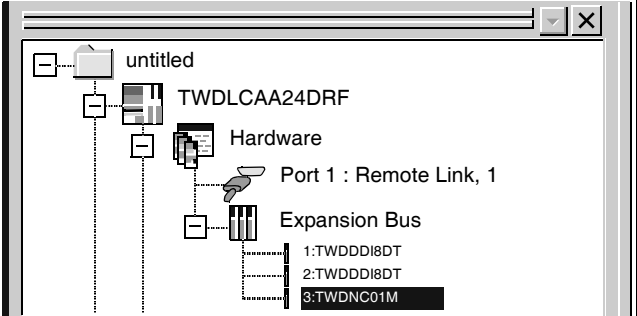
Precautions Prior to Connection

Before connecting (via the software) the PC to the controller and to avoid any detection problem:

- Ensure that no slave is physically present on the bus with address 127 (127 is a reserved, factory-set address assigned to the TWDNCO1M master module).
- Ensure that there are no slaves installed on the CANopen bus with duplicate addresses.

Declaration of CANopen Master

Procedure The table below shows the different stages when declaring the master CANopen.

Step	Action	Comment
1	From the TwidoSoft Application Browser, right-click Expansion Bus → Add a module...	
2	When the Add a module dialogbox appears: ● Select TWDNCO1M. ● Click Add. ● At this stage, you may continue adding any other expansion module (up to 7) that you want to include into your Twido system. Note: Only one TWDNCO1M CANopen master module is allowed. ● Click Done.	Only TWDC•A24DRF, TWDCA•40DRF, TWDLMDA20*** and TWDLMDA40*** controllers are supported 
3	An expansion bus structure similar to this example appears. Note: A TWDNCO1M master can be inserted in any available expansion slot numbered 1 to 7 on the Twido bus.	

Network CANopen Slave Declaration

Overview

The network CANopen slave declaration is a three-stage process that consists in:

1. importing the CANopen slave devices' EDS files into the Twido CANopen configurator's catalog,
2. building the CANopen network by adding up to 16 slave devices from the catalog to the network,
3. configuring the network management parameters (network speed and error control protocol parameters.)

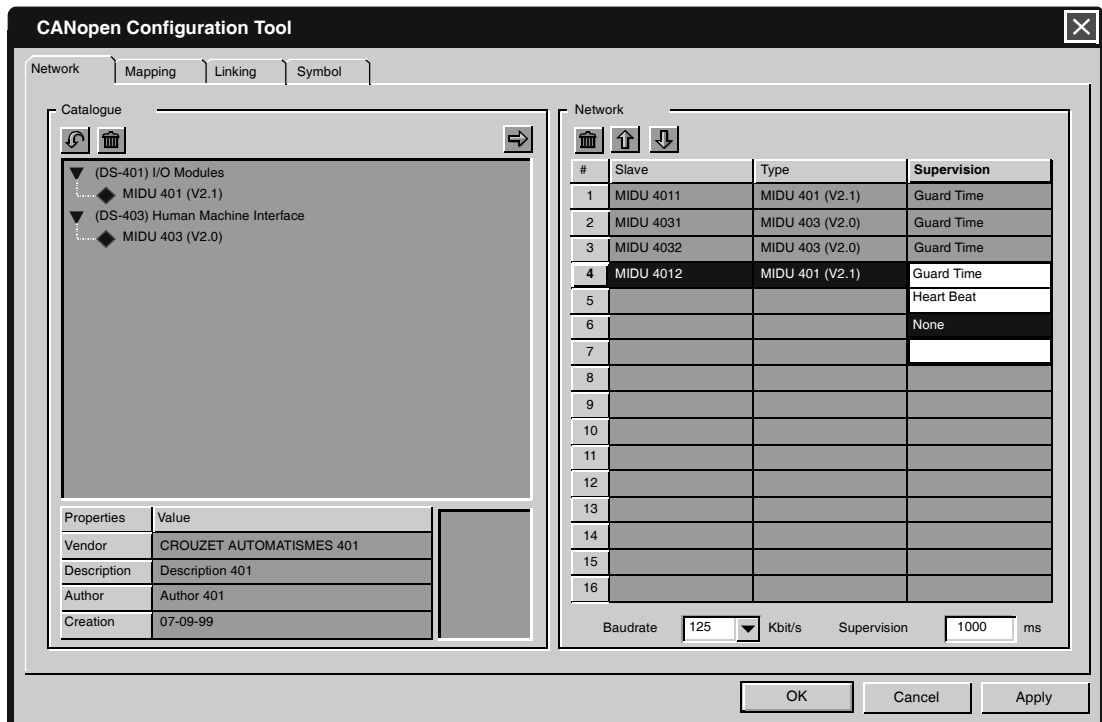
CANopen Configurator

From the TwidoSoft Application Browser, right-click on the master module name to select **Hardware** → **Expansion bus** → **TWDNCO1M** → **Configure**

Result: The CANopen Configuration Tool appears on screen, as shown in the following sub-section.

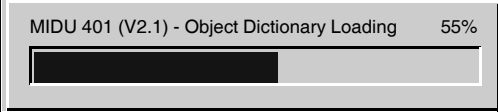
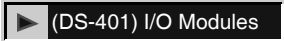
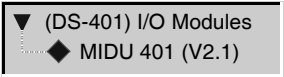
Network Dialogbox

The network CANopen configuration and slave declaration is performed via the TwidoSoft CANopen Configurator **Network** dialogbox, as shown below:



Importing Slave Profiles

The table below describes how to import CANopen slaves profiles (.EDS files) into the CANopen Configuration Tool catalog:

Step	Action
1	From the Catalog area in the Network dialog box, click the Import icon . Result: The operating system's Open dialogbox appears.
2	Browse to the location of the folder containing the EDS files of CANopen slave devices you want to add to the catalog. Result: The name of available EDS files appears in the Open dialogbox:
3	Choose an EDS file ("filename".EDS) from the list and click Open. Result: The CANopen Configuration Tool loads the object dictionary for the selected device. Note: This process may take several minutes, depending on the size of the selected EDS file. A progress bar indicates the state of completion of the loading process, as shown in the example below: 
4	Wait till the loading process is complete, then repeat steps 2 to 3 for any new slave profile you want to add to the catalog. Note: You only need to perform this process once, for all device profiles and object dictionaries listed in the loaded catalog are stored by TwidoSoft.
5	To display the device properties of a CANopen slave: - Click twice on the device type listed in the catalog. Example:  . Result:  . - Click once on the slave profile (for example, MIDU 401 V2.1). Result: The device properties of the selected CANopen slave appear in lower half of the Catalog area, showing: - the vendor's name (for example, Crouzet Automatismes 401), - the slave profile (for example, Description 401), - the author's name (for example, Author 401), - the creation date for that profile (for example, 07-09-99.)

Step	Action
6	To delete a slave profile from the Catalog, select the device name in the Catalog window and click the Delete icon  . Note: You may store in the Network CANopen Catalog more device profiles than you actually need for your current CANopen bus configuration. Profiles that are already loaded to the Catalog may be provisioned for future use.
7	Press the Apply button to confirm changes to the Catalog and save slave profiles to the TwidoSoft project.

Building the CANopen Network

The table below describes how to declare slave devices on the Twido CANopen network. (Note that you may only declare slaves which EDS profiles have been prior added to or are already stored in the Catalog.)

Step	Action
1	From the Catalog area in the Network dialog box, select the slave profile from the list of available devices already stored in the catalog. Result: The Add icon  appears at the top right corner of the catalog frame.
2	Click the Add icon  once. Result: The slave device is added to the network slaves table. Notes: • A maximum of 16 slaves can be declared on the Twido CANopen network. • The newly declared slave device takes the node address with the lowest available index. (For example, if slave devices are declared at node addresses 1, 2 and 4, the a newly added slave device with take the available node address 3, as default.)
3	You may assign a slave device to any available node address (1 to 16). To move a slave device to the desired node address, use the Move up/down arrow icons  /  .
4	Repeat steps 1 to 3 for any new slave device you want to declare on the CANopen network.
5	To delete a slave device from the Network, select the device name in the slaves table and click the Delete icon  .
6	Press the Apply button to confirm changes and save the network configuration to the TwidoSoft project.

Configuring the Network Management Parameters

The procedure below describes how to configure network management parameters such as the Baudrate (network speed), life-time and error control protocol.)

Step	Action																
1	In the Network dialog box, select the Baudrate (network speed) from the drop-down list: 10, 20, 50, 100, 125 (default value), 250, 500, 800 or 1000 Kbit/s. Note: Make sure that each slave device declared on the network is individually configured so that its own Baudrate is strictly identical to the network speed defined above, or otherwise the CANopen network communications will not function properly.																
2	Configure the Life-time period. This parameter defines the communications cycle-time period that will be implemented in the supervision field of each slave device, as explained in step 3 below. Note: Do not enter 0 in this field.																
3	Click once in the Supervision field to configure the error control protocol options of each slave device declared in the network slaves table. Result: The available supervision options supported by the selected device appear in a listbox as shown in the following example: <table border="1"><tr><td>4</td><td>MIDU 4012</td><td>MIDU 401 (V2.1)</td><td>Guard Time</td></tr><tr><td>5</td><td></td><td></td><td>Heart Beat</td></tr><tr><td>6</td><td></td><td></td><td>None</td></tr><tr><td>7</td><td></td><td></td><td></td></tr></table>	4	MIDU 4012	MIDU 401 (V2.1)	Guard Time	5			Heart Beat	6			None	7			
4	MIDU 4012	MIDU 401 (V2.1)	Guard Time														
5			Heart Beat														
6			None														
7																	
4	Select the error control protocol you wish to use to manage communications between the TWDNCO1M master module and the selected slave device: ● Guard Time ● Heartbeat ● None																
5	If the supervision option is set to None in the network slaves table, the outputs will not return to their fallback values in the event of a break in connection (*) between this slave and the TWDNCO1M master module. (*) this disconnection can be caused by: ● disconnection of the expansion bus cable linking the TWDNCO1M CANopen master module to the Twido PLC base controller, ● disconnection of this CANopen slave from the Twido CANopen bus, ● a faulty bus cable, ● a TwidoSoft "Reset" command (Online → Firmware / Reset), ● a TwidoSoft load configuration command (Online → Download), ● a command for firmware download to the TWDNCO1M master module via TwidoSoft (Online → Firmware Download).																
6	Press the Apply button to confirm changes and save the network configuration to the TwidoSoft project.																

CANopen Objects Mapping

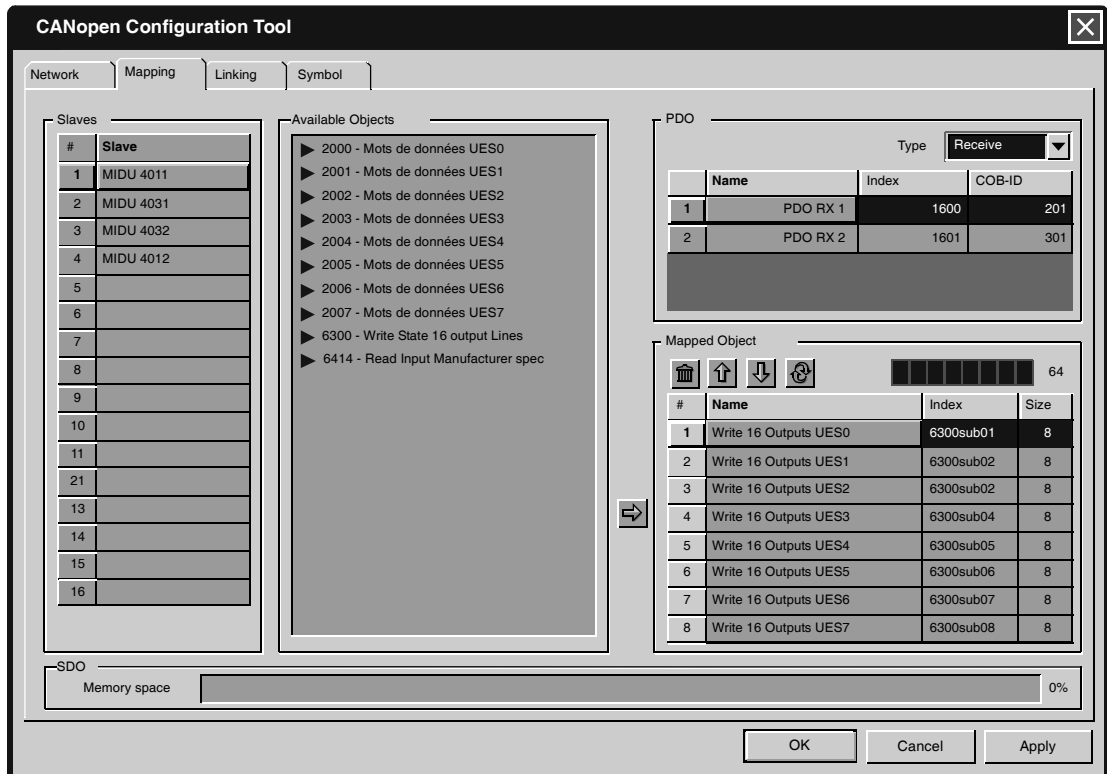
Overview

The **Mapping** dialogbox of the CANopen Configuration Tool allows you configure the PDOs of each slave device declared on the network.

Mapping Dialogbox

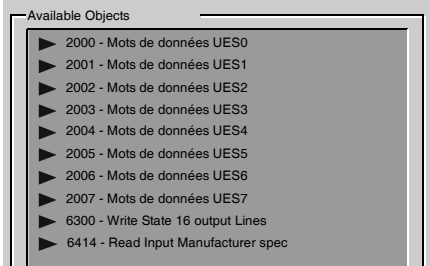
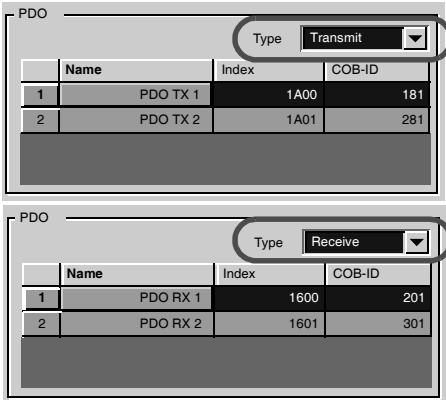
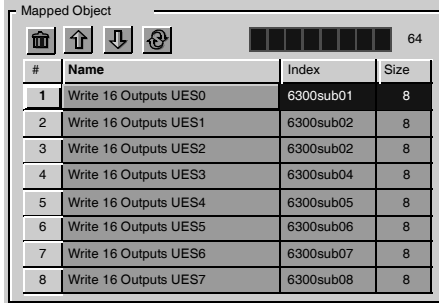
From the TwidoSoft Application Browser, right-click on the master module name to select **Hardware** → **Expansion bus** → **TWDNCO1M** → **Configure**, then select the **Mapping** tab from the CANopen Configuration Tool.

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



Objects Mapping To find out how to use the Mapping dialog box to configure the TPDOs and RPDOs of each slave device, follow these guidelines:

Step	Action												
1	In the Slaves frame, click once on the device name to select the slave you wish to configure the PDOs.												
2	Example: The DS-401 I/O module labeled MIDU 4011. Note that the slave names and node addresses appear in this frame exactly as defined in the previous stage of network configuration (see <i>Network CANopen Slave Declaration</i> , p. 257.) Slaves <table><thead><tr><th>#</th><th>Slave</th></tr></thead><tbody><tr><td>1</td><td>MIDU 4011</td></tr><tr><td>2</td><td>MIDU 4031</td></tr><tr><td>3</td><td>MIDU 4032</td></tr><tr><td>4</td><td>MIDU 4012</td></tr><tr><td>5</td><td></td></tr></tbody></table>	#	Slave	1	MIDU 4011	2	MIDU 4031	3	MIDU 4032	4	MIDU 4012	5	
#	Slave												
1	MIDU 4011												
2	MIDU 4031												
3	MIDU 4032												
4	MIDU 4012												
5													

Step	Action
3	Results: - All of the CANopen objects supported by the selected slave are displayed in the Available Objects frame, as shown in the example below:  - The PDO frame is showing the predefined Transmit-PDOs (PDO TX) for the selected slave, as default. You may use the Type toggle list to display the predefined Receive-PDOs (PDO RX) as well. In this example, the MIDU 4011 DS-401 I/O module supports two Transmit-PDOs (PDO TX) and two Receive-PDOs (PDO RX), as shown below:  - The predefined mapping of each selected PDO is showing in the Mapped Objects frame as well. : 

Step	Action
4	You may choose to customize the PDO mapping, using the Mapped Objects frame. A RPDO or TPDO is a 64-byte object that can contain up to eight 8-byte word objects or four 16-byte word objects or any combination of those two types of word objects, not too exceed the overall 64-byte limit of the PDO. To customize PDO mapping, continue to step 5 and subsequent below, and follow these directions.
5	For the desired slave (see step 2), select the PDO you wish to modify the mapping from the PDO frame. Example: Select the first Transmit-PDO (PDO TX 1). Result: The predefined PDO mapping (or the current customized mapping) appears in the Mapped Object frame.
6	To delete an unused word object from the PDO mapping structure, select the word object (indexed 1 to 8) and click the Delete icon .
7	From the Available Objects frame, select the word object in the object family that you wish to map, and click the Add icon  to append the word object to the Mapped Objects structure. Note: To restore the default mapping structure for the selected PDO, click the Default icon .
8	To change a word object's address within the mapped PDO structure, use the Move up/down arrow icons .
9	Press the Apply button to confirm changes to the mapped PDO structure and save the PDO mapping to the TwidoSoft project.
10	Repeat steps 5 through 9 for each PDO mapping you wish to configure.
11	Notes on memory usage: ● PDO memory usage: Usage of PDO memory can be monitored via the memory status bar located in the upper right corner of the Mapped Objects frame: . ● SDO additional memory usage: Predefined PDOs and word objects do not use any additional SDO memory. However, both the removal and the addition of word objects to the PDO mapping structure require the use of additional system memory. The current use of SDO memory is in the status bar located at the bottom of the Mapping dialogbox: .

CANopen Objects Linking

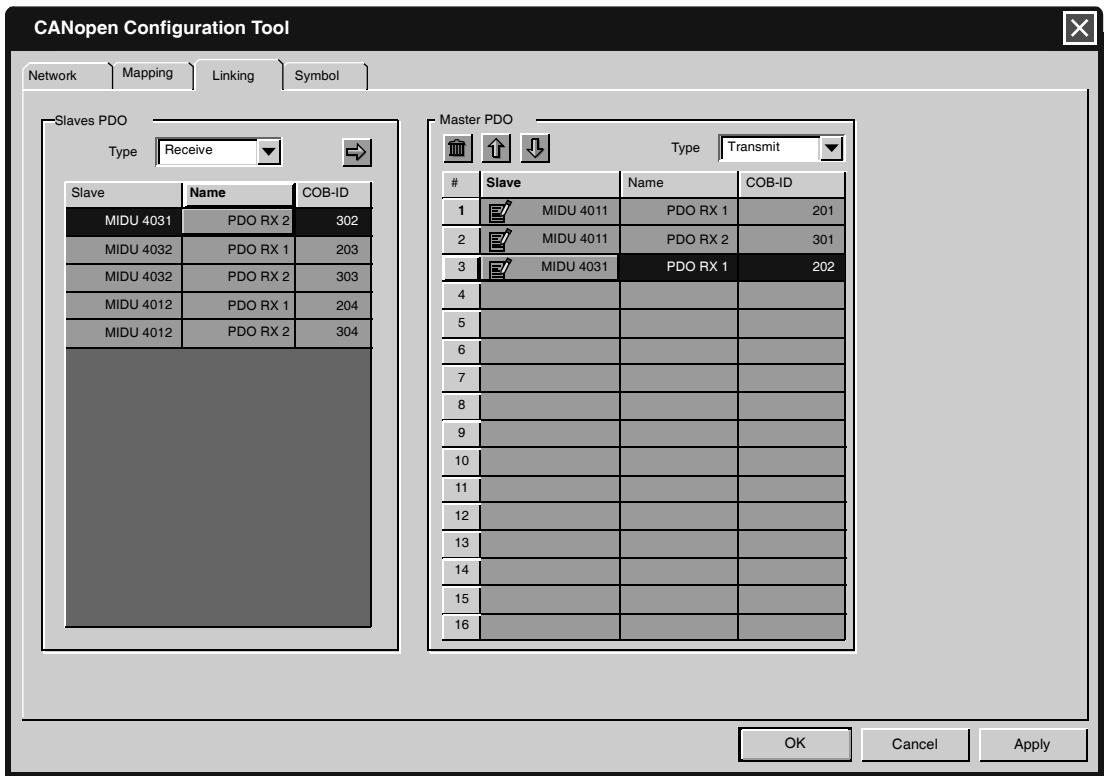
Overview

The **Linking** dialogbox of the CANopen Configuration Tool is used to define the physical link between the selected PDOs of the slave devices and the TWDNCO1M CANopen master module PDOs.

Linking Dialogbox

From the TwidoSoft Application Browser, right-click on the master module name to select **Hardware** → **Expansion bus** → **TWDNCO1M** → **Configure**, then select the **Linking** tab from the CANopen Configuration Tool.

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



Objects Linking

To find out how to use the Linking dialogbox to define the physical link between slave device and master module PDOs, follow these guidelines:

Step	Action																		
1	In the Slave PDOs frame, select the PDO Type: Receive or Transmit. Result: All the PDO slaves of the selected type are displayed in the Slave PDOs frame, as shown in the following example: Slaves PDO Type Receive <table><thead><tr><th>Slave</th><th>Name</th><th>COB-ID</th></tr></thead><tbody><tr><td>MIDU 4031</td><td>PDO RX 2</td><td>302</td></tr><tr><td>MIDU 4032</td><td>PDO RX 1</td><td>203</td></tr><tr><td>MIDU 4032</td><td>PDO RX 2</td><td>303</td></tr><tr><td>MIDU 4012</td><td>PDO RX 1</td><td>204</td></tr><tr><td>MIDU 4012</td><td>PDO RX 2</td><td>304</td></tr></tbody></table> Note: Selecting Receive or Transmit in the Slave PDOs frame automatically toggles the Master PDOs to the opposite type: Transmit or Receive, respectively.	Slave	Name	COB-ID	MIDU 4031	PDO RX 2	302	MIDU 4032	PDO RX 1	203	MIDU 4032	PDO RX 2	303	MIDU 4012	PDO RX 1	204	MIDU 4012	PDO RX 2	304
Slave	Name	COB-ID																	
MIDU 4031	PDO RX 2	302																	
MIDU 4032	PDO RX 1	203																	
MIDU 4032	PDO RX 2	303																	
MIDU 4012	PDO RX 1	204																	
MIDU 4012	PDO RX 2	304																	
2	From the Slave PDOs frame, select the PDO you wish to link to the TWDNCO1M CANopen master and click the Add icon  to append the PDO to the Master PDOs link list. Note: The TWDNCO1M master supports a maximum of 16 TPDO links and 16 RPDO links.																		
3	To change the address index of a PDO link within the Master PDOs frame, use the Move up/down arrow icons .																		
4	To delete an unused PDO link within the Master PDOs frame, select the desired PDO (indexed 1 to 16) and click the Delete icon .																		
5	Press the Apply button to confirm changes to the mapped PDO structure and save the PDO mapping to the TwidoSoft project.																		
6	Repeat steps 1 through 5 for each slave PDO you wish to link to the CANopen master.																		

CANopen Objects Symbolization

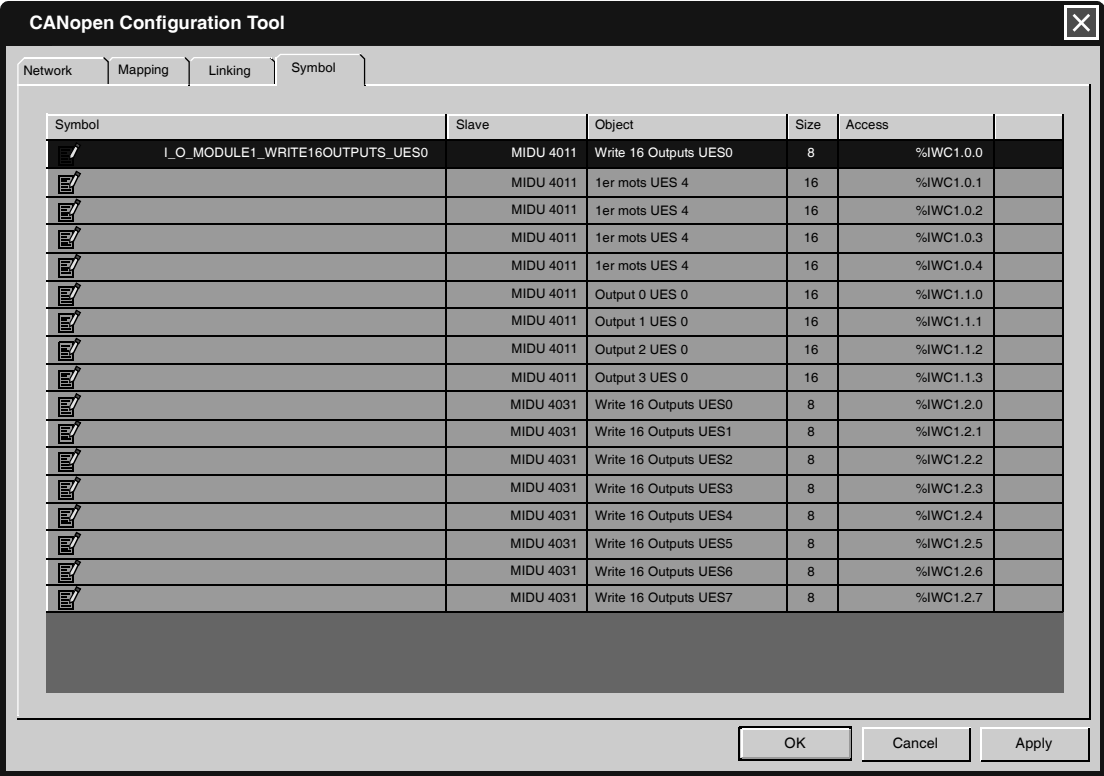
Overview

The **Symbol** dialogbox allows you to define a symbolization of the variables associated with the CANopen master.

Symbol Dialogbox

From the TwidoSoft Application Browser, right-click on the master module name to select **Hardware** → **Expansion bus** → **TWDNCO1M** → **Configure**, then select the **Symbol** tab from the CANopen Configuration Tool.

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



Objects
Symbolization

To find out how to use the Symbol dialogbox to define symbols for the CANopen object variables, follow these guidelines:

Step	Action
1	In the Symbol field, double-click the symbol editing icon  on the same line you wish to symbolize the variable. Result: The symbol textbox is activated and the cursor is right-aligned inside this textbox.
2	Fill in a descriptive name. A valid symbol can have up to 32 characters: only letters A-Z, numbers 0-9 and underscore " _ " are allowed (no "/", "%", space or any other special characters.) Note: For more details about editing symbols, please refer to <i>Symbolizing Objects</i>, p. 50.
3	Press the Apply button to confirm changes to the symbols table and save to the TwidoSoft project.
4	Repeat steps 1 to 3 for each variable you wish to symbolize.

Addressing PDOs of the CANopen master

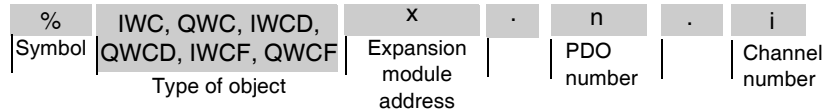
At a Glance

This sub-section describes addressing of PDO inputs and PDO outputs of the CANopen master.

To avoid confusion with Remote I/Os, a new designation is implemented for CANopen objects' syntax: %I**WC** for example.

Illustration

Reminder of the addressing principles:



Specific Values

The table below gives specific values to CANopen slave objects:

Part	Values	Comment
IWC	-	Image of the physical PDO input.
QWC	-	Image of the physical PDO output.
IWCD	-	Same usage as IWC, but in double-word format.
QWCD	-	Same usage as QWC, but in double-word format.
IWCF	-	Same usage as IWC, but in float format.
QWCF	-	Same usage as QWC, but in float format.
x	1 to 7	Address of TWDNCO1M CANopen master module on the Twido expansion bus.
n	0 to 15	PDO number (according to PDO index.)
i	0 to 7	Channel number (according to PDO sub-index.)

Example

The table below shows an example of PDO addressing:

I/O object	Description
%IWC4.1.0	PDO number 1, sub-index 0 input of the CANopen module located at address 4 on the Twido expansion bus.

Implicit Exchanges

The objects described below are exchanged implicitly, in other words they are exchanged automatically on each PLC cycle.

Programming and diagnostics for the CANopen fieldbus

Explicit Exchanges

Objects (words and bits) associated with the CANopen fieldbus contribute data (for example: bus operation, slave status, etc.) and additional commands to carry out advanced programming of the CANopen function.

These objects are exchanged explicitly between the Twido controller and the CANopen Master module via the expansion bus:

- at the request of the program user by way of the instruction: CAN_CMD (see "Presentation of the CAN_CMD" instruction below)
- via the debug screen or the animation table.

CANopen Master Reserved Specific System Words

System words reserved in the Twido controller for the CANopen Master module enable you to determine the status of the network: %SW8x (x=1-7) is reserved for the CANopen master module installed at expansion address x on the Twido bus. Only the first 7 bits of these words are used; they are read-only.

The following table shows the bits used:

System Words	Bit	Description
%SW8x (x=1-7)	0	Configuration status of CANopen master (= 1 if configuration OK, otherwise 0)
	1	Operational mode of CANopen master (= 1 data exchange is enabled, otherwise 0)
	2	System stopped (= 1 if the Offline mode is enabled, otherwise 0)
	3	CAN_CMD instruction complete (= 1 if command complete, otherwise 0 when command is in progress)
	4	CAN_CMD instruction error (= 1 if there is an error in the instruction, otherwise 0)
	5	Initialization error (= 1)
	6	Loss of message, power supply error (= 1)

Example of use (for the CANopen master module installed at expansion address 1 on the Twido bus):

Before using an CAN_CMD instruction, the %SW81:X3 bit must be checked to see whether an instruction is not in progress: check that %SW81:X3 = 1.

To ascertain whether the instruction has then correctly executed, check that the %SW81:X4 bit equals 0.

CANopen Slave Reserved Specific System Words

%SW20 to %SW27 are reserved system words that allow you to know the current state of the 16 CANopen slaves with node addresses ranging from 1 to 16. The content of these memory words is read-only.

The following table describes system words %SW20 to %SW27:

System words	Node address (slave number)		Word content / Description
	Bit [0-7]	Bit [8-15]	
%SW20	1	2	When %SW2x takes the following value: • = 1 => Unexpected module was present on the network. It has signalled itself as "not error free" before it was removed from the network. • = 2 => Node State Operational (module is in operational state): - "error free". • = 3 => Node State Operational (module is in operational state): - "not error free". • = 4 => Node State Preoperational (module is in preoperational state): - expected modules only (those declared as expected in the configuration table); - module can be set to operational; - "error free". • = 5 => Node State Preoperational (module is in preoperational state): - expected modules only (those declared as expected in the configuration table); - module can be set to operational; - "not error free".
%SW21	3	4	
%SW22	5	6	
%SW23	7	8	
%SW24	9	10	
%SW25	11	12	
%SW26	13	14	
%SW27	15	16	

System words	Node address (slave number)		Word content / Description
	Bit [0-7]	Bit [8-15]	
			● = 6 => Node State Preoperational (module is in preoperational state): - expected modules only (those declared as expected in the configuration table); - module is present but its current state does not allow to be set it to operational; - "error free". ● = 7 => Node State Preoperational (module is in preoperational state): - expected modules only (those declared as expected in the configuration table); - module is present but its current state does not allow to be set it to operational; - "not error free". ● = 8 => Wrong module (a module was detected with different device identity information): - "error free". ● = 9 => Wrong module (a module was detected with different device identity information): - "not error free". ● = 10 => Slave configuration error (module has answered SDO Write request of the SDO command table with an error confirmation or has not followed the rules of the SDO protocol): - "error free". ● = 11 => Slave configuration error: - "not error free". ● = 12 => Missing Module / Error Control Timeout / SDO Timeout (a module that was configured is not available, has disappeared during operation or does not answer SDO access): - "error free".

System words	Node address (slave number)		Word content / Description
	Bit [0-7]	Bit [8-15]	
			● = 13 => Missing Module / Error Control Timeout / SDO Timeout (a module that was configured is not available, has disappeared during operation or does not answer SDO access): - "not error free". ● = 14 => Unexpected module (a module was detected that is not in the configuration table): - "error free". ● = 15 => Unexpected module (a module was detected that is not in the configuration table): - "not error free".

Presentation of the CAN_CMD Instruction

For each user program, the CAN_CMD instruction allows the user to program his network and obtain the slave diagnostics. The instruction parameters are passed by internal words (memory words) %MWx.

The syntax of the instruction is as follows:

CAN_CMD_n %MW_{x:l}

Legend:

Symbol	Description
n	Expansion address of CANopen master module on the Twido bus (1 to 7).
x	Number of the first internal word (memory word) passed in parameter (0 to 254).
l	Length of the instruction in number of words (2).

Using the CAN_CMD Instruction

The CAN_CMD instruction allows you to program and manage the CANopen network and to perform diagnostic checks of individual slave devices. Command parameters are passed via memory words %MWx. The following table describes the action of the CAN_CMD instruction according to the value of the parameters %MW(x) to %MW(x+5) as needed:

%MWx	%MWx+1		%MWx+2		%MWx+3		%MWx+4		%MWx+5		Action
	Bit [0-7]	Bit [8-15]	Bit [0-7]	Bit [8-15]	Bit [0-7]	Bit [8-15]	Bit [0-7]	Bit [8-15]	Bit [0-7]	Bit [8-15]	
1	0		—								Reset CANopen communication.
1	1										Reset CANopen nodes.
2	0										Switch from operational to pre-operational mode.
2	1										Switch to operational mode.
3 or 4											3 => Start Read SDO command. 4 => Start Write SDO command.
	Node										Node = 1-16 => Node address
			Index								PDO object index.
					Len	Sub					Sub = 0-255 => Object sub-index Len = Length of data in byte
							Data 1				Payload according to the length field (Len) value
									Data 2		Payload according to the length field (Len) value

Note: Bus status is updated on each PLC scan. However, the result of the CAN_CMD bus reading instruction is available only at the end of the following PLC scan.

Programming Examples for the CAN_CMD Instruction

Example 1:

To force the CANopen Master (located at address 1 on the Twido expansion bus) to switch to Init mode:

```
LD 1
[%MW0 := 16#0001 ]
[%MW1 := 16#0001 ]
LD %SW81:X3          // If no CAN_CMD instruction is in progress, then continue
[CAN_CMD1 %MW0:2]    // To force the CANopen master to switch to Init mode
LD %SW81:X4          // (optional) To know if the CAN_CMD instruction has been
                     // successfully completed, before sending a new one.
```

Example 2:

To read the following variable: SDO_Slave:1_index:24576_sub-index:1_length:4

```
LD 1
[%MW6 := %MW4]       // Store the result of the last SDO command
[%MW7 := %MW5]       // Store the result of the last SDO command

LD %SW81:X3 // If there is no CAN_CMD instruction in progress, then continue
[%MW0 := 16#0003]
[%MW1 := 16#0001]    // SDO read to address node 1
[%MW2 := 16#6000]    // Access to index number 24576
[%MW3 := 16#0104]    // Access to sub-index number 1 and length value 4
[CAN_CMD1 %MW0:6] // Start SDO command
```

Example 3:

To write the following variable: SDO_Slave:1_index:24576_sub-index:1_length:4

```
LD 1
[%MW0 := 16#0004]
[%MW1 := 16#0001]    // SDO write to address node 1
[%MW2 := 16#6000]    // Access to index number 24576
[%MW3 := 16#0104]    // Access to sub-index number 1 and length value 4
[%MW4 := 16#1234]    // Data 1 value
[%MW5 := 16#1234]    // Data 2 value
LD %SW81:X3 // If there is no CAN_CMD instruction in progress, then continue
[CAN_CMD1 %MW0:6] // Start SDO command
```

Configuring the TwidoPort Ethernet Gateway

11

At a Glance

Subject of this Chapter

This chapter provides information on the software configuration of the ConneXium TwidoPort Ethernet Gateway module.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	Normal Configuration and Connection of TwidoPort	279
11.2	TwidoPort's Telnet Configuration	286
11.3	Communication Features	300

11.1

Normal Configuration and Connection of TwidoPort

At a Glance

Subject of this Section

This section provides information on how to perform a normal configuration of the ConneXium TwidoPort module with the TwidoSoft application program, module connectivity and BootP configuration information, as well.

What's in this Section?

This section contains the following topics:

Topic	Page
Normal Configuration with TwidoSoft	280
BootP Configuration	285

Normal Configuration with TwidoSoft

Foreword

If you have TwidoSoft (v. 3.0 or higher), configure TwidoPort with these instructions:

Note: Plug 'n play feature

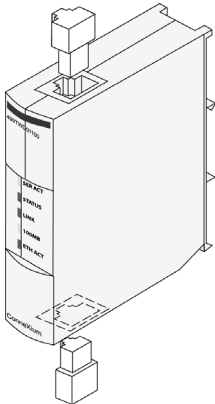
When TwidoPort is configured with TwidoSoft, TwidoPort's IP configuration is stored in the Twido controller. Therefore, maintenance personnel can exchange TwidoPorts without additional configuration.

To use the plug 'n play functionality, use TwidoSoft version 3.0 or higher and upgrade the Twido firmware to 3.0 or higher. Use Telnet to manually configure TwidoPort with older versions of TwidoSoft.

Installing the 499TWD01100 TwidoPort Module

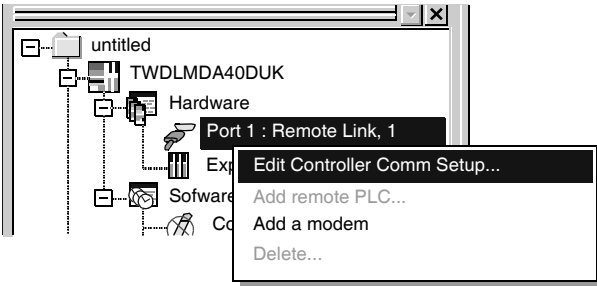
To install TwidoPort on a Twido PLC system (DIN-rail or panel mounting) and connect it to the Twido PLC internal bus, follow these steps:

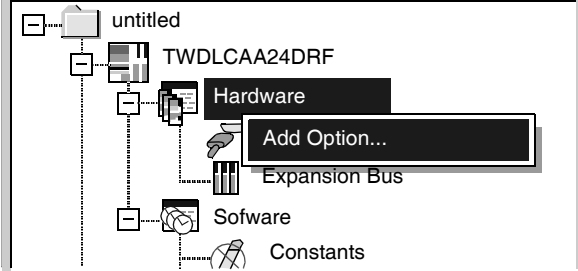
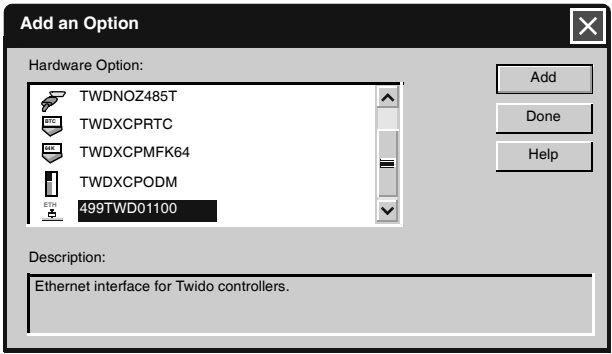
Step	Description	Action
1	Installation Preparation	Consult the <i>Twido Programmable Controllers Hardware Reference Guide (TWD USE 10AE)</i> for instructions on: ● correct mounting positions for Twido modules, ● adding and removing Twido components from a DIN rail, ● direct mounting on a panel surface, ● minimum clearances for modules in a control panel.
2	Mounting the 499TWD01100 TwidoPort Module	Install the module on a DIN rail or panel. For more details, see .
3	Protective Earth (PE) Grounding	Attach a grounded wire to the M3 screw terminal on the bottom of TwidoPort.

Step	Description	Action
4	Serial and Ethernet Connections Top plug: from Twido (serial)  Bottom plug: from Ethernet, either a straight or crossover cable	Connect the modular plug end of the (supplied) TwidoPort-to-Twido cable to TwidoPort's serial port and connect the other end to the Twido PLC's RS-485 serial port. Connect the RJ-45 plug from a standard Ethernet network cable (not supplied) into TwidoPort's Ethernet port.

Declaring the 499TWD01100 TwidoPort Module

The table below shows the different stages when declaring the 499TWD01100 TwidoPort module.

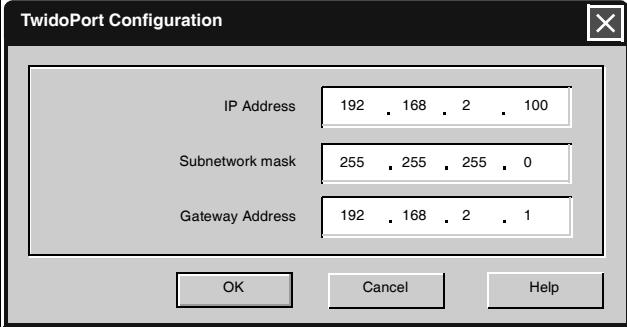
Step	Action	Comment
1	While using TwidoSoft (v. 3.0 or higher), configure the Twido controller's communication options by performing a right-click Port 1 : xxxxxx , 1 → Edit Controller Comm Setup... , using the TwidoSoft Application Browser. (See note 1.)	
Note 1	Any RS-485 Modbus port on Twido can be used.)	
Note 2	For the fastest initial autobaud, choose 19200-8-N-1 with a Twido Modbus address of 1.	

Step	Action	Comment
2	From the Controller Communications Setup dialog box, set the communication protocol to Modbus .	Note that the 499TWD01100 TwidoPort module cannot be added to the Twido hardware if the communication protocol is not set to Modbus.
3	Configure the Modbus communication parameters.	The Twido controller's RS-485 Modbus port must be configured to 9600, 19200, or 38400 baud to support TwidoPort's autobaud feature. (See notes 1 and 2.)
4	From the TwidoSoft Application Browser, right-click Hardware → Add Option....	
5	When the Add an option dialog box appears: • Select 499TWD01100 • Click Add. • At this stage, you may continue adding any other optional module supported by your Twido controller. Note: Only one 499TWD01100 TwidoPort module is allowed. • Click Done.	All Twido controllers are supported (with the exception of TWDLCAE40DRF with built-in Ethernet.) 
Note 1	Any RS-485 Modbus port on Twido can be used.)	
Note 2	For the fastest initial autobaud, choose 19200-8-N-1 with a Twido Modbus address of 1.	

**Configuring the
499TWD01100
TwidoPort
Module**

Note: The Ethernet parameters of TwidoPort can be configured when the TwidoSoft application program is in offline mode only.

To configure TwidoPort's Ethernet parameters, follow this procedure:

Step	Action	Comment
Foreword	To find out more about IP parameters (IP address, subnet mask and gateway address), please refer to <i>IP Addressing, p. 160</i> and <i>Private IP Addresses, p. 164</i> .	
1	Right click on TwidoPort Icon in the application browser to configure TwidoPort's IP parameters.	Result: The Ethernet Configuration dialog box appears, as shown in the example below. 
2	Enter TwidoPort's static IP Address in dotted decimal notation. (See <i>notes 1 and 2</i> .)	Caution: For good device communication, the IP addresses of the PC running the TwidoSoft application and TwidoPort must share the same network ID.
Note 1	Consult with your network or system administrator to obtain valid IP parameters for your network.	
Note 2	To allow good communication over the network, each connected device must have a unique IP address. When connected to the network, TwidoPort runs a check for duplicate IP address. If a duplicate IP address is located over the network, the STATUS LED will emit 4 flashes periodically. You must then enter a new duplicate-free IP address in this field.	
Note 3	Unless TwidoPort has special need for subnetting, use the default subnet mask.	
Note 4	If there is no gateway device on your network, simply enter TwidoPort's IP address in the Gateway Address field.	

Step	Action	Comment
3	Enter the valid Subnetwork mask assigned to TwidoPort by your network administrator. Please note that you cannot leave this field blank; you must enter a value. (See <i>notes 1 and 3</i> .)	Caution: For good device communication, the subnet mask configured on the PC running the TwidoSoft application and TwidoPort's subnet mask must match. As default, the TwidoSoft application automatically computes and displays a default subnet mask based on the class IP that you have provided in the IP Address field above. Default subnet mask values, according to the category of the TwidoPort's network IP address, follow this rule: Class A network -> Default subnet mask: 255.0.0.0 Class B network -> Default subnet mask: 255.255.0.0 Class C network -> Default subnet mask: 255.255.255.0
4	Enter the IP address of the Gateway . (See <i>notes 1 and 4</i> .)	On the LAN, the gateway must be on the same segment as TwidoPort. This information typically is provided to you by your network administrator. Please note that no default value is provided by the application, and that you must enter a valid gateway address in this field.
5	Validate the configuration and download it to the Twido controller.	
Note 1	Consult with your network or system administrator to obtain valid IP parameters for your network.	
Note 2	To allow good communication over the network, each connected device must have a unique IP address. When connected to the network, TwidoPort runs a check for duplicate IP address. If a duplicate IP address is located over the network, the STATUS LED will emit 4 flashes periodically. You must then enter a new duplicate-free IP address in this field.	
Note 3	Unless TwidoPort has special need for subnetting, use the default subnet mask.	
Note 4	If there is no gateway device on your network, simply enter TwidoPort's IP address in the Gateway Address field.	

BootP Configuration

BootP Process TwidoPort expects a response from the BootP server within two minutes of its BootP request transmission. Failing that, TwidoPort assumes the default IP configuration that is constructed from a MAC address of this structure:

85	16	MAC[4]	MAC[5]
----	----	--------	--------

MAC Address The MAC Address has the structure:
MAC[0] MAC[1] MAC[2] MAC[3] MAC[4] MAC[5] .

For Example, if the MAC address is 0080F4012C71, the default IP address would be 85.16.44.113.

11.2 TwidoPort’s Telnet Configuration

At a Glance

Subject of this Section This section describes how to configure the ConneXium TwidoPort module with a Tenet session.

What's in this Section? This section contains the following topics:

Topic	Page
Introducing Telnet Configuration	287
Telnet Main Menu	288
IP/Ethernet Settings	289
Serial Parameter Configuration	290
Configuring the Gateway	291
Security Configuration	292
Ethernet Statistics	293
Serial Statistics	294
Saving the Configuration	295
Restoring Default Settings	296
Upgrading the TwidoPort Firmware	297
Forget Your Password and/or IP Configuration?	299

Introducing Telnet Configuration

Overview of Telnet Configuration

Configure TwidoPort with a Telnet session (using a VT100-compatible Telnet client) for those cases in which a specific Twido configuration is not found or in which the BootP request is not answered after two minutes (resulting in the implementation of the default IP address).

Preparation to Telnet Configuration

Note: TwidoPort's Telenet requirements

While configuring TwidoPort with Telnet, make sure:

- TwidoPort is supplied with power (from a Twido controller) through its serial connection.
- Telnet's `local echo` is set to `off`.

To use Telnet, add TwidoPort's default IP address (or TwidoPort's configured IP address) to the PC's routing table using the command:

```
C:\> route add 85.0.0.0 mask 255.0.0.0 local_IP_address_of_PC
```

Example:

If the IP address of the PC is 192.168.10.30 and the default IP address (or the configured IP address) of TwidoPort is 85.16.44.113, the complete command would be:

```
C:\> route add 85.0.0.0 mask 255.0.0.0 192.168.10.30
```

Telnet Main Menu

Launching the Telnet Main Menu

When you start a Telnet session (e.g., by typing `telnet 85.16.44.113` at a command prompt or by using Windows[™] Hyperterminal[™]), the Telnet main menu appears after your press **Enter**:

```

      Telemecanique 499 TWD 01 100 Configuration and Diagnostics
      (c) 2004 Schneider Automation Inc

1) IP/Ethernet Settings
      IP Source: DEFAULT
      IP Address: 85.16.44.113
      Default Gateway: 85.16.44.113
      Netmask: 0.0.0.0
      Ethernet Frame Type: ETHERNETII

2) Serial Configuration
      Baud Rate: 19200
      Data Bits: 8
      Parity: NONE
      Stop Bits: 1
      Protocol: RTU

3) Gateway Configuration
      Slave Address Source: UNIT_ID
      Gateway Mode: SLAVE
      MB Broadcasts: ENABLED

4) Security Configuration

5) Ethernet Statistics

6) Serial Statistics

Commands: D)efault settings, S)ave, F)irmware Upgrade, Q)uit without save
      Select Command or Parameter(1..6) to change:
```

IP/Ethernet Settings

Configuring the IP/Ethernet Settings Use the following instructions to change the IP/Ethernet settings:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select (type) 1 to change the IP source to STORED and press Enter .	STORED may already be the IP source.
3	Set desired IP parameters manually. (See <i>TwidoPort Ethernet settings</i> following this table.)	Other parameters include: • IP Address • Default Gateway • Netmask • Ethernet Frame Type
4	Select R and press Enter .	The Telnet main menu appears. (You may have to press Enter again to update the screen.)

IP Source

The select **IP Source** option dictates the location from which the IP configuration is obtained:

- **STORED**—from local flash memory.
- **SERVED**—from BootP server.
- **TWIDO**—from the Twido controller.

The default IP address (**DEFAULT**) is derived from the MAC address. (By definition, the default is not selectable.)

Note: A valid IP configuration in the Twido controller overrides the user selection.

Example of Ethernet Settings

The following figure shows an example of TwidoPort's Ethernet settings:

```
Telemecanique 499 TWD 01 100 Configuration and Diagnost
(c) 2004 Schneider Automation Inc
```

IP/Ethernet Settings

```
-----
1>IP Source: DEFAULT
2>IP Address: 85.16.44.113
3>Default Gateway: 85.16.44.113
4>Netmask: 0.0.0.0
5>Ethernet Frame Type: ETHERNET2
-----
```

```
Commands: R>return to Main Menu
Select Command or Parameter<1..N> to change:
```

Serial Parameter Configuration

Foreword

Note: Under normal circumstances, it is not necessary to configure TwidoPort's serial parameters because the module supports an autobaud algorithm that eliminates the need for serial configuration.

Configuring the Serial Parameters

To configure TwidoPort's serial parameters:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select (type) 2 to change the serial settings.	See the following figure.
3	Verify or reset the settings.	Other parameters include: • Baud Rate • Data Bits • Parity • Stop Bits • Protocol
4	Select R and press <i>Enter</i> .	The Telnet main menu appears. (You may have to press <i>Enter</i> again to update the screen.)

Example of Serial Settings

The following figure shows an example of TwidoPort's serial settings:

```
Telemecanique 499 TWD 01 100 Configuration and Diagnostics
(c) 2004 Schneider Automation Inc
```

```
Serial Configuration
1> Baud Rate: 19200
2> Data Bits: 8
3> Parity: NONE
4> Stop Bits: 1
   Protocol: RTU
```

```
Commands: R>return to Main Menu
Select Command or Parameter(1..N) to change:
```

Configuring the Gateway

Foreword

Note: Usually, it is not necessary to configure TwidoPort's gateway parameters.

Configuring the Gateway Parameters

To configure the TwidoPort's gateway:

Step	Action	Comment	
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).	
2	Select (type) 3 to change the gateway parameters.	See the following figure.	
3	The following gateway parameters are available:		
	(1) Slave Address Source	FIXED	If the slave address source is FIXED , set the address to the value of the Twido controller's Modbus address. Valid addresses are in the 1 to 247 range.
		UNIT_ID	The unit ID of the Modbus/TCP frame will be used.
	(2) Gateway Mode	SLAVE	Only option for this version.
	(3) MB broadcasts	DISABLED	No broadcast messages are sent on TwidoPort's serial port.
ENABLED		Broadcast messages are sent from the Twido controller's serial port. (See note below.)	
4	Select R and press <i>Enter</i> .	The Telnet main menu appears. (You may have to press <i>Enter</i> again to update the screen.)	
Note	Twido does not support any broadcast Modbus messages.		

Example of Gateway Settings

The following figure shows an example of TwidoPort's gateway settings:

```

Telemecanique 499 TWD 01 100 Configuration and Diagnostics
(c) 2004 Schneider Automation Inc

Gateway Configuration
1) Slave Address Source: UNIT_ID
2) Slave Address: 20
3) Gateway Mode: SLAVE
4) MB Broadcasts: ENABLED

Commands: R)Return to Main Menu
Select Command or Parameter(1..4) to change: _

```

Security Configuration

Configuring the Security Settings

Use the following instructions to change the default password:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select (type) 4 and press Enter .	The Security Configuration Screen appears.
3	Select c and press Enter .	
4	Enter the old password.	Authorized users will know the old password (default is USERUSER).
5	Enter a new password.	Retype the new password. (See <i>note below</i> .)
6	Enter the new password again.	See the note below for acceptable passwords.
7	Select R and press Enter .	The Telnet main menu appears. (You may have to press Enter again to update the screen.)
Note	Password details: ● minimum length: 4 characters ● maximum length: 10 characters ● allowed characters: 0 to 9, a to z, A to Z (no spaces)	

Ethernet Statistics

Viewing Ethernet Statistics

To view TwidoPort's Ethernet statistics:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select (type) 5 to display the Ethernet Module Statistics screen.	See the figure that follows this table.
3	Press Enter to refresh the screen.	
4	Press C to clear statistics and press Enter.	All counters are reset to 0.
5	Select R and press Enter.	The Telnet main menu appears. (You may have to press Enter again to update the screen.)

The Ethernet Module Statistics Screen

TwidoPort's Ethernet Module Statistics screen:

Telemecanique 499 TWD 01 100 Configuration and Diagnostics (c) 2004 Schneider Automation Inc ETHERNET MODULE STATISTICS		
Status: 0x9103	IP Address: 192.168.1.141	
System Log Entry: No	Mac Address: 0:80:f4:0:4c:18	
Transmit Speed: 100BASE-T	Subnet Mask: 255.255.0.0	
Full/Half Duplex: Half Duplex	Gateway Address: 192.168.1.1	
Transmit Statistics	Receive Statistics	Functioning Errors
Transmits: 63	Receives: 532	Missed Packets: 0
Transmit Retries: 0	Framing Errors: 0	Collision Errors: 0
Lost Carrier: 0	Overflow Errors: 0	Transmit Timeouts: 0
Late Collision: 0	CRC Errors: 0	Memory Errors: 0
Tx Buffer Errors: 0	Rx Buffer Errors: 0	Net Interface Restarts: 0
SIL0 Underflow: 0		
Broadcast Packets Received: 37		Multicast Packets Received: 7
Commands: [Enter] to Refresh, C)lear Statistics, R)eturn to Main Menu		

Serial Statistics

Viewing Serial Statistics

To view TwidoPort’s serial statistics:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select (type) 6 to display the Serial Statistics screen and press Enter .	See the figure that follows this table. The serial statistics are updated.
3	Press C to clear statistics and press Enter .	All counters are reset to 0.
4	Select R and press Enter .	The Telnet main menu appears. (You may have to press Enter again to update the screen.)

The Serial Statistics Screen

TwidoPort’s **Serial Statistics** screen:

```

                                Telemecanique 499 TWD 01 100 Configuration and Diagnostics
                                (c) 2004 Schneider Automation Inc
----- SERIAL STATISTICS -----
Serial Bus Statistics
    Bus Message Count: 8284
    Bus Comm. Error Count: 0
Modbus Slave Statistics
    Slave Message Count: 4142
    Slave Exception Error Count: 3187
    Slave No Response Count: 0
-----
Commands: [Enter] to Refresh, C>lear Statistics, R>eturn to Main Menu
```

Saving the Configuration

Saving the Configuration

To save the changes to the configuration, type the configuration password:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select s and press Enter .	
3	Enter the configuration password.	The default password is USERUSER . (See note below.)
Note	For more details on how to set a personalized security password, please refer to <i>Security Configuration</i>, p. 292.	

The Save Configuration Confirmation Screen

TwidoPort's Save Configuration confirmation screen:

```

Telenecanique 499 TWD 01 100 Configuration and Diagnostics
(c) 2004 Schneider Automation Inc
SAVE CONFIGURATION

```

```

Configuration successfully stored to Twido.
Reboot your module for the new Configuration to be in effect.

```

```

Rebooting in 5 Seconds. You will lose your telnet connection.

```

```

Connection to host lost.

```

Restoring Default Settings

Restoring Default Settings

To restore TwidoPort's default settings:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu</i> , p. 288).
2	Select D to display the Default Configuration screen.	See the figure that follows this table.
3	Press Enter.	Press Enter. is required to display the main menu.
4	Save the default configuration.	See Saving the Configuration (See <i>Saving the Configuration</i> , p. 295), above.

The Default Configuration Screen

TwidoPort's Default Configuration screen:

```
Telenecanique 499 TWD 01 100 Configuration and Diagnostics
(c) 2004 Schneider Automation Inc
DEFAULT CONFIGURATION
```

```
IP Address: 192.168.2.102
Gateway Address: 192.168.2.102
Subnet Mask: 255.255.0.0
Frame Type: Ethernet II

Serial Mode: 19200-8-N-1

Gateway Mode: Modbus/RTU Slave Attached
               Broadcasts Disabled, Slave Address Source=Unit ID

Configuration Password: USERUSER

You must (S)ave the configuration to make it active.

Returning to Main Menu in 2 Seconds, Hit Enter to refresh._
```

Upgrading the TwidoPort Firmware

Foreword

Note:

1. Obtain a newer version of the TwidoPort firmware before attempting to upgrade the firmware with these instructions.
2. Stop the process before upgrading the firmware.
3. Modbus communication will not be available during the firmware upgrade procedure.

Upgrading the Firmware

To upgrade the current TwidoPort's firmware with the latest firmware release you have obtained, follow this procedure:

Step	Action	Comment
1	Start a Telnet session.	Use the instructions above to open the Telnet main menu (See <i>Telnet Main Menu, p. 288</i>).
2	Select (type) F to initiate the firmware upgrade.	Five seconds after selecting F (firmware upgrade), TwidoPort resets and the Telnet connection is lost.
3	At the command line, type: ftp and TwidoPort's IP address.	For example: ftp 85.16.44.113
4	Enter: ftptwd	At the login name prompt.
5	Enter: cd fw	This takes the user to the fw directory.
6	Enter: put App.out . (See notes 1 and 2.)	A message indicates that the ftp was successful. (See note 3.)
Note 1	File names are case-sensitive.	
Note 2	Make sure App.out is in the current working directory of the ftp client.	
Note 3	A message indicates that TwidoPort will automatically reboot 5 seconds after a successful ftp.	

Firmware Upgrade In Progress

The following figure shows a typical **Firmware Upgrade In-Progress** screen:

```
Telemechanique 499 TWD 01 100 Configuration and Diagnostics
-----
FIRMWARE UPGRADE IN-PROGRESS...
Module will reboot in 5 Seconds.
After Reboot, Connect via FTP to download new Firmware.

FTP Instructions:
1> Connect via FTP: ftp 192.168.2.160
2> Change to /fw directory: ftp>cd fw
3> Download new fw: ftp>put App.out

After the FTP download is complete, the module will reboot automatically
.

Rebooting now. Goodbye.

Connection to host lost.
```

Kernel Mode

In the absence of valid firmware, TwidoPort goes into **Kernel** mode. If you attempt to use Telnet to connect to TwidoPort while it is in this mode, you will see:

```
Telemechanique 499 TWD 01 100
-----
Kernel Version 90.02d

Download valid Exec,App.out, to leave kernel mode.

To exit type 'quit' 'QUIT' or control D
```

Forget Your Password and/or IP Configuration?

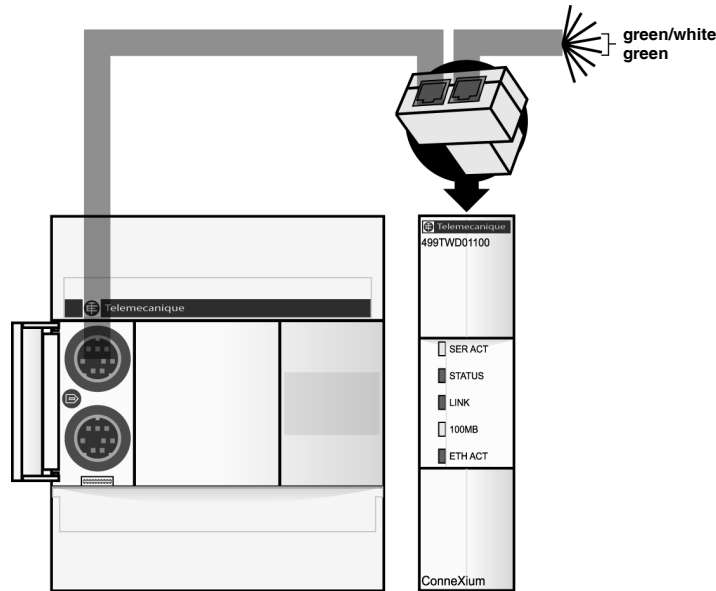
Connecting in Backup Mode

Use these instructions to connect to TwidoPort in backup mode.

Step	Action	Comment
1	Connect pin 3 to pin 6 (ground) of the serial connector.	Use the Schneider 170 XTS 04 100 RJ-45 T-connector. <i>(See the following illustration.)</i>
2	Connect via ftp to TwidoPort. <i>(See note.)</i>	TwidoPort uses the following default IP configuration: • IP address: 192.168.2.102 • Subnet mask: 255.255.0.0 • Gateway address: 192.168.2.102 • Frame type: Ethernet II
3	Get file fw/Conf.dat .	Obtain the IP configuration and password from the Conf.dat file.
4	Open the Conf.dat file in a text editor.	
Note	No password is required.	

FTP Connection

The following illustration shows how to connect to TwidoPort via ftp in backup mode:



11.3 Communication Features

At a Glance

Subject of this Section This section describes the communications features supported by the ConneXium TwidoPort Ethernet gateway.

What's in this Section? This section contains the following topics:

Topic	Page
Ethernet Features	301
Modbus/TCP Communications Protocol	302
Locally Supported Modbus Function Codes	303

Ethernet Features

Introduction

The ConneXium TwidoPort adds Ethernet connectivity to Telemecanique's Twido product line. It is the gateway between a single Twido Modbus/RTU (RS-485) device and the physical layer of Modbus/TCP networks in slave mode. TwidoPort does not require a separate power supply because it gets power from the Twido controller through its serial port. This gateway module supports slave mode only.

Ethernet Features

TwidoPort supports the following Ethernet features:

- **Auto-negotiation**

TwidoPort supports 10/100TX auto-negotiation. It communicates only in half-duplex mode.

- **Auto-MDI/MDI-X**

TwidoPort supports auto-switching of transmit and receive wire pairs to establish communications with the end device (auto-MDI/MDI-X). TwidoPort, therefore, transparently interconnects infrastructure or end devices with either straight-through or crossover cables.

Modbus/TCP Communications Protocol

About Modbus

The Modbus protocol is a master/slave protocol that allows one master to request responses from slaves or to take action based on their requests. The master can address individual slaves or can initiate a broadcast message to all slaves. Slaves return a message (response) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

About Modbus/ TCP Communications

TwidoPort supports up to 8 simultaneous Modbus/TCP connections. Attempting to use more than 8 connections results in a degradation of performance because TwidoPort closes the connection with the longest idle time to accept a new connection request.

Theory of Operations

Modbus/TCP clients can communicate with Twido through TwidoPort, a bridge between Twido devices (Modbus/RTU over RS-485 serial link) and Modbus/TCP over Ethernet networks.

Note: When implementing TwidoPort on a network, the system design requirements must account for the inherent limited bandwidth of serial connections. Expect a peak performance of approximately 40 Modbus transactions per second. Requesting multiple registers in a single request is more efficient than placing a separate request for each register.

You cannot initiate read or write requests from the Twido controller through TwidoPort.

Locally Supported Modbus Function Codes

List Function Codes

TwidoPort answers the following locally supported Modbus function codes only when the unit ID is set to 254. (Locally supported function codes are those answered directly by TwidoPort and not by the Twido controller.)

Modbus Function Code	Subfunction Code	OPCODE	Description
8	0	N/A	return query data
8	10	N/A	clear counters
8	11	N/A	return bus message count
8	12	N/A	return bus comm. error count
8	13	N/A	return bus exception error count
8	14	N/A	return slave message count
8	15	N/A	return slave no response count
8	21	3	get Ethernet statistics
8	21	4	clear Ethernet statistics
43	14	N/A	read device ID (see <i>note 1.</i>)
Note 1	TwidoPort supports only the basic object IDs of the read device identification function code with both stream and individual access.		

Note: See the Modbus specification at www.modbus.org for details on message formats and access classes.

Operator Display Operation

12

At a Glance

Subject of this Chapter

This chapter provides details for using the optional Twido Operator Display.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Operator Display	306
Controller Identification and State Information	309
System Objects and Variables	311
Serial Port Settings	317
Time of Day Clock	318
Real-Time Correction Factor	319

Operator Display

Introduction

The Operator Display is a Twido option for displaying and controlling application data and some controller functions such as operating state and the Real-Time Clock (RTC). This option is available as a cartridge (TWDXCPODC) for the Compact controllers or as an expansion module (TWDXCPODM) for the Modular controllers. The Operator Display has two operating modes:

- Display Mode: only displays data.
- Edit mode: allows you to change data.

Note: The operator display is updated at a specific interval of the controller scan cycle. This can cause confusion in interpreting the display of dedicated outputs for %PLS or %PWM pulses. At the time these outputs are sampled, their value will always be zero, and this value will be displayed.

Displays and Functions

The Operator Display provides the following separate displays with the associated functions you can perform for each display.

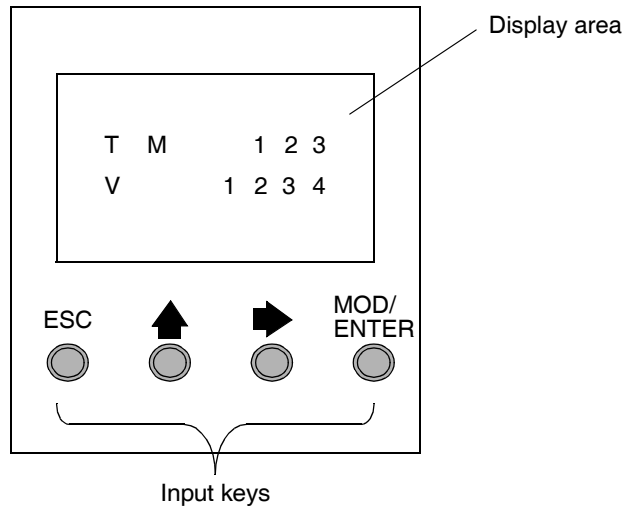
- Controller Identification and State Information: Operations Display
Display firmware revision and the controller state. Change the controller state with the Run, Initial, and Stop commands.
- System Objects and Variables: Data Display
Select application data by the address: %I, %Q, and all other software objects on the base controller. Monitor and change the value of a selected software data object.
- Serial Port Settings: Communication Display
Display and modify communication port settings.
- Time of Day Clock: Time/Date Display
Display and configure the current date and time (if the RTC is installed).
- Real Time Correction: RTC Factor
Display and modify the RTC Correction value for the optional RTC.

Note:

1. The TWDLCA•40DRF series of compact controllers have RTC onboard.
2. On all other controllers, time of day clock and real-time correction are only available if the Real-Time Clock (RTC) option cartridge (TWDXCPRTC) is installed.

Illustration

The following illustration shows a view of the Operator Display, which consists of a display area (here in Normal mode) and four push-button input keys.

**Display area**

The Operator Display provides an LCD display capable of displaying two lines of characters:

- The first line of the display has three 13-segment characters and four 7-segment characters.
- The second line has one 13-segment character, one 3-segment character (for a plus/minus sign), and five 7-segment characters.

Note: If in Normal mode, the first line indicates an object name and the second line displays its value. If in Data mode, the first line displays %SW68 value and the second line displays %SW69 value.

Input keys

The functions of the four input push-buttons depend on the Operator Display mode.

Key	In Display Mode	In Edit Mode
ESC		Discard changes and return to previous display.
▲		Go to the next value of an object being edited.
▶	Advance to next display.	Go to the next object type to edit.
MOD/ ENTER	Go to edit mode.	Accept changes and return to previous display.

Selecting and Navigating the Displays

The initial display or screen of the Operator Display shows the controller identification and state information. Press the ▶ push-button to sequence through each of the displays. The screens for the Time of Day Clock or the Real-Time Correction Factor are not displayed if the optional RTC cartridge (TWDXCPRTC) is not detected on the controller.

As a shortcut, press the ESC key to return to the initial display screen. For most screens, pressing the ESC key will return to the Controller Identification and State Information screen. Only when editing System Objects and Variables that are not the initial entry (%I0.0.0), will pressing ESC take you to the first or initial system object entry.

To modify an object value, instead of pressing the ▶ push-button to go to the first value digit, press the MOD/ENTER key again.

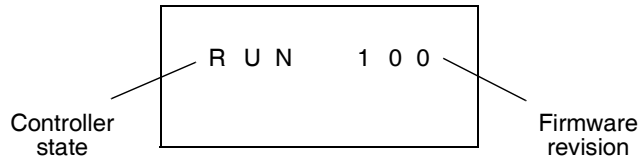
Controller Identification and State Information

Introduction

The initial display or screen of the Twido optional Operator Display shows the Controller Identification and State Information.

Example

The firmware revision is displayed in the upper-right corner of the display area, and the controller state is displayed in the upper-left corner of the display area, as seen in the following:



Controller States Controller states include any of the following:

- **NCF: Not Configured**

The controller is in the NCF state until an application is loaded. No other state is allowed until an application program is loaded. You can test the I/O by modifying system bit S8 (see *System Bits (%S)*, p. 596).

- **STP: Stopped**

Once an application is present in the controller, the state changes to the STP or Stopped state. In this state, the application is not running. Inputs are updated and data values are held at their last value. Outputs are not updated in this state.

- **INI: Initial**

You can choose to change the controller to the INI or initial state only from the STP state. The application is not running. The controller's inputs are updated and data values are set to their initial state. No outputs are updated from this state.

- **RUN: Running**

When in the RUN or running state the application is running. The controller's inputs are updated and data values are set according to the application. This is the only state where the outputs are updated.

- **HLT: Halted (User Application Error)**

If the controller has entered an ERR or error state, the application is halted. Inputs are updated and data values are held at their last value. From this state, outputs are not updated. In this mode, the error code is displayed in the lower-right portion of the Operator Display as an unsigned decimal value.

- **NEX: Not Executable (not executable)**

An online modification was made to user logic. Consequences: The application is no longer executable. It will not go back into this state until all causes for the Non-Executable state have been resolved.

**Displaying and
Changing
Controller States**

Using the Operator Display, you can change to the INI state from the STP state, or from STP to RUN, or from RUN to STP. Do the following to change the state of the controller:

Step	Action
1	Press the  key until the Operations Display is shown (or press ESC). The current controller state is displayed in the upper-left corner of the display area.
2	Press the MOD/ENTER key to enter edit mode.
3	Press the  key to select a controller state.
4	Press the MOD/ENTER key to accept the modified value, or press the ESC key to discard any modifications made while in edit mode.

System Objects and Variables

Introduction

The optional Operator Display provides these features for monitoring and adjusting application data:

- Select application data by address (such as %I or %Q).
- Monitor the value of a selected software object/variable.
- Change the value of the currently displayed data object (including forcing inputs and outputs).

System Objects and Variables

The following table lists the system objects and variables, in the order accessed, that can be displayed and modified by the Operator Display.

Object	Variable/Attribute	Description	Access
Input	%Ix.y.z	Value	Read/Force
Output	%Qx.y.z	Value	Read/Write/Force
Timer	%TMX.V %TMX.P %TMX.Q	Current Value Preset value Done	Read/Write Read/Write Read
Counter	%Cx.V %Cx.P %Cx.D %Cx.E %Cx.F	Current Value Preset value Done Empty Full	Read/Write Read/Write Read Read Read
Memory Bit	%Mx	Value	Read/Write
Word Memory	%MWx	Value	Read/Write
Constant Word	%KWx	Value	Read
System Bit	%Sx	Value	Read/Write
System Word	%SWx	Value	Read/Write
Analog Input	%IWx.y.z	Value	Read
Analog output	%QWx.y.z	Value	Read/Write
Fast Counter	%FCx.V %FCx.VD* %FCx.P %FCx.PD* %FCx.D	Current Value Current Value Preset value Preset value Done	Read Read Read/Write Read/Write Read

Object	Variable/Attribute	Description	Access
Very Fast Counter	%VFCx.V %VFCx.VD* %VFCx.P %VFCx.PD* %VFCx.U %VFCx.C %VFCx.CD* %VFCx.S0 %VFCx.S0D* %VFCx.S1 %VFCx.S1D* %VFCx.F %VFCx.T %VFCx.R %VFCx.S	Current Value Current Value Preset value Preset value Count Direction Catch Value Catch Value Threshold 0 Value Threshold 0 Value Threshold Value1 Threshold Value1 Overflow Timebase Reflex Output Enable Reflex Input Enable	Read Read Read/Write Read/Write Read Read Read Read/Write Read/Write Read/Write Read/Write Read Read/Write Read/Write Read/Write
Input Network Word	%INWx.z	Value	Read
Output Network Word	%QNWx.z	Value	Read/Write
Grafcet	%Xx	Step Bit	Read
Pulse Generator	%PLS.N %PLS.ND* %PLS.P %PLS.D %PLS.Q	Number of Pulses Number of Pulses Preset value Done Current Output	Read/Write Read/Write Read/Write Read Read
Pulse Width Modulator	%PWM.R %PWM.P	Ratio Preset value	Read/Write Read/Write
Drum Controller	%DRx.S %DRx.F	Current Step Number Full	Read Read
Step counter	%SCx.n	Step Counter bit	Read/Write
Register	%Rx.I %Rx.O %Rx.E %Rx.F	Input Output Empty Full	Read/Write Read/Write Read Read
Shift bit register	%SBR.x.yy	Register Bit	Read/Write
Message	%MSGx.D %MSGx.E	Done Error	Read Read
AS-Interface slave input	%IAx.y.z	Value	Read/Force
AS-Interface analog slave input	%IWAx.y.z	Value	Read
AS-Interface slave output	%QAx.y.z	Value	Read/Write/Force
AS-Interface analog slave output	%QWAx.y.z	Value	Read/Write
CANopen slave PDO input	%IWCx.y.z	Single-word value	Read
CANopen slave PDO output	%QWCx.y.z	Single-word value	Read/Write

Notes:

1. (*) means a 32-bit double word variable. The double word option is available on all controllers with the exception of the Twido TWDLC•A10DRF controllers.
2. Variables will not be displayed if they are not used in an application since Twido uses dynamic memory allocation.
3. If the value of %MW is greater than +32767 or less than -32768, the operator display will continue to blink.
4. If the value of %SW is greater than 65535, the operator display continues to blink, except for %SW0 and %SW11. If a value is entered that is more than the limit, the value will return to the configured value.
5. If a value is entered for %PLS.P that is more than the limit, the value written is the saturation value.

Displaying and Modifying Objects and Variables

Each type of system object is accessed by starting with the Input Object (%I), sequencing through to the Message object (%MSG), and finally looping back to the Input Object (%I).

To display a system object:

Step	Action
1	Press the  key until the Data Display screen is shown. The Input object ("I") will be displayed in the upper left corner of the display area. The letter "I" (or the name of the object previously viewed as data) is not blinking.
2	Press the MOD/ENTER key to enter edit mode. The Input Object "I" character (or previous object name viewed as data) begins blinking.
3	Press the  key to step sequentially through the list of objects.
4	Press the  key to step sequentially through the field of an object type and press the  key to increment through the value of that field. You can use the  key and  key to navigate and modify all fields of the displayed object.
5	Repeat steps 3 and 4 until editing is complete.
6	Press the MOD/ENTER key to accept the modified values. Note: The object's name and address have to be validated before accepting any modifications. That is, they must exist in the configuration of the controller prior to using the operator display. Press ESC to discard any changes made in edit mode.

Data Values and Display Formats

In general, the data value for an object or variable is shown as a signed or unsigned integer in the lower-right of the display area. In addition, all fields suppress leading zeros for displayed values. The address of each object is displayed on the Operator Display in one of the following seven formats:

- I/O format
 - AS-Interface slaves I/O format
 - CANopen slaves I/O format
 - Function Block Format
 - Simple Format
 - Network I/O format
 - Step Counter Format
 - Shift bit register format
-

Input/Output Format

The input/output objects (%I, %Q, %IW and %QW) have three-part addresses (e.g.: %IX.Y.Z) and are displayed as follows:

- Object type and controller address in the upper-left
- Expansion address in the upper-center
- I/O channel in the upper-right

In the case of a simple input (%I) and output (%Q), the lower-left portion of the display will contain a character that is either "U" for unforced or "F" for a forced bit. The force value is displayed in the lower-right of the screen.

The output object %Q0.3.11 appears in the display area as follows:

Q	0	3	1	1
F				1

AS-Interface slaves I/O format

AS-Interface slave I/O objects (%IA, %QA, %IWA and %QWA) have four-part addresses (e.g.: %IAx.y.z) and are displayed as follows:

- The object type in the upper-left
- AS-Interface master address on the expansion bus in the upper-left center
- Address of the slave on the AS-Interface bus in the upper-right center
- Slave I/O channel in the upper-right.

In the case of a simple input (%IA) and output (%QA), the lower-left portion of the display will contain a character that is either "U" for unforced or "F" for a forced bit. The force value is displayed in the lower-right of the screen.

The output object %QA1.3A.2 appears in the display area as follows:

QA	1	3A		2
F				1

**CANopen slaves
I/O format**

CANopen slave PDO I/O objects (%IWC and %QWC) have four-part addresses (e.g.: %IWCx.y.z) and are displayed as follows:

- The object type in the upper-left
- CANopen master address on the expansion bus in the upper-left center
- Address of the slave on the CANopen bus in the upper-right center
- Slave PDO I/O channel in the upper-right.
- Signed value for the object in the lower portion

In the following example, the PDO output object %QWC1.3.2 contains the signed value +24680:

QWC 1	3	2	
+	2 4 6 8 0		

**Function Block
Format**

The function blocks (%TM, %C, %FC, %VFC, %PLS, %PWM, %DR, %R, and %MSGj) have two-part addresses containing an object number and a variable or attribute name. They are displayed as follows:

- Function block name in the upper-left
- Function block number (or instance) in the upper-right
- The variable or attribute in the lower-left
- Value for the attribute in the lower-right

In the following example, the current value for timer number 123 is set to 1,234.

T	M	1 2 3	
V		1 2 3 4	

Simple Format

A simple format is used for objects %M, %MW, %KW, %MD, %KD, %MF, %KF, %S, %SW and %X as follows:

- Object number in the upper-right
- Signed value for the objects in the lower portion

In the following example, memory word number 67 contains the value +123.

M	W	6 7	
	+	1 2 3	

**Network Input/
Output Format**

The network input/output objects (%INW and %QNW) appear in the display area as follows:

- Object type in the upper-left
- Controller address in the upper-center
- Object number in the upper-right
- Signed value for the object in the lower portion

In the following example, the first input network word of the remote controller configured at remote address #2 is set to a value -4.

I	N	W	2	0
		-		4

**Step Counter
Format**

The step counter (%SC) format displays the object number and the step counter bit as follows:

- Object name and number in the upper-left
- Step counter bit in the upper right
- The value of the step counter bit in the lower portion of the display

In the following example, bit number 129 of step counter number 3 is set to 1.

S	C	3	1	2	9
					1

**Shift Bit Register
Format**

The shift bit register (%SBR) appears in the display area as follows:

- Object name and number in the upper-left
- Register bit number in the upper-right
- Register bit value in the lower-right

The following example shows the display of shift bit register number 4.

S	B	R	4	9
				1

Serial Port Settings

Introduction

The operator display allows you to display the protocol settings and change the addresses of all serial ports configured using TwidoSoft. The maximum number of serial ports is two. In the example below, the first port is configured as Modbus protocol with an address 123. The second serial port is configured as a remote link with an address of 4.

M	1	2	3
R			4

Displaying and Modifying Serial Port Settings

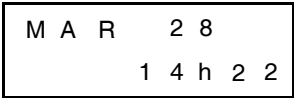
Twido controllers can support up to two serial ports. To display the serial port settings using the operator display:

Step	Action
1	Press the  key until the Communication Display is shown. The single letter of the protocol setting of the first serial port ("M", "R", or "A") will be displayed in the upper left corner of the operator display.
2	Press the MOD/ENTER key to enter the edit mode.
3	Press the  key until you are in the field that you wish to modify.
4	Press the  key to increment the value of that field.
5	Continue steps 3 and 4 until the address settings are complete.
6	Press the MOD/ENTER key to accept the modified values or ESC to discard any modifications made while in edit mode.
Note: The address is part of the configuration data on the Controller. Changing its value using the operator display means that you can no longer connect using TwidoSoft as equal. TwidoSoft will require that you do a download to become equal again.	

Time of Day Clock

Introduction

You can modify the date and time using the operator display if the RTC option cartridge (TWDXCPRTC) is installed on your Twido controller. The Month is displayed in the upper-left side of the HMI Display. Until a valid time has been entered, the month field will contain the value "RTC". The day of the month is displayed in the upper-right corner of the display. The time of day is in military format. The hours and minutes are shown in the lower-right corner of the display and are separated by the letter "h". The example below shows that the RTC is set to March 28, at 2:22 PM.



Note:

1. The TWDLCA•40DRF series of compact controllers have RTC onboard.
2. On all other controllers, time of day clock and real-time correction are only available if the Real-Time Clock (RTC) option cartridge (TWDXCPRTC) is installed.

Displaying and Modifying Time of Day Clock

To display and modify the Time of Day Clock:

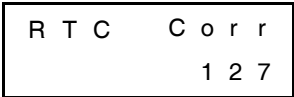
Step	Action
1	Press the  key until the Time/Date Display is shown. The month value ("JAN", "FEB") will be displayed in the upper-left corner of the display area. The value "RTC" will be displayed in the upper-left corner if no month has been initialized.
2	Press the MOD/ENTER key to enter the edit mode.
3	Press the  key until you are in the field that you wish to modify.
4	Press the  key increment the value of that field.
5	Continue steps 3 and 4 until the Time of Day value is complete.
6	Press the MOD/ENTER key to accept the modified values or ESC to discard any modifications made while in edit mode.

Real-Time Correction Factor

Introduction

You can display and modify the Real-Time Correction Factor using the operator display. Each Real-Time Clock (RTC) Option module has a RTC Correction Factor value that is used to correct for inaccuracies in the RTC module's crystal. The correction factor is an unsigned 3-digit integer from 0 to 127 and is displayed in the lower-right corner of the display.

The example below shows a correction factor of 127.



Displaying and Modifying RTC Correction

To display and modify the Real-Time Correction Factor:

Step	Action
1	Press the  key until the RTC Factor Display is shown. "RTC Corr" will be displayed in the upper line of the operator display.
2	Press the MOD/ENTER key to enter edit mode.
3	Press the  key until you are in the field that you wish to modify.
4	Press the  key to increment the value of that field.
5	Continue Steps 3 and 4 until the RTC correction value is complete.
6	Press the MOD/ENTER key to accept the modified values or ESC to discard any modifications made while in edit mode.

Description of Twido Languages



At a Glance

Subject of this Part

This part provides instructions for using the Ladder, List, and Grafcet programming languages to create control programs for Twido programmable controllers.

What's in this Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
13	Ladder Language	323
14	Instruction List Language	343
15	Grafcet	355

At a Glance

Subject of this Chapter

This chapter describes programming using Ladder Language.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Introduction to Ladder Diagrams	324
Programming Principles for Ladder Diagrams	326
Ladder Diagram Blocks	328
Ladder Language Graphic Elements	330
Special Ladder Instructions OPEN and SHORT	333
Programming Advice	334
Ladder/List Reversibility	337
Guidelines for Ladder/List Reversibility	338
Program Documentation	340

Introduction to Ladder Diagrams

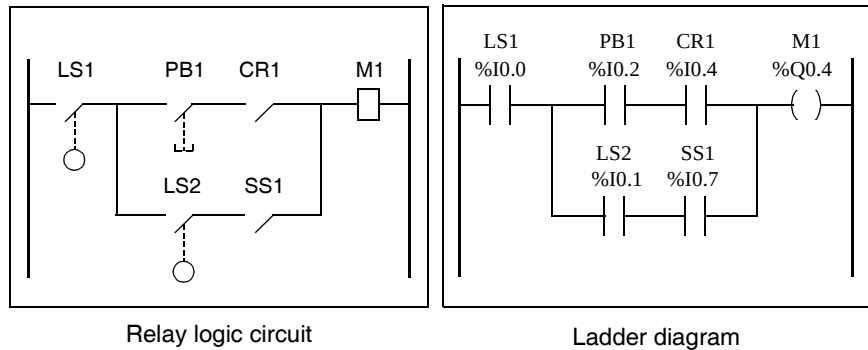
Introduction

Ladder diagrams are similar to relay logic diagrams that represent relay control circuits. The main differences between the two are the following features of Ladder programming that are not found in relay logic diagrams:

- All inputs are represented by contact symbols ($\neg$ —|—).
 - All outputs are represented by coil symbols ($\neg$ —|—).
 - Numerical operations are included in the graphical Ladder instruction set.
-

Ladder Equivalents to Relay Circuits

The following illustration shows a simplified wiring diagram of a relay logic circuit and the equivalent Ladder diagram.



Notice that in the above illustration, all inputs associated with a switching device in the relay logic diagram are shown as contacts in the Ladder diagram. The M1 output coil in the relay logic diagram is represented with an output coil symbol in the Ladder diagram. The address numbers appearing above each contact/coil symbol in the Ladder diagram are references to the locations of the external input/output connections to the controller.

Ladder Rungs

A program written in Ladder language is composed of rungs which are sets of graphical instructions drawn between two vertical potential bars. The rungs are executed sequentially by the controller.

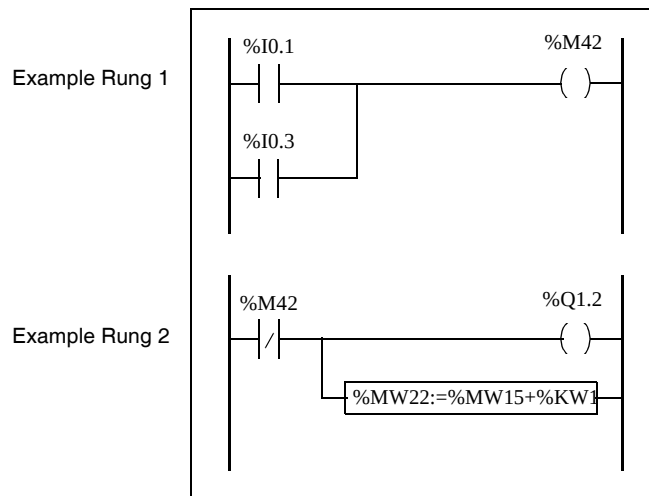
The set of graphical instructions represent the following functions:

- Inputs/outputs of the controller (push buttons, sensors, relays, pilot lights, etc.)
- Functions of the controller (timers, counters, etc.)
- Math and logic operations (addition, division, AND, XOR, etc.)
- Comparison operators and other numerical operations ($A < B$, $A = B$, shift, rotate, etc.)
- Internal variables in the controller (bits, words, etc.)

These graphical instructions are arranged with vertical and horizontal connections leading eventually to one or several outputs and/or actions. A rung cannot support more than one group of linked instructions.

Example of Ladder Rungs

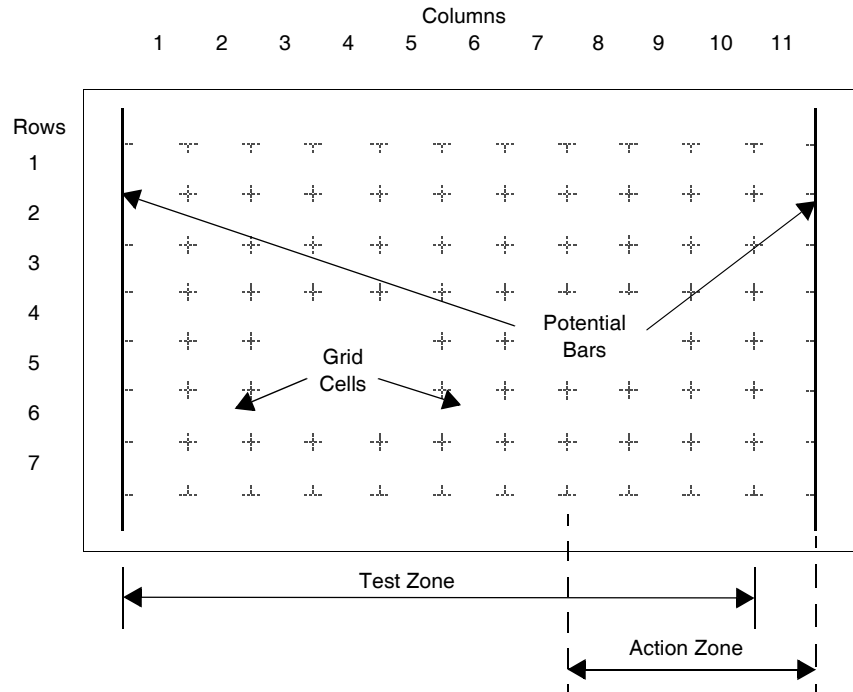
The following diagram is an example of a Ladder program composed of two rungs.



Programming Principles for Ladder Diagrams

Programming Grid

Each Ladder rung consists of a grid of seven rows by eleven columns that are organized into two zones as shown in the following illustration.



Grid Zones

The Ladder diagram programming grid is divided into two zones:

- **Test Zone**
Contains the conditions that are tested in order to perform actions. Consists of columns 1 - 10, and contains contacts, function blocks, and comparison blocks.
- **Action Zone**
Contains the output or operation that will be performed according to the results of the tests of the conditions in the Test Zone. Consists of columns 8 - 11, and contains coils and operation blocks.

**Entering
Instructions in
the Grid**

A Ladder rung provides a seven by eleven programming grid that starts in the first cell in the upper left-hand corner of the grid. Programming consists of entering instructions into the cells of the grid. Test instructions, comparisons, and functions are entered in cells in the test zone and are left-justified. The test logic provides continuity to the action zone where coils, numerical operations, and program flow control instructions are entered and are right-justified. The rung is solved or executed (tests made and outputs assigned) within the grid from top to bottom and from left to right.

Rung Headers

In addition to the rung, a rung header appears directly above the rung. Use the rung header to document the logical purpose of the rung. The rung header can contain the following information:

- Rung number
- Labels (%Li)
- Subroutine declarations (SRi:)
- Rung title
- Rung comments

For more details about using the rung header to document your programs, see *Program Documentation*, p. 340.

Ladder Diagram Blocks

Introduction

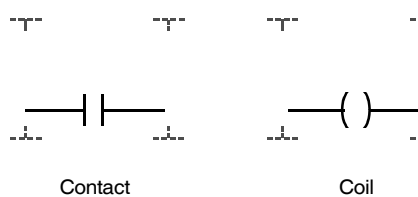
Ladder diagrams consist of blocks representing program flow and functions such as the following:

- Contacts
- Coils
- Program flow instructions
- Function blocks
- Comparison blocks
- Operate blocks

Contacts, Coils, and Program Flow

Contacts, coils, and program flow (jump and call) instructions occupy a single cell of the ladder programming grid. Function blocks, comparison blocks, and operate blocks occupy multiple cells.

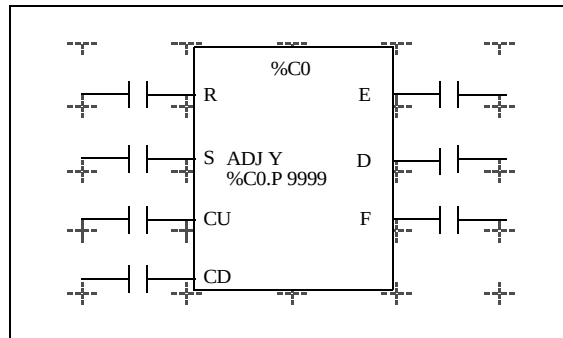
The following are examples of a contact and a coil.



Function Blocks

Function blocks are placed in the test zone of the programming grid. The block must appear in the first row; no ladder instructions or lines of continuity may appear above or below the function block. Ladder test instructions lead to the function block's input side, and test instructions and/or action instructions lead from the block's output side. Function blocks are vertically oriented and occupy two columns by four rows of the programming grid.

The following is an example of a counter function block.

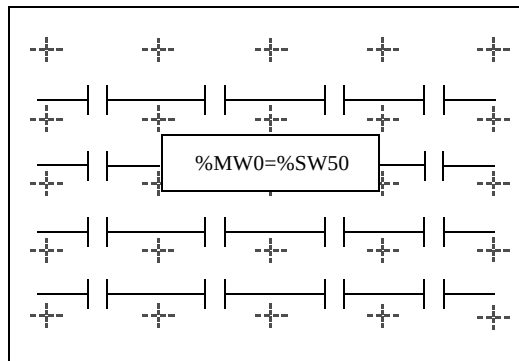


Comparison Blocks

Comparison blocks are placed in the test zone of the programming grid. The block may appear in any row or column in the test zone as long as the entire length of the instruction resides in the test zone.

Comparison blocks are horizontally oriented and occupy two columns by one row of the programming grid.

See the following example of a comparison block.

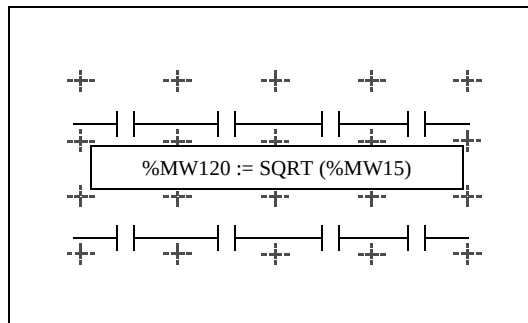


Operate blocks

Operate blocks are placed in the action zone of the programming grid. The block may appear in any row in the action zone. The instruction is right-justified; it appears on the right and ends in the last column.

Operate blocks are horizontally oriented and occupy four columns by one row of the programming grid.

The following is an example of an operate block.



Ladder Language Graphic Elements

Introduction

Instructions in Ladder diagrams consist of graphic elements.

Contacts

The contacts graphic elements are programmed in the test zone and take up one cell (one row high by one column wide).

Name	Graphic element	Instruction	Function
Normally open contact		LD	Passing contact when the controlling bit object is at state 1.
Normally closed contact		LDN	Passing contact when the controlling bit object is at state 0.
Contact for detecting a rising edge		LDR	Rising edge: detecting the change from 0 to 1 of the controlling bit object.
Contact for detecting a falling edge		LDF	Falling edge: detecting the change from 1 to 0 of the controlling bit object.

Link Elements

The graphic link elements are used to connect the test and action graphic elements.

Name	Graphic element	Function
Horizontal connection		Links in series the test and action graphic elements between the two potential bars.
Vertical connection		Links the test and action graphic elements in parallel.

Coils

The coil graphic elements are programmed in the action zone and take up one cell (one row high and one column wide).

Name	Graphic element	Instruction	Function
Direct coil		ST	The associated bit object takes the value of the test zone result.
Inverse coil		STN	The associated bit object takes the negated value of the test zone result.
Set coil		S	The associated bit object is set to 1 when the result of the test zone is 1.
Reset coil		R	The associated bit object is set to 0 when the result of the test zone is 1.
Jump or Subroutine call	->>%Li ->>%SRi	JMP SR	Connect to a labeled instruction, upstream or downstream.
Transition condition coil			Grafcet language. Used when the programming of the transition conditions associated with the transitions causes a changeover to the next step.
Return from a subroutine	<RET>	RET	Placed at the end of subroutines to return to the main program.
Stop program	<END>	END	Defines the end of the program.

Function blocks

The graphic elements of function blocks are programmed in the test zone and require four rows by two columns of cells (except for very fast counters which require five rows by two columns).

Name	Graphic element	Function
Timers, counters, registers, and so on.		Each of the function blocks uses inputs and outputs that enable links to the other graphic elements.. Note: Outputs of function blocks can not be connected to each other (vertical shorts).

Operate and Comparison Blocks

Comparison blocks are programmed in the test zone, and operate blocks are programmed in the action zone.

Name	Graphic element	Function
Comparison block		Compares two operands, the output changes to 1 when the result is checked. Size: one row by two columns
Operation block		Performs arithmetic and logic operations. Size: one row by four columns

Special Ladder Instructions OPEN and SHORT

Introduction The OPEN and SHORT instructions provide a convenient method for debugging and troubleshooting Ladder programs. These special instructions alter the logic of a rung by either shorting or opening the continuity of a rung as explained in the following table.

Instruction	Description	List Instruction
OPEN	Creates a break in the continuity of a ladder rung regardless of the results of the last logical operation.	AND 0
SHORT	Allows the continuity to pass through the rung regardless of the results of the last logical operation.	OR 1

In List programming, the OR and AND instructions are used to create the OPEN and SHORT instructions using immediate values of 0 and 1 respectively.

Examples The following are examples of using the OPEN and SHORT instructions.

```
graph LR
    subgraph Rung1 [Rung 1]
        direction TB
        B1_1[%I0.1] --- B1_2[%M3] --- B1_3[OPEN] --- C1[%Q0.1]
        B1_4[%Q1.5] --- C1
        B1_5[%I0.9] --- C1
    end
    subgraph Rung2 [Rung 2]
        direction TB
        S1[SHORT] --- C2[%Q1.6]
        B2_1[%I0.9] --- C2
    end
```

```
LD      %I0.1
OR      %Q1.5
ANDN    %M3
AND     0
ST      %Q0.1
LD      %I0.9
OR      1
ST      %Q1.6
```

Programming Advice

Handling Program Jumps

Use program jumps with caution to avoid long loops that can increase scan time. Avoid jumps to instructions that are located upstream. (An upstream instruction line appears before a jump in a program. A downstream instruction line appears after a jump in a program.).

Programming of Outputs

Output bits, like internal bits, should only be modified once in the program. In the case of output bits, only the last value scanned is taken into account when the outputs are updated.

Using Directly- Wired Emergency Stop Sensors

Sensors used directly for emergency stops must not be processed by the controller. They must be connected directly to the corresponding outputs.

Handling Power Returns

Make power returns conditional on a manual operation. An automatic restart of the installation could cause unexpected operation of equipment (use system bits %S0, %S1 and %S9).

Time and Schedule Block Management

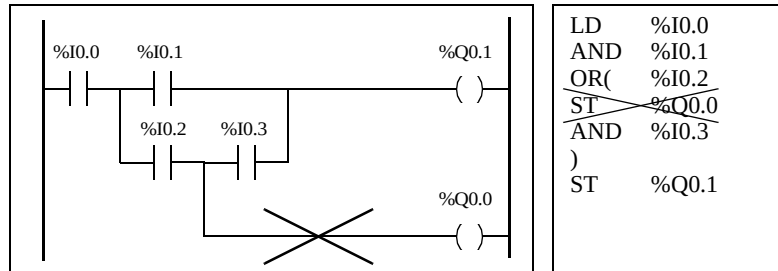
The state of system bit %S51, which indicates any RTC faults, should be checked.

Syntax and Error Checking

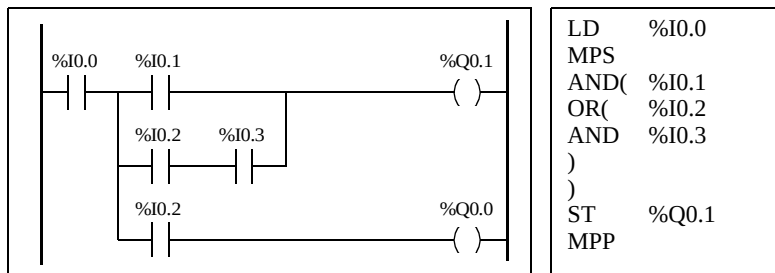
When a program is entered, TwidoSoft checks the syntax of the instructions, the operands, and their association.

Additional Notes on Using Parentheses

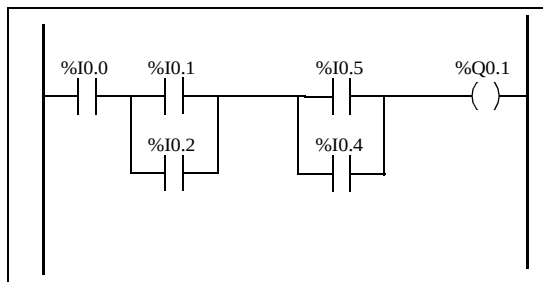
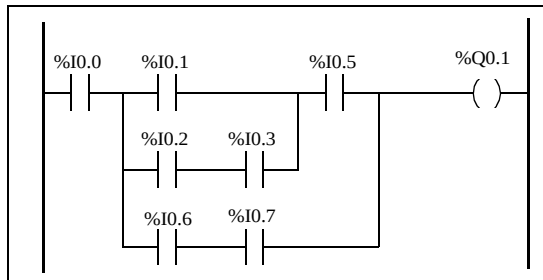
Assignment operations should not be placed within parentheses:



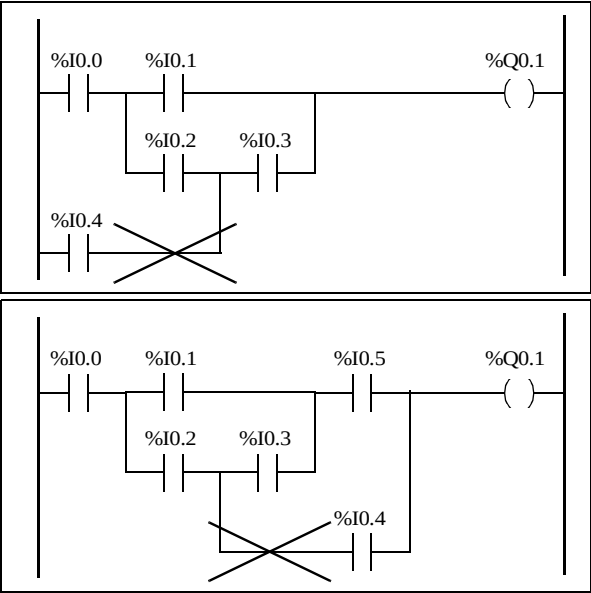
In order to perform the same function, the following equations must be programmed:



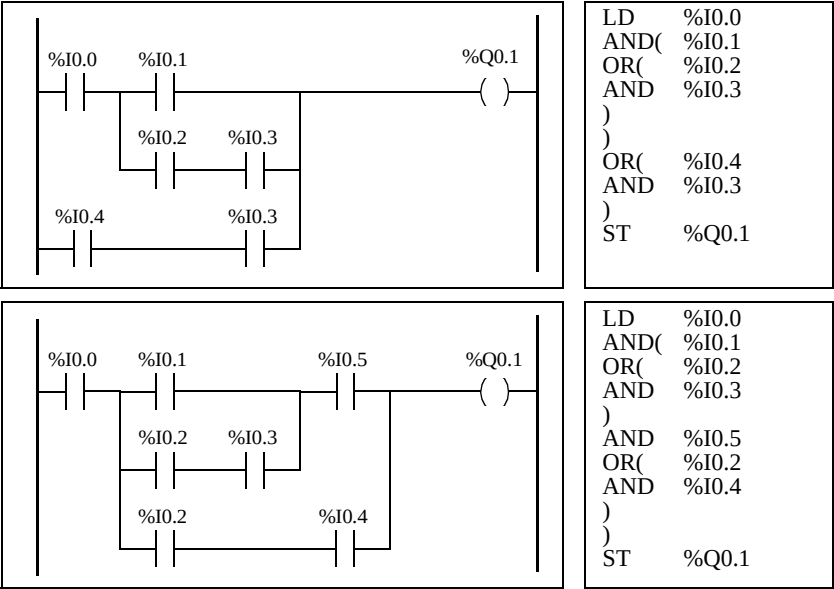
If several contacts are parallelized, they must be nested within each other or completely separate:



The following schematics cannot be programmed:



In order to execute schematics equivalent to those, they must be modified as follows:



Ladder/List Reversibility

Introduction

Program reversibility is a feature of the TwidoSoft programming software that provides conversion of application programs from Ladder to List and from List back to Ladder.

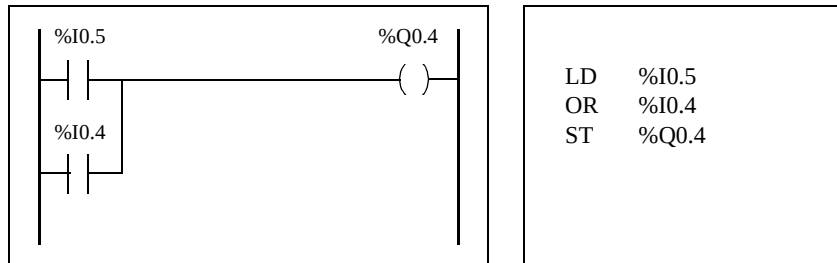
Use TwidoSoft to set the default display of programs: either List or Ladder format (by setting user preferences). TwidoSoft can also be used to toggle List and Ladder views.

Understanding Reversibility

A key to understanding the program reversibility feature is examining the relationship of a Ladder rung and the associated instruction List sequence:

- **Ladder rung:** A collection of Ladder instructions that constitute a logical expression.
- **List sequence:** A collection of List programming instructions that correspond to the Ladder instructions and represents the same logical expression.

The following illustration displays a common Ladder rung and its equivalent program logic expressed as a sequence of List instructions.



An application program is stored internally as List instructions, regardless if the program is written in Ladder language or List language. TwidoSoft takes advantage of the program structure similarities between the two languages and uses this internal List image of the program to display it in the List and Ladder viewers and editors as either a List program (its basic form), or graphically as a Ladder diagram, depending upon the selected user preference.

Ensuring Reversibility

Programs created in Ladder can always be reversed to List. However, some List logic may not reverse to Ladder. To ensure reversibility from List to Ladder, it is important to follow the set of List programming guidelines in *Guidelines for Ladder/List Reversibility*, p. 338.

Guidelines for Ladder/List Reversibility

Instructions Required for Reversibility

The structure of a reversible function block in List language requires the use of the following instructions:

- **BLK** marks the block start, and defines the beginning of the rung and the start of the input portion to the block.
- **OUT_BLK** marks the beginning of the output portion of the block.
- **END_BLK** marks the end of the block and the rung.

The use of the reversible function block instructions are not mandatory for a properly functioning List program. For some instructions it is possible to program in List which is not reversible. For a description of non-reversible List programming of standard function blocks, see *Standard function blocks programming principles, p. 388*.

Non-Equivalent Instructions to Avoid

Avoid the use of certain List instructions, or certain combinations of instructions and operands, which have no equivalents in Ladder diagrams. For example, the N instruction (inverses the value in the Boolean accumulator) has no equivalent Ladder instruction.

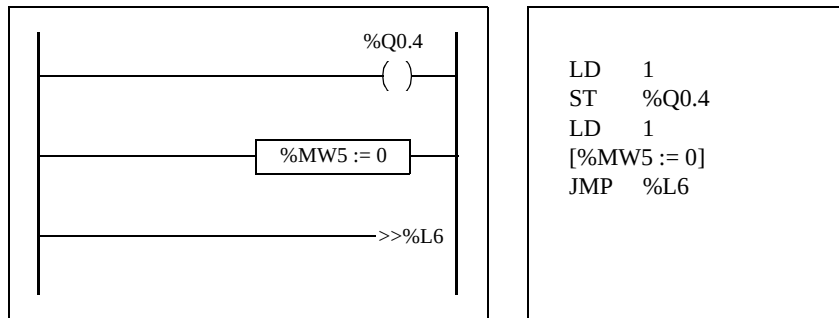
The following table identifies all List programming instructions that will not reverse to Ladder.

List Instruction	Operand	Description
JMPCN	%Li	Jump Conditional Not
N	none	Negation (Not)
ENDCN	none	End Conditional Not

Unconditional Rungs

Programming unconditional rungs also requires following List programming guidelines to ensure List-to-Ladder reversibility. Unconditional rungs do not have tests or conditions. The outputs or action instructions are always energized or executed.

The following diagram provides examples of unconditional rungs and the equivalent List sequence.



Notice that each of the above unconditional List sequences begin with a load instruction followed by a one, except for the JMP instruction. This combination sets the Boolean accumulator value to one, and therefore sets the coil (store instruction) to one and sets `%MW5` to zero on every scan of the program. The exception is the unconditional jump List instruction (JMP `%L6`) which is executed regardless of the value of the accumulator and does not need the accumulator set to one.

Ladder List Rungs

If a List program is reversed that is not completely reversible, the reversible portions are displayed in the Ladder view and the irreversible portions are displayed in Ladder List Rungs.

A Ladder List Rung functions just like a small List editor, allowing the user to view and modify the irreversible parts of a Ladder program.

Program Documentation

Documenting Your Program

You can document your program by entering comments using the List and Ladder editors:

- Use the List Editor to document your program with List Line Comments. These comments may appear on the same line as programming instructions, or they may appear on lines of their own.
- Use the Ladder Editor to document your program using rung headers. These are found directly above the rung.

The TwidoSoft programming software uses these comments for reversibility. When reversing a program from List to Ladder, TwidoSoft uses some of the List comments to construct a rung header. For this, the comments inserted between List sequences are used for rung headers.

Example of List Line Comments

The following is an example of a List program with List Line Comments.

```
---- ( * THIS IS THE TITLE OF THE HEADER FOR RUNG 0 * )
---- ( * THIS IS THE FIRST HEADER COMMENT FOR RUNG 0 * )
---- ( * THIS IS THE SECOND HEADER COMMENT FOR RUNG 0 * )
0 LD %I0.0 ( * THIS IS A LINE COMMENT * )
1 OR %I0.1 ( * A LINE COMMENT IS IGNORED WHEN REVERSING TO LADDER * )
2 ANDM %M10
3 ST M101
---- ( * THIS IS THE HEADER FOR RUNG 1 * )
---- ( * THIS RUNG CONTAINS A LABEL * )
---- ( * THIS IS THE SECOND HEADER COMMENT FOR RUNG 1 * )
---- ( * THIS IS THE THIRD HEADER COMMENT FOR RUNG 1 * )
---- ( * THIS IS THE FOURTH HEADER COMMENT FOR RUNG 1 * )
4 %L5:
5 LD %M101
6 [ %MW20 := %KW2 * 16 ]
---- ( * THIS RUNG ONLY CONTAINS A HEADER TITLE * )
7 LD %Q0.5
8 OR %I0.3
9 ORR I0.13
10 ST %Q0.5
```

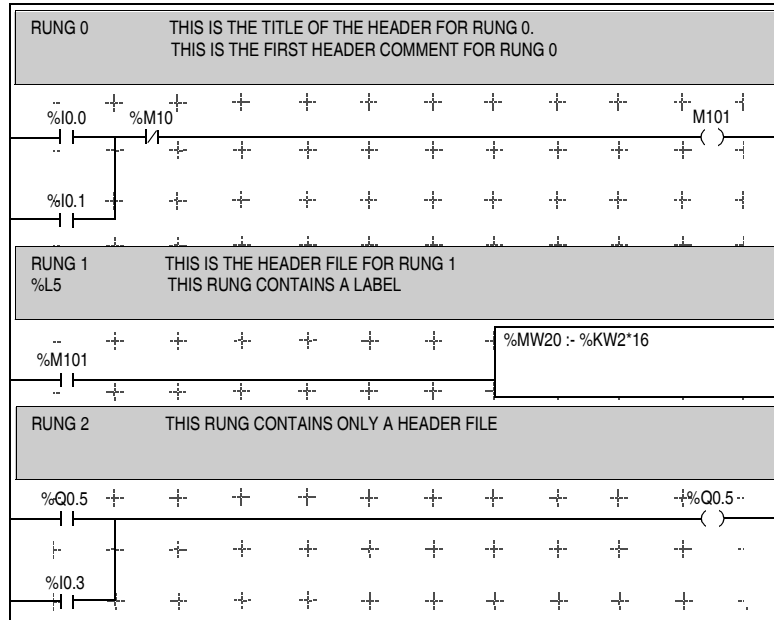
Reversing List Comments to Ladder

When List instructions are reversed to a Ladder diagram, List Line Comments are displayed in the Ladder Editor according to the following rules:

- The first comment that is on a line by itself is assigned as the rung header.
 - Any comments found after the first become the body of the rung.
 - Once the body lines of the header are occupied, then the rest of the line comments between List sequences are ignored, as are any comments that are found on list lines that also contain list instructions.
-

Example of Rung Header Comments

The following is an example of a Ladder program with rung header comments.



Reversing Ladder Comments to List

When a Ladder diagram is reversed to List instructions, rung header comments are displayed in the List Editor according to the following rules:

- Any rung header comments are inserted between the associated List sequences.
- Any labels (%Li:) or subroutine declarations (SRI:) are placed on the next line following the header and immediately prior to the List sequence.
- If the List was reversed to Ladder, any comments that were ignored will reappear in the List Editor.

Instruction List Language

14

At a Glance

Subject of this Chapter

This chapter describes programming using Instruction List Language.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Overview of List Programs	344
Operation of List Instructions	346
List Language Instructions	347
Using Parentheses	350
Stack Instructions (MPS, MRD, MPP)	352

Overview of List Programs

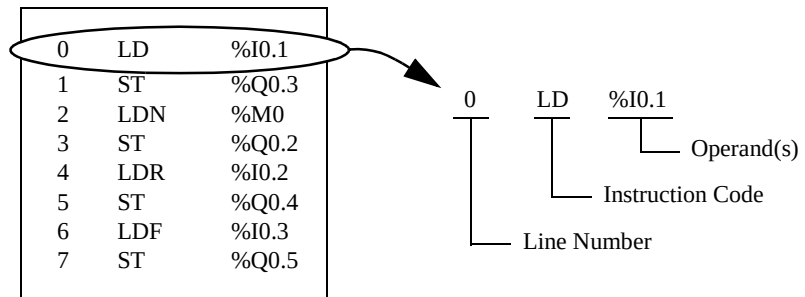
Introduction

A program written in List language consists of a series of instructions executed sequentially by the controller. Each List instruction is represented by a single program line and consists of three components:

- Line number
 - Instruction code
 - Operand(s)
-

Example of a List Program

The following is an example of a List program.



Line Number

Line numbers are generated automatically when you enter an instruction. Blank lines and Comment lines do not have line numbers.

Instruction Code

The instruction code is a symbol for an operator that identifies the operation to be performed using the operand(s). Typical operators specify Boolean and numerical operations.

For example, in the sample program above, LD is the abbreviation for the instruction code for a LOAD instruction. The LOAD instruction places (loads) the value of the operand %I0.1 into an internal register called the accumulator.

There are basically two types of instructions:

- Test instructions
These setup or test for the necessary conditions to perform an action. For example, LOAD (LD) and AND.
 - Action instructions
These perform actions as a result of setup conditions. For example, assignment instructions such as STORE (ST) and RESET (R).
-

Operand

An operand is a number, address, or symbol representing a value that a program can manipulate in an instruction. For example, in the sample program above, the operand %I0.1 is an address assigned the value of an input to the controller. An instruction can have from zero to three operands depending on the type of instruction code.

Operands can represent the following:

- Controller inputs and outputs such as sensors, push buttons, and relays.
 - Predefined system functions such as timers and counters.
 - Arithmetic, logical, comparison, and numerical operations.
 - Controller internal variables such as bits and words.
-

Operation of List Instructions

Introduction

List instructions have only one explicit operand, the other operand is implied. The implied operand is the value in the Boolean accumulator. For example, in the instruction LD %I0.1, %I0.1 is the explicit operand. An implicit operand is stored in the accumulator and will be written over by value of %I0.1.

Operation

A List instruction performs a specified operation on the contents of the accumulator and the explicit operand, and replaces the contents of the accumulator with the result. For example, the operation AND %I1.2 performs a logical AND between the contents of the accumulator and the Input 1.2 and will replace the contents of the accumulator with this result.

All Boolean instructions, except for Load, Store, and Not, operate on two operands. The value of the two operands can be either True or False, and program execution of the instructions produces a single value: either True or False. Load instructions place the value of the operand in the accumulator, while Store instructions transfer the value in the accumulator to the operand. The Not instruction has no explicit operands and simply inverts the state of the accumulator.

Supported List Instructions

The following table shows a selection of instructions in List Instruction language:

Type of Instruction	Example	Function
Bit instruction	LD %M10	Reads internal bit %M10
Block instruction	IN %TM0	Starts the timer %TM0
Word instruction	[%MW10 := %MW50+100]	Addition operation
Program instruction	SR5	Calls subroutine #5
Grafcet instruction	-*-8	Step #8

List Language Instructions

Introduction

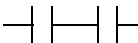
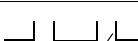
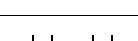
List language consists of the following types of instructions:

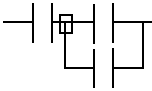
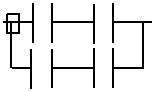
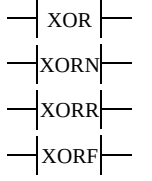
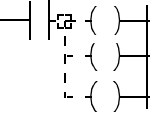
- Test Instructions
- Action instructions
- Function block instructions

This section identifies and describes the Twido instructions for List programming.

Test Instructions

The following table describes test instructions in List language.

Name	Equivalent graphic element	Function
LD		The Boolean result is the same as the status of the operand.
LDN		The Boolean result is the same as the reverse status of the operand.
LDR		The Boolean result changes to 1 on detection of the operand (rising edge) changing from 0 to 1.
LDF		The Boolean result changes to 1 on detection of the operand (falling edge) changing from 1 to 0.
AND		The Boolean result is equal to the AND logic between the Boolean result of the previous instruction and the status of the operand.
ANDN		The Boolean result is equal to the AND logic between the Boolean result of the previous instruction and the reverse status of the operand.
ANDR		The Boolean result is equal to the AND logic between the Boolean result of the previous instruction and the detection of the operand's rising edge (1 = rising edge).
ANDF		The Boolean result is equal to the AND logic between the Boolean result of the previous instruction and the detection of the operand's falling edge (1 = falling edge).
OR		The Boolean result is equal to the OR logic between the Boolean result of the previous instruction and the status of the operand.

Name	Equivalent graphic element	Function
AND(		Logic AND (8 parenthesis levels)
OR(		Logic OR (8 parenthesis levels)
XOR, XORN, XORR, XORF		Exclusive OR
MPS MRD MPP		Switching to the coils.
N	-	Negation (NOT)

Action instructions

The following table describes action instructions in List language.

Name	Equivalent graphic element	Function
ST	—()—	The associated operand takes the value of the test zone result.
STN	—(/)—	The associated operand takes the reverse value of the test zone result.
S	—(S)—	The associated operand is set to 1 when the result of the test zone is 1.
R	—(R)—	The associated operand is set to 0 when the result of the test zone is 1.
JMP	->>%Li	Connect unconditionally to a labeled sequence, upstream or downstream.
SRn	->>%SRi	Connection at the beginning of a subroutine.
RET	<RET>	Return from a subroutine.
END	<END>	End of program.
ENDC	<ENDC>	End of the conditioned program at a Boolean result of 1.
ENDCN	<ENDCN>	End of the conditioned program at a Boolean result of 0.

Function Block Instructions

The following table describes function blocks in List language.

Name	Equivalent graphic element	Function
Timers, counters, registers, and so on.		For each of the function blocks, there are instructions for controlling the block. A structured form is used to hardwire the block inputs and outputs directly. Note: Outputs of function blocks can not be connected to each other (vertical shorts).

Using Parentheses

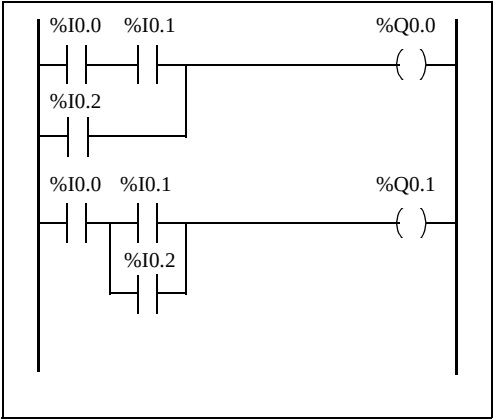
Introduction

In AND and OR logical instructions, parentheses are used to specify divergences in Ladder Editors. Parentheses are associated with instructions, as follows:

- Opening the parentheses is associated with the AND or OR instruction.
- Closing the parentheses is an instruction which is required for each open parentheses.

Example Using an AND Instruction

The following diagrams are examples of using a parentheses with an AND instruction: AND(...).

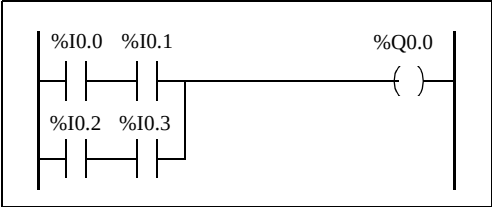


```
LD    %I0.0
AND   %I0.1
OR    %I0.2
ST    %Q0.0

LD    %I0.0
AND(  %I0.1
OR    %I0.2
)
ST    %Q0.1
```

Example Using an OR Instruction

The following diagrams are examples of using a parentheses with an OR instruction: OR(...).



```
LD    %I0.0
AND   %I0.1
OR(   %I0.2
AND   %I0.3
)
ST    %Q0.0
```

Modifiers

The following table lists modifiers that can be assigned to parentheses.

Modifier	Function	Example
N	Negation	AND(N or OR(N
F	Falling edge	AND(F or OR(F
R	Rising edge	AND(R or OR(R
[	Comparison	See <i>Comparison Instructions, p. 416</i>

Nesting Parenthesis

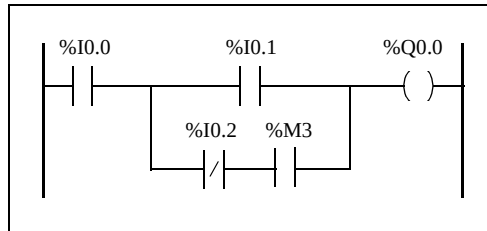
It is possible to nest up to eight levels of parentheses.

Observe the following rules when nesting parentheses:

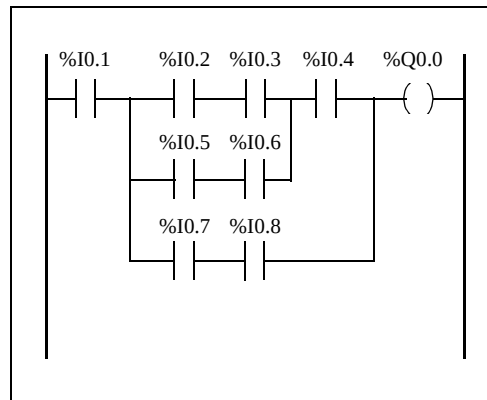
- Each open parentheses must have a corresponding closed parentheses.
- Labels (%Li:), subroutines (SRI:), jump instructions (JMP), and function block instructions must not be placed in expressions between parentheses.
- Store instructions ST, STN, S, and R must not be programmed between parentheses.
- Stack instructions MPS, MRD, and MPP cannot be used between parentheses.

Examples of Nesting Parentheses

The following diagrams provide examples of nesting parentheses.



```
LD      %I0.0
AND(    %I0.1
OR(N    %I0.2
AND     %M3
)
)
ST      %Q0.0
```



```
LD      %I0.1
AND(    %I0.2
AND     %I0.3
OR(     %I0.5
AND     %I0.6
)
AND     %I0.4
OR(     %I0.7
AND     %I0.8
)
)
ST      %Q0.0
```

Stack Instructions (MPS, MRD, MPP)

Introduction

The Stack instructions process routing to coils. The MPS, MRD, and MPP instructions use a temporary storage area called the stack which can store up to eight Boolean expressions.

Note: These instructions can not be used within an expression between parentheses.

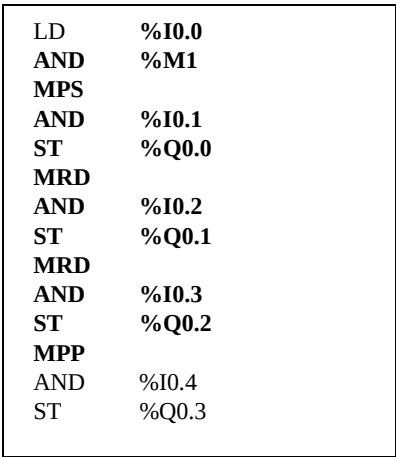
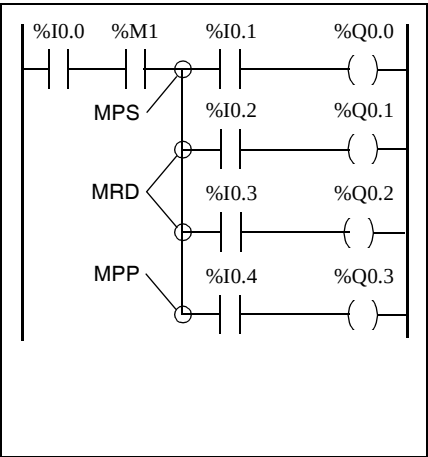
Operation of Stack Instructions

The following table describes the operation of the three stack instructions.

Instruction	Description	Function
MPS	Memory Push onto stack	Stores the result of the last logical instruction (contents of the accumulator) onto the top of stack (a push) and shifts the other values to the bottom of the stack.
MRD	Memory Read from stack	Reads the top of stack into the accumulator.
MPP	Memory Pop from stack	Copies the value at the top of stack into the accumulator (a pop) and shifts the other values towards the top of the stack.

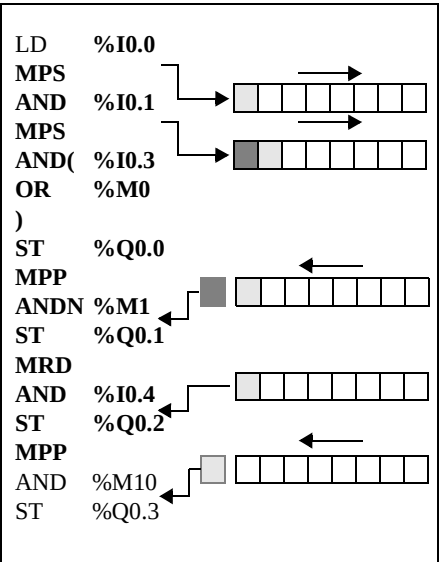
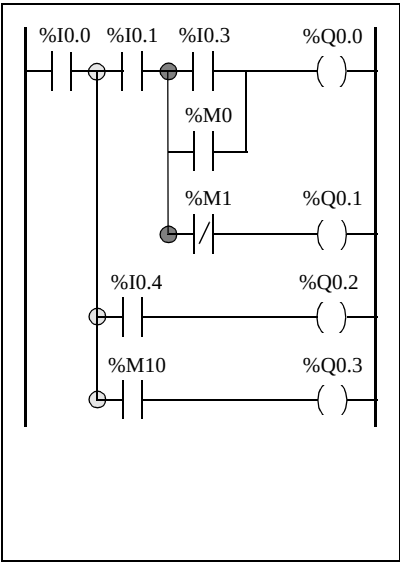
Examples of Stack Instructions

The following diagrams are examples of using stack instructions.



Examples of Stack Operation

The following diagrams display how stack instructions operate.



At a Glance

Subject of this Chapter

This chapter describes programming using Grafcet Language.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Description of Grafcet Instructions	356
Description of Grafcet Program Structure	359
Actions Associated with Grafcet Steps	362

Description of Grafcet Instructions

Introduction

Grafcet instructions in TwidoSoft offer a simple method of translating a control sequence (Grafcet chart).

The maximum number of Grafcet steps depend on the type of Twido controller. The number of steps active at any one time is limited only by the total number of steps. For the TWDLCAA10DRF and the TWDLCAA16DRF, steps 1 through 62 are available. Steps 0 and 63 are reserved for pre- and post-processing. For all other controllers, steps 1 through 95 are available.

Grafcet Instructions

The following table lists all instructions and objects required to program a Grafcet chart:

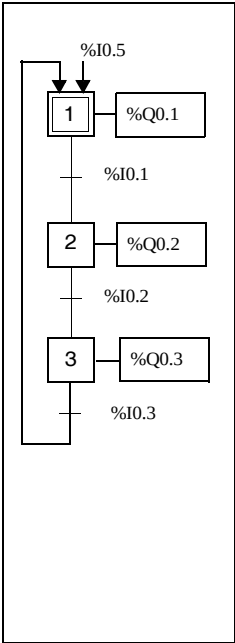
Graphic representation (1)	Transcription in TwidoSoft language	Function
Illustration:		
	<code>=* = i</code>	Start the initial step (2)
	<code># i</code>	Activate step i after deactivating the current step
	<code>-* - i</code>	Start step i and validate the associated transition (2)
	<code>#</code>	Deactivate the current step without activating any other steps
	<code>#Di</code>	Deactivate step i and the current step
	<code>=* = POST</code>	Start post-processing and end sequential processing
	<code>%Xi</code>	Bit associated with step i, can be tested and written (maximum number of steps depends on controller)
	<code>LD %Xi, LDN %Xi</code> <code>AND %Xi, ANDN %Xi,</code> <code>OR %Xi, ORN %Xi</code> <code>XOR %Xi, XORN %Xi</code>	Test the activity of step i
	<code>S %Xi</code>	Activate step i
	<code>R %Xi</code>	Deactivate step i

(1) The graphic representation is not taken into account.

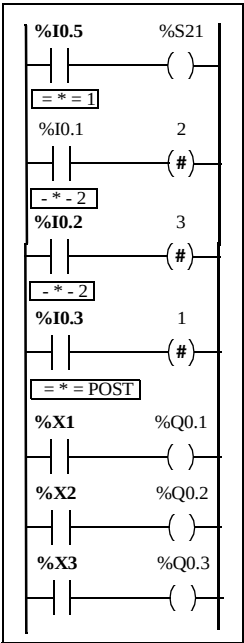
(2) The first step `=* = i` or `-* - i` written indicates the start of sequential processing and thus the end of preprocessing.

**Grafcet
Examples**

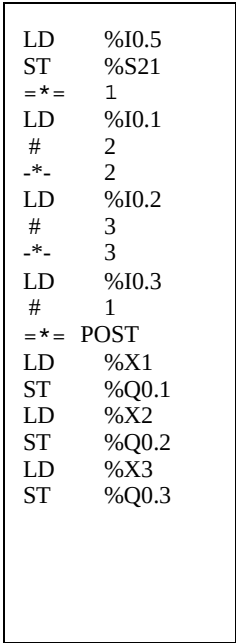
Linear sequence:



Not supported

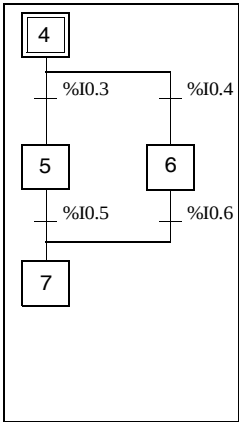


Twido Ladder
Language program

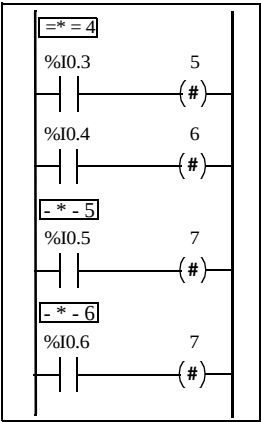


Twido Instruction
List program

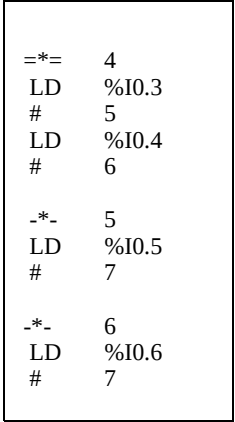
Alternative sequence:



Not supported

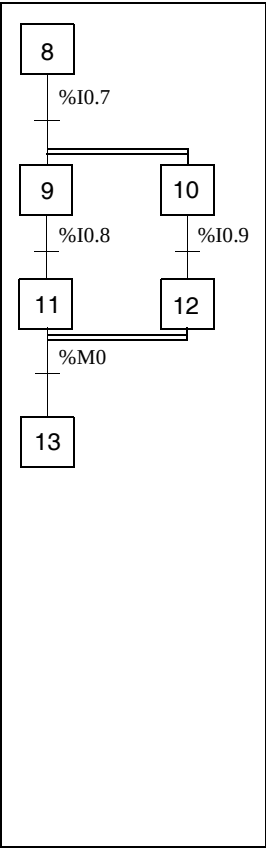


Twido Ladder
Language program

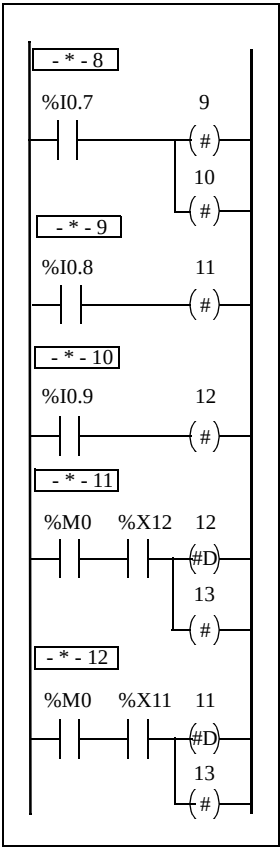


Twido Instruction
List program

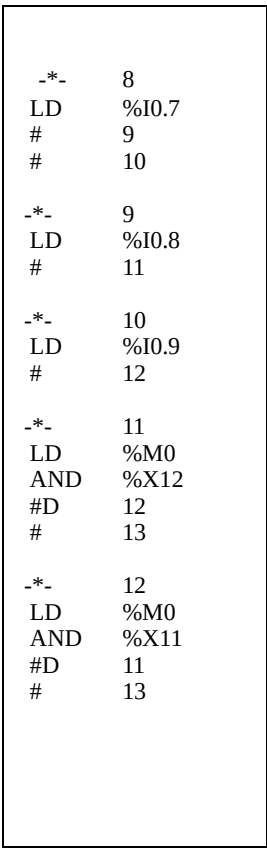
Simultaneous sequences:



Not supported



Twido Ladder
Language program



Twido Instruction
List program

Note: For a Grafcet Chart to be operational, at least one active step must be declared using the =*=i instruction (initial step) or the chart should be pre-positioned during preprocessing using system bit %S23 and the instruction S %Xi.

Description of Grafcet Program Structure

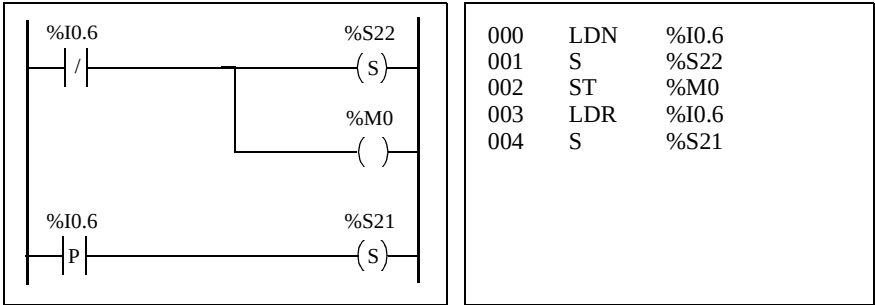
Introduction A TwidoSoft Grafcet program has three parts:

- Preprocessing
- Sequential processing
- Post-Processing

Preprocessing Preprocessing consists of the following:

- Power returns
- Faults
- Changes of operating mode
- Pre-positioning Grafcet steps
- Input logic

The rising edge of input %I0.6 sets bit %S21 to 1. This disables the active steps and enables the inactive steps.



Preprocessing begins with the first line of the program and ends with the first occurrence of a "= * =" or "- * -" instruction.

Three system bits are dedicated to Grafcet control: %S21, %S22 and %S23. Each of these system bits are set to 1 (if needed) by the application, normally in preprocessing. The associated function is performed by the system at the end of preprocessing and the system bit is then reset to 0 by the system.

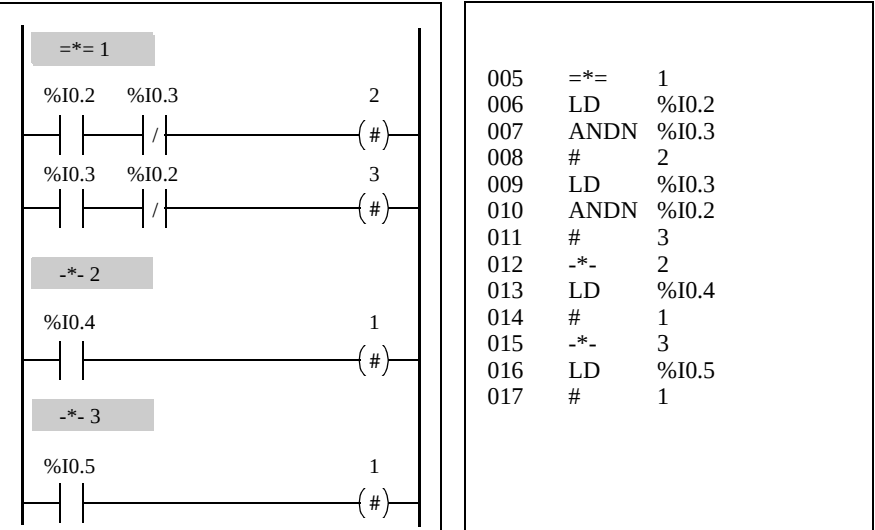
System Bit	Name	Description
%S21	Grafcet initialization	All active steps are deactivated and the initial steps are activated.
%S22	Grafcet re-initialization	All steps are deactivated.
%S23	Grafcet pre-positioning	This bit must be set to 1 if %Xi objects are explicitly written by the application in preprocessing. If this bit is maintained to 1 by the preprocessing without any explicit change of the %Xi objects, Grafcet is frozen (no updates are taken into account).

Sequential Processing

Sequential processing takes place in the chart (instructions representing the chart):

- Steps
- Actions associated with steps
- Transitions
- Transition conditions

Example:

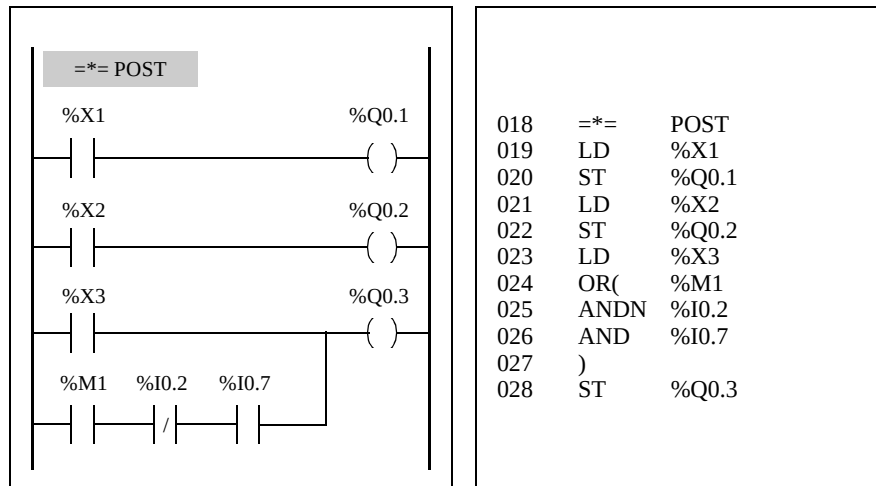


Sequential processing ends with the execution of the "= * = POST" instruction or with the end of the program.

Post-Processing Post-processing consists of the following:

- Commands from the sequential processing for controlling the outputs
- Safety interlocks specific to the outputs

Example:



Actions Associated with Grafcet Steps

Introduction

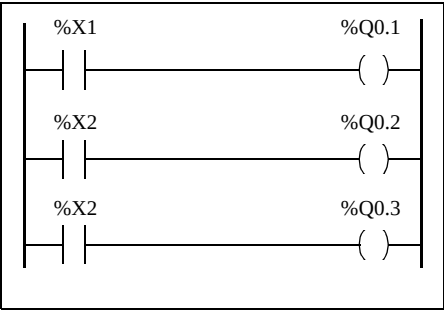
A TwidoSoft Grafcet program offers two ways to program the actions associated with steps:

- In the post-processing section
- Within List instructions or Ladder rungs of the steps themselves

Associating Actions in Post-Processing

If there are security or running mode constraints, it is preferable to program actions in the post-processing section of a Grafcet application. You can use Set and Reset List instructions or energize coils in a Ladder program to activate Grafcet steps (%Xi).

Example:

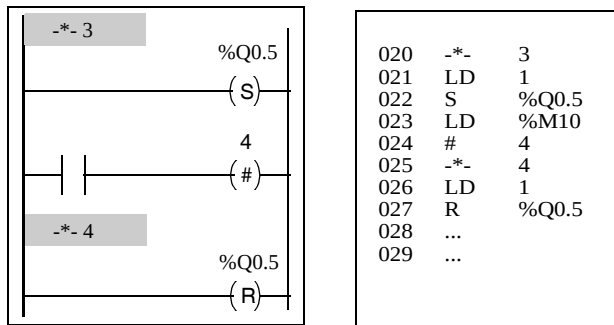


```
018  ==  POST
019  LD   %X1
020  ST   %Q0.1
021  LD   %X2
022  ST   %Q0.2
023  LD   %X3
024  ST   %Q0.3
```

Associating Actions from an Application

You can program the actions associated with steps within List instructions or Ladder rungs. In this case, the List instruction or Ladder rung is not scanned unless the step is active. This is the most efficient, readable, and maintainable way to use Grafcet.

Example:



Description of Instructions and Functions



At a Glance

Subject of this Part This part provides detailed descriptions about basic and advanced instructions and system bits and words for Twido languages.

What's in this Part? This part contains the following chapters:

Chapter	Chapter Name	Page
16	Basic Instructions	367
17	Advanced Instructions	435
18	System Bits and System Words	595

At a Glance

Subject of this Chapter

This chapter provides details about instructions and function blocks that are used to create basic control programs for Twido controllers.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
16.1	Boolean Processing	369
16.2	Basic Function Blocks	385
16.3	Numerical Processing	409
16.4	Program Instructions	428

16.1 Boolean Processing

At a Glance

Aim of this Section

This section provides an introduction to Boolean processing including descriptions and programming guidelines for Boolean instructions.

What's in this Section?

This section contains the following topics:

Topic	Page
Boolean Instructions	370
Understanding the Format for Describing Boolean Instructions	372
Load Instructions (LD, LDN, LDR, LDF)	374
Assignment instructions (ST, STN, R, S)	376
Logical AND Instructions (AND, ANDN, ANDR, ANDF)	378
Logical OR Instructions (OR, ORN, ORR, ORF)	380
Exclusive OR, instructions (XOR, XORN, XORR, XORF)	382
NOT Instruction (N)	384

Boolean Instructions

Introduction

Boolean instructions can be compared to Ladder language elements. These instructions are summarized in the following table.

Item	Instruction	Example	Description
Test elements	The Load (LD) instruction is equivalent to an open contact.	LD %I0.0	Contact is closed when bit %I0.0 is at state 1.
Action elements	The Store (ST) instruction is equivalent to a coil.	ST %Q0.0	The associated bit object takes a logical value of the bit accumulator (result of previous logic).

The Boolean result of the test elements is applied to the action elements as shown by the following instructions.

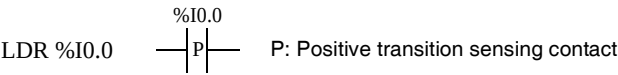
LD %I0.0
AND %I0.1
ST %Q0.0

Testing Controller Inputs

Boolean test instructions can be used to detect rising or falling edges on the controller inputs. An edge is detected when the state of an input has changed between "scan n-1" and the current "scan n". This edge remains detected during the current scan.

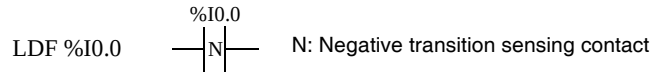
Rising Edge Detection

The LDR instruction (Load Rising Edge) is equivalent to a rising edge detection contact. The rising edge detects a change of the input value from 0 to 1. A positive transition sensing contact is used to detect a rising edge as seen in the following diagram.



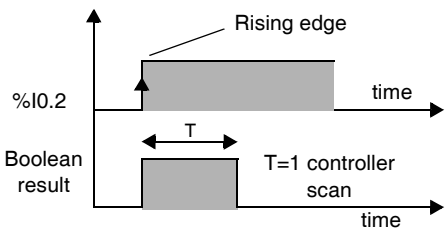
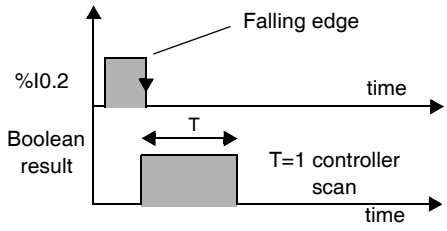
Falling Edge Detection

The LDF instruction (Load Falling Edge) is equivalent to a falling edge detection contact. The falling edge detects a change of the controlling input from 1 to 0. A negative transition sensing contact is used to detect a falling edge as seen in the following diagram.



Edge Detection

The following table summarizes the instructions and timing for detecting edges:

Edge	Test Instruction	Ladder diagram	Timing diagram
Rising edge	LDR %I0.0		
Falling edge	LDF %I0.0		

Note: It is now possible to apply edge instructions to the %Mi internal bits.

Understanding the Format for Describing Boolean Instructions

Introduction

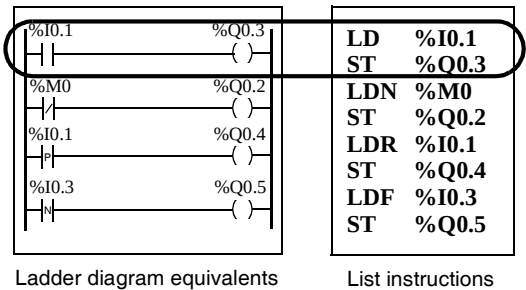
Each Boolean instruction in this section is described using the following information:

- Brief description
- Example of the instruction and the corresponding ladder diagram
- List of permitted operands
- Timing diagram

The following explanations provide more detail on how Boolean instructions are described in this section.

Examples

The following illustration shows how examples are given for each instruction.

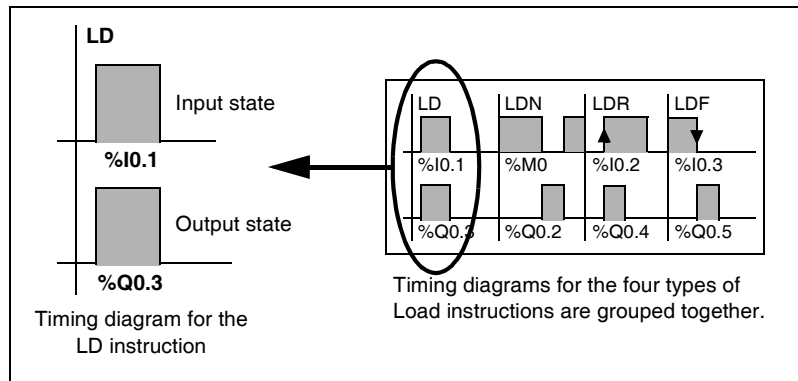


Permitted Operands

The following table defines the types of permitted operands used for Boolean instructions.

Operand	Description
0/1	Immediate value of 0 or 1
%I	Controller input %Ii.j
%Q	Controller output %Qi.j
%M	Internal bit %Mi
%S	System bit %Si
%X	Step bit %Xi
%BLK.x	Function block bit (for example, %TMi.Q)
%•:Xk	Word bit (for example, %MWi:Xk)
[	Comparison expression (for example, [%MWi<1000])

Timing Diagrams The following illustration shows how timing diagrams are displayed for each instruction.



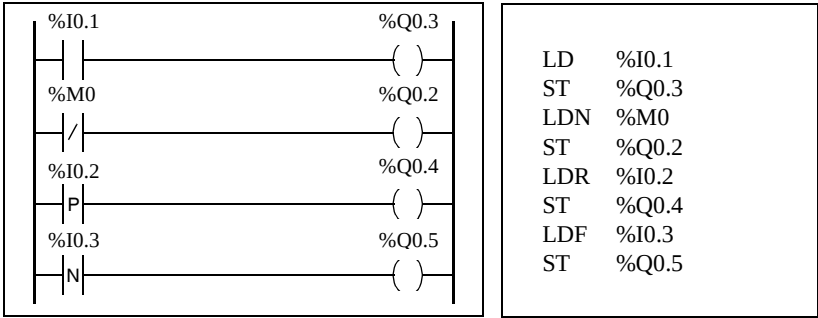
Load Instructions (LD, LDN, LDR, LDF)

Introduction

Load instructions LD, LDN, LDR, and LDF correspond respectively to the opened, closed, rising edge, and falling edge contacts (LDR and LDF are used only with controller inputs and internal words, and for AS-Interface and PDO CANopen slave inputs).

Examples

The following diagrams are examples of Load instructions.

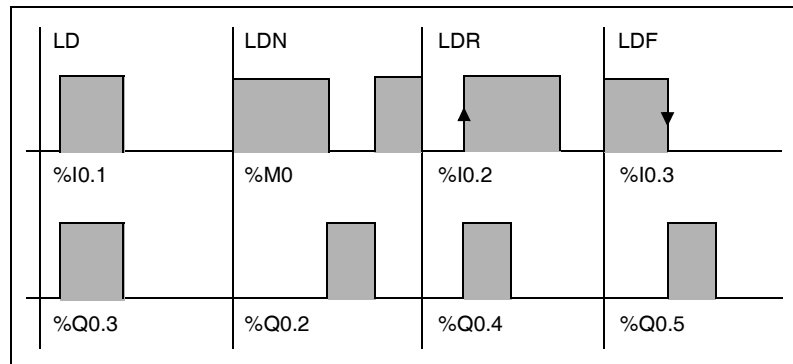


Permitted Operands

The following table lists the types of load instructions with Ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
LD		0/1, %I, %IA, %IWCx.y.z:Xk, %Q, %QA, %M, %S, %X, %BLK.x, %*:Xk,[
LDN		0/1, %I, %IA, %IWCx.y.z:Xk, %Q, %QA, %M, %S, %X, %BLK.x, %*:Xk,[
LDR		%I, %IA, %M
LDF		%I, %IA, %M

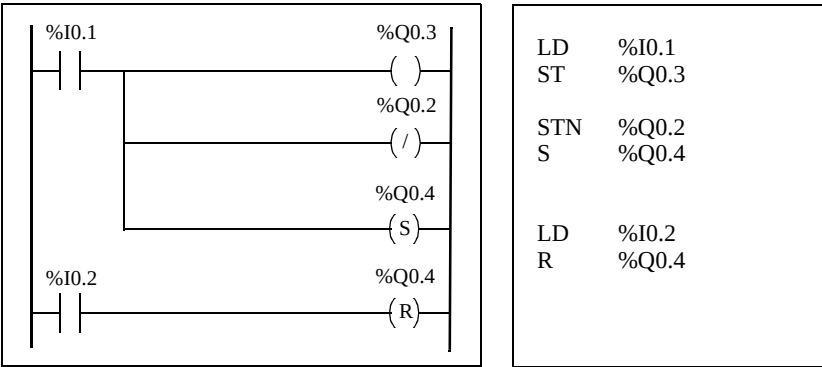
Timing diagram The following diagram displays the timing for Load instructions.



Assignment instructions (ST, STN, R, S)

Introduction The assignment instructions ST, STN, S, and R correspond respectively to the direct, inverse, set, and reset coils.

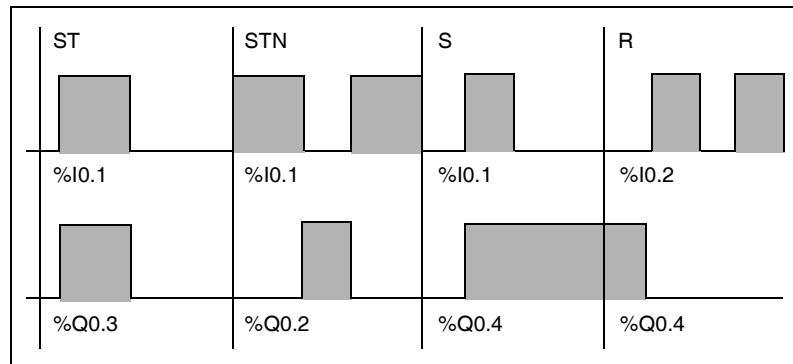
Examples The following diagrams are examples of assignment instructions.



Permitted Operands The following table lists the types of assignment instructions with ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
ST	()	%Q,%QA,%M,%S,%BLK.x,%•:Xk
STN	(/)	%Q,%QA%M,%S,%BLK.x,%•:Xk
S	(s)	%Q,%QA,%M,%S,%X,%BLK.x,%•:Xk
R	(R)	%Q,%QA,%M,%S,%X,%BLK.x,%•:Xk

Timing diagram The following diagram displays the timing for assignment instructions.



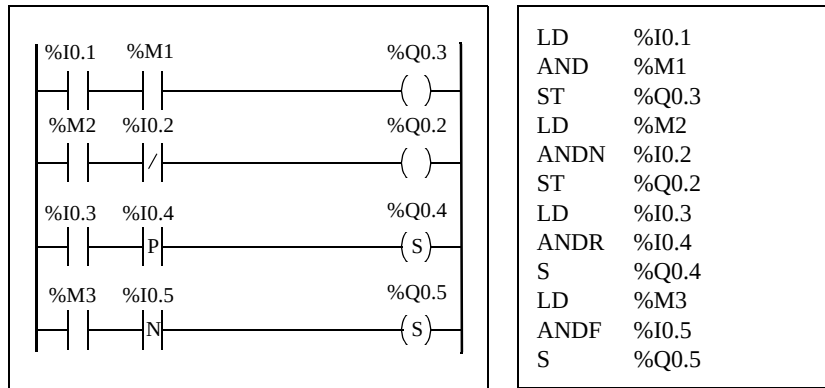
Logical AND Instructions (AND, ANDN, ANDR, ANDF)

Introduction

The AND instructions perform a logical AND operation between the operand (or its inverse, or its rising or falling edge) and the Boolean result of the preceding instruction.

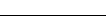
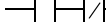
Examples

The following diagrams are examples of logic AND instructions.

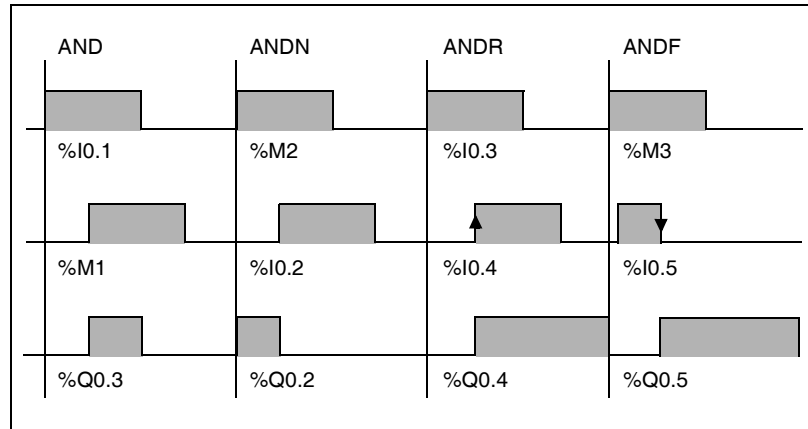


Permitted Operands

The following table lists the types of AND instructions with ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
AND		0/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %•:Xk, [
ANDN		0/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %•:Xk, [
ANDR		%I, %IA, %M
ANDF		%I, %IA, %M

Timing diagram The following diagram displays the timing for the AND instructions.



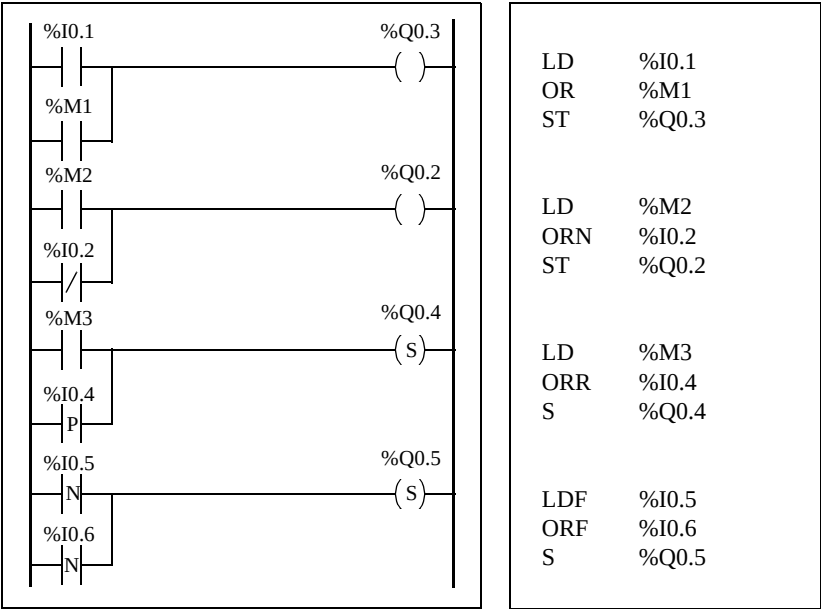
Logical OR Instructions (OR, ORN, ORR, ORF)

Introduction

The OR instructions perform a logical OR operation between the operand (or its inverse, or its rising or falling edge) and the Boolean result of the preceding instruction.

Examples

The following diagrams are examples of logic OR instructions.



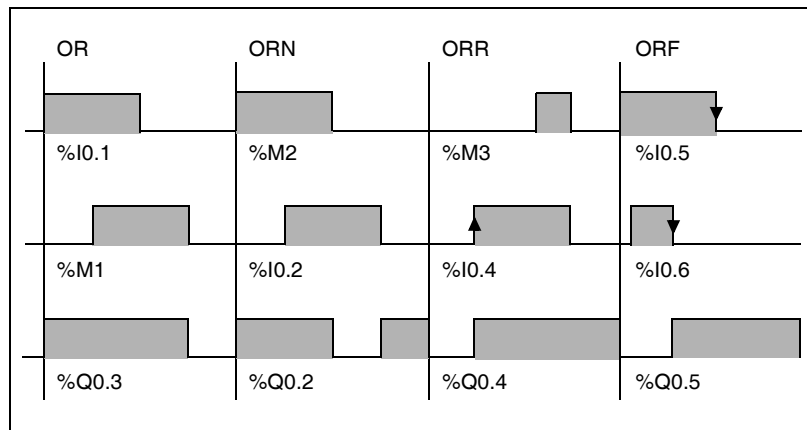
Permitted Operands

The following table lists the types of OR instructions with Ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
OR		0/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %•:Xk
ORN		0/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %•:Xk
ORR		%I, %IA, %M
ORF		%I, %IA, %M

Timing diagram

The following diagram displays the timing for the OR instructions.



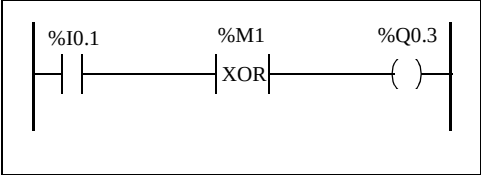
Exclusive OR, instructions (XOR, XORN, XORR, XORF)

Introduction

The XOR instructions perform an exclusive OR operation between the operand (or its inverse, or its rising or falling edge) and the Boolean result of the preceding instruction.

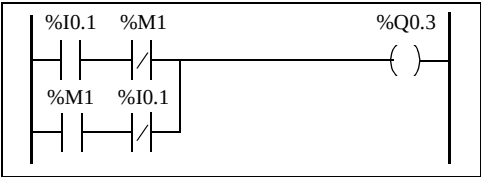
Examples

The following example shows the use of XOR instructions.
Schematic using XOR instruction:



```
LD    %I0.1
XOR   %M1
ST    %Q0.3
```

Schematic NOT using XOR instruction :



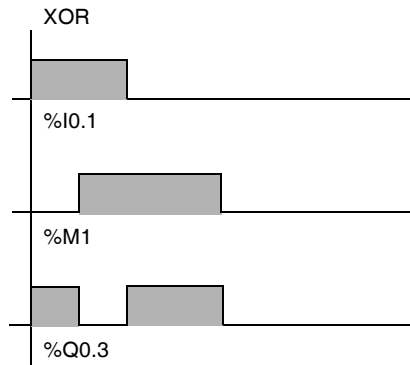
```
LD    %I0.1
ANDN  %M1
OR(    %M1
ANDN  %I0.1
)
ST    %Q0.3
```

Permitted Operands

The following table lists the types of XOR instructions and permitted operands.

List instruction	Permitted Operands
XOR	%I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %*:Xk
XORN	%I, %IA, %Q, %QA, %M, %S, %X, %BLK.x, %*:Xk
XORR	%I, %IA, %M
XORF	%I, %IA, %M

Timing Diagram The following diagram displays the timing for the XOR instructions.

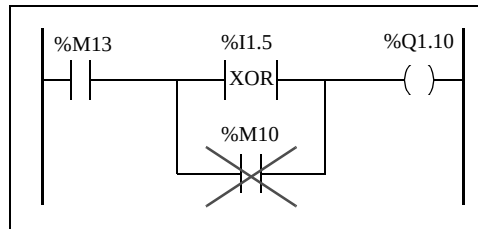


Special Cases

The following are special precautions for using XOR instructions in Ladder programs:

- Do not insert XOR contacts in the first position of a rung.
- Do not insert XOR contacts in parallel with other ladder elements (see the following example.)

As shown in the following example, inserting an element in parallel with the XOR contact will generate a validation error.



NOT Instruction (N)

Introduction The NOT (N) instruction negates the Boolean result of the preceding instruction.

Example The following is an example of using the NOT instruction.

LD

%I0.1

OR

%M2

ST

%Q0.2

N

AND

%M3

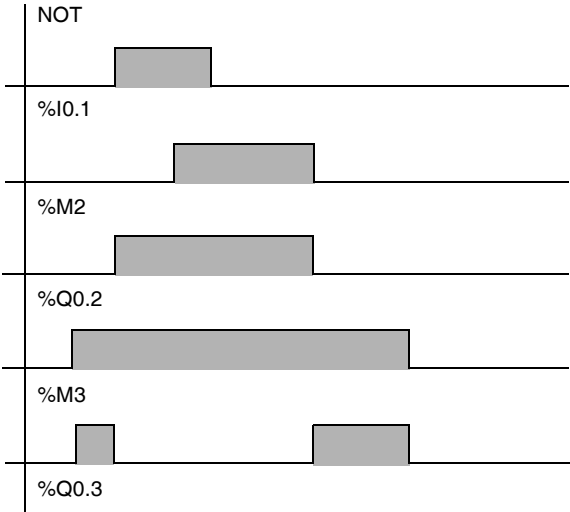
ST

%Q0.3

Note: The NOT instruction is not reversible.

Permitted Operands Not applicable.

Timing Diagram The following diagram displays the timing for the NOT instruction.



16.2 Basic Function Blocks

At a Glance

Aim of this Section

This section provides descriptions and programming guidelines for using basic function blocks.

What's in this Section?

This section contains the following topics:

Topic	Page
Basic Function Blocks	386
Standard function blocks programming principles	388
Timer Function Block (%TMi)	390
TOF Type of Timer	392
TON Type of Timer	393
TP Type of Timer	394
Programming and Configuring Timers	395
Up/Down Counter Function Block (%Ci)	398
Programming and Configuring Counters	402
Shift Bit Register Function Block (%SB Ri)	404
Step Counter Function Block (%SCi)	406

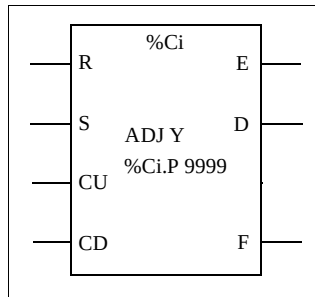
Basic Function Blocks

Introduction

Function blocks are the sources for bit objects and specific words that are used by programs. Basic function blocks provide simple functions such as timers or up/down counting.

Example of a Function Block

The following illustration is an example of an up/down Counter function block.



Up/down counter block

Bit Objects

Bit objects correspond to the block outputs. These bits can be accessed by Boolean test instructions using either of the following methods:

- Directly (for example, LD E) if they are wired to the block in reversible programming (see *Standard function blocks programming principles*, p. 388).
- By specifying the block type (for example, LD %Ci.E).

Inputs can be accessed in the form of instructions.

Word Objects

Word objects correspond to specified parameters and values as follows:

- **Block configuration parameters:** Some parameters are accessible by the program (for example, pre-selection parameters) and some are inaccessible by the program (for example, time base).
 - **Current values:** For example, %Ci.V, the current count value.
-

Accessible Bit and Word Objects

The following table describes the Basic function blocks bit and word objects that can be accessed by the program.

Basic Function Block	Symbol	Range (i)	Types of Objects	Description	Address	Write Access
Timer	%T <i>Mi</i>	0 - 127	Word	Current Value	%T <i>Mi</i> .V	no
				Preset value	%T <i>Mi</i> .P	yes
			Bit	Timer output	%T <i>Mi</i> .Q	no
Up/Down Counter	%C <i>i</i>	0 - 127	Word	Current Value	%C <i>i</i> .V	no
				Preset value	%C <i>i</i> .P	yes
			Bit	Underflow output (empty)	%C <i>i</i> .E	no
				Preset output reached	%C <i>i</i> .D	no
				Overflow output (full)	%C <i>i</i> .F	no

Standard function blocks programming principles

Introduction

Use one of the following methods to program standard function blocks:

- Function block instructions (for example, `BLK %TM2`): This reversible method of programming ladder language enables operations to be performed on the block in a single place in the program.
- Specific instructions (for example, `CU %Ci`): This non-reversible method enables operations to be performed on the block's inputs in several places in the program (for example, line 100 `CU %C1`, line 174 `CD %C1`, line 209 `LD %C1.D`).

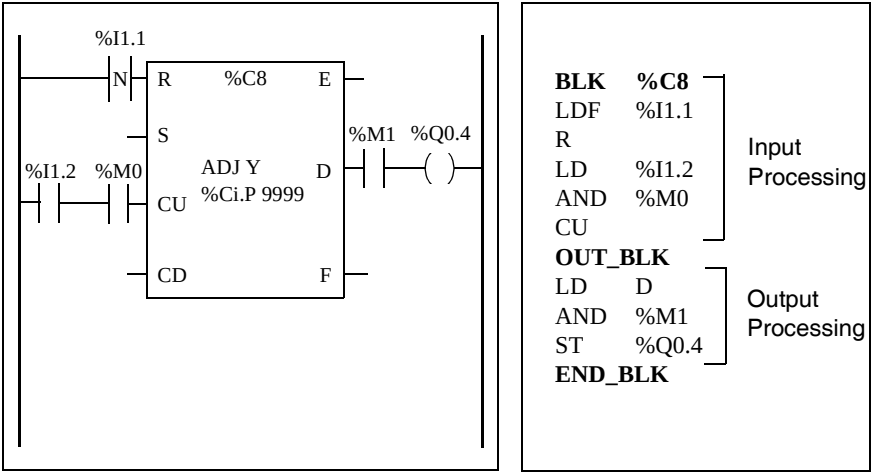
Reversible Programming

Use instructions `BLK`, `OUT_BLK`, and `END_BLK` for reversible programming:

- **BLK**: Indicates the beginning of the block.
- **OUT_BLK**: Is used to directly wire the block outputs.
- **END_BLK**: Indicates the end of the block.

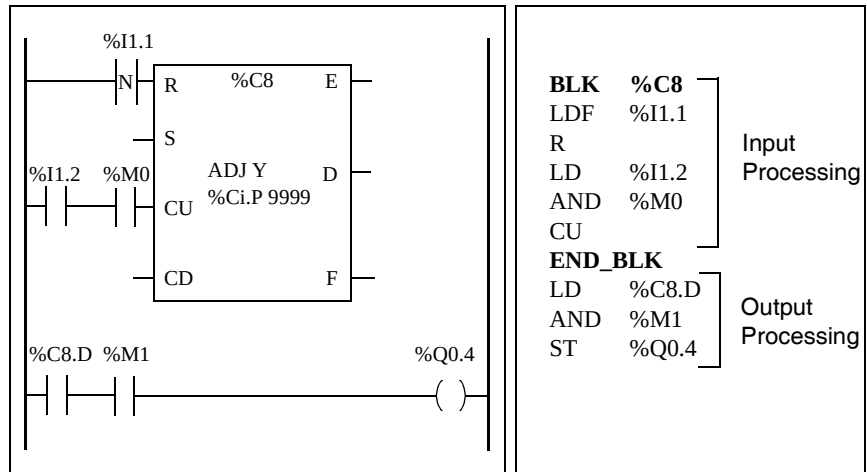
Example with Output Wiring

The following example shows reversible programming of a counter function block with wired outputs.



Example without Output Wiring

This example shows reversible programming of a counter function block without wired outputs.



Note: Only test and input instructions on the relevant block can be placed between the BLK and OUT_BLK instructions (or between BLK and END_BLK when OUT_BLK is not programmed).

Timer Function Block (%Tmi)

Introduction

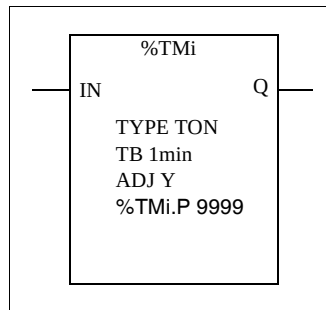
There are three types of Timer function blocks:

- TON (Timer On-Delay): this type of timer is used to control on-delay actions.
- TOF (Timer Off-Delay): this type of timer is used to control off-delay actions.
- TP (Timer - Pulse): this type of timer is used to create a pulse of a precise duration.

The delays or pulse periods are programmable and may be modified using the TwidoSoft.

Illustration

The following is an illustration of the Timer function block.



Timer function block

Parameters

The Timer function block has the following parameters:

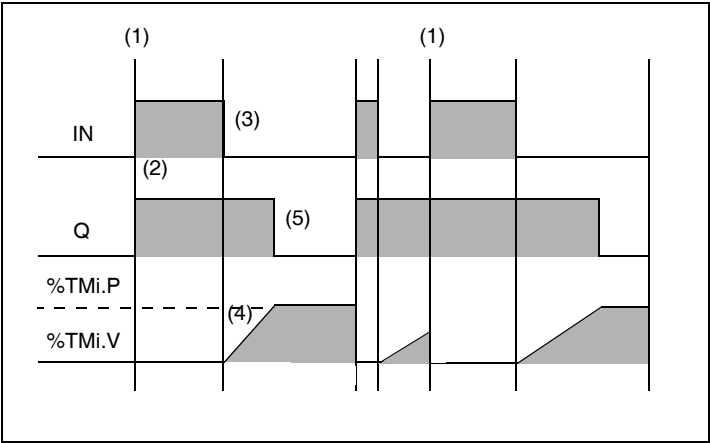
Parameter	Label	Value
Timer number	%TMI	0 to 63: TWDLCAA10DRF and TWDLCAA16DRF 0 to 127 for all other controllers.
Type	TON	• Timer On-Delay (default)
	TOF	• Timer Off-Delay
	TP	• pulse (monostable)
Time base	TB	1 min (default), 1 s, 100 ms, 10 ms, 1 ms
Current Value	%TMI.V	Word which increments from 0 to %TMI.P when the timer is running. May be read and tested, but not written by the program. %TMI.V can be modified using the Animation Tables Editor.
Preset value	%TMI.P	0 - 9999. Word which may be read, tested, and written by the program. Default value is 9999. The period or delay generated is %TMI.P x TB.
Animation Tables Editor	Y/N	Y: Yes, the preset %TMI.P value can be modified using the Animation Tables Editor. N: No, the preset %TMI.P value cannot be modified.
Enable (or instruction) input	IN	Starts the timer on rising edge (TON or TP types) or falling edge (TOF type).
Timer output	Q	Associated bit %TMI.Q is set to 1 depending on the function performed: TON, TOF, or TP

Note: The larger the preset value, the greater the timer accuracy.

TOF Type of Timer

Introduction Use the TOF (Timer Off-Delay) type of timer to control off-delay actions. This delay is programmable using TwidoSoft.

Timing Diagram The following timing diagram illustrates the operation of the TOF type timer.



Operation The following table describes the operation of the TOF type timer.

Phase	Description
1	The current value %Tmi.V is set to 0 on a rising edge at input IN, even if the timer is running.
2	The %Tmi.Q output bit is set to 1 when a rising edge is detected at input N.
3	The timer starts on the falling edge of input IN.
4	The current value %Tmi.V increases to %Tmi.P in increments of one unit for each pulse of the time base TB.
5	The %Tmi.Q output bit is reset to 0 when the current value reaches %Tmi.P.

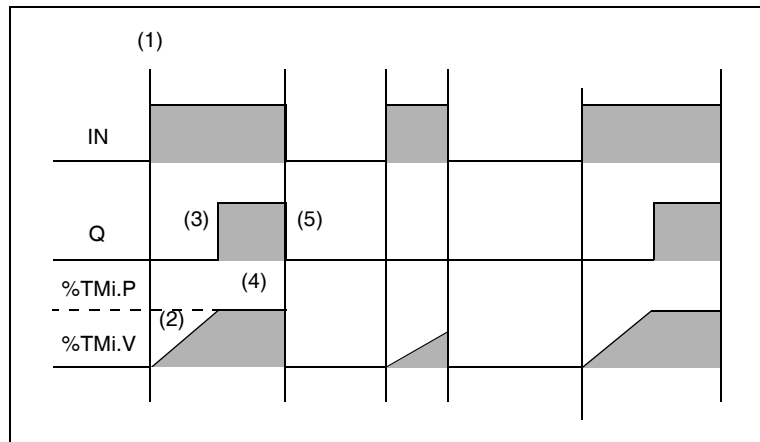
TON Type of Timer

Introduction

The TON (Timer On-Delay) type of timer is used to control on-delay actions. This delay is programmable using the TwidoSoft.

Timing Diagram

The following timing diagram illustrates the operation of the TON type timer.



Operation

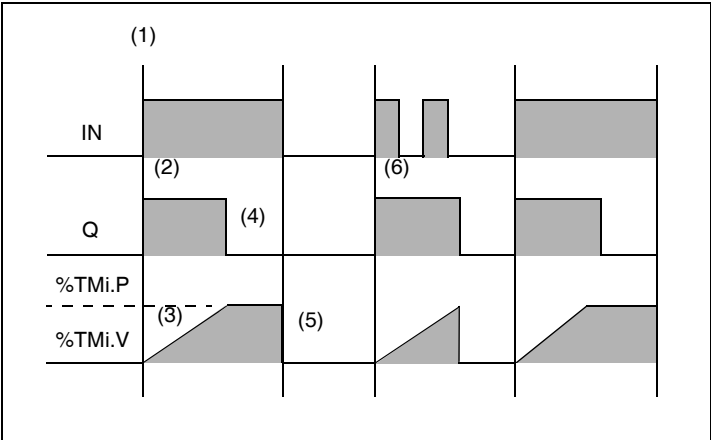
The following table describes the operation of the TON type timer.

Phase	Description
1	The timer starts on the rising edge of the IN input.
2	The current value %TMI.V increases from 0 to %TMI.P in increments of one unit for each pulse of the time base TB.
3	The %TMI.Q output bit is set to 1 when the current value has reached %TMI.P.
4	The %TMI.Q output bit remains at 1 while the IN input is at 1.
5	When a falling edge is detected at the IN input, the timer is stopped, even if the timer has not reached %TMI.P, and %TMI.V is set to 0.

TP Type of Timer

Introduction The TP (Timer - Pulse) type of timer is used to create pulses of a precise duration. This delay is programmable using the TwidoSoft.

Timing Diagram The following timing diagram illustrates the operation of the TP type timer.



Operation The following table describes the operation of the TP type timer.

Phase	Description
1	The timer starts on the rising edge of the IN input. The current value %Tmi.V is set to 0 if the timer has not already started.
2	The %Tmi.Q output bit is set to 1 when the timer starts.
3	The current value %Tmi.V of the timer increases from 0 to %Tmi.P in increments of one unit per pulse of the time base TB.
4	The %Tmi.Q output bit is set to 0 when the current value has reached %Tmi.P.
5	The current value %Tmi.V is set to 0 when %Tmi.V equals %Tmi.P and input IN returns to 0.
6	This timer cannot be reset. Once %Tmi.V equals %Tmi.P, and input IN is 0, then %Tmi.V is set to 0.

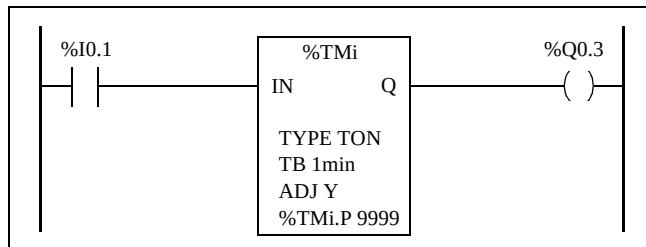
Programming and Configuring Timers

Introduction

Timer function blocks (%T_{Mi}) are programmed in the same way regardless of how they are to be used. The timer function (TON, TOF, or TP) is selected during configuration.

Examples

The following illustration is a timer function block with examples of reversible and non-reversible programming.



Reversible programming

```

BLK    %TM1
LD      %I0.1
IN
OUT_BLK
LD      Q
ST      %Q0.3
END_BLK
  
```

Non-Reversible programming

```

LD      %I0.1
IN      %TM1
LD      %TM1.Q
ST      %Q0.3
  
```

Configuration

The following parameters must be entered during configuration:

- Timer type: TON, TOF, or TP
- Timebase: 1 min, 1 s, 100 ms, 10 ms or 1 ms
- Preset value (%T_{Mi}.P): 0 to 9999
- Adjust: Checked or Not Checked

Special Cases

The following table contains a list of special cases for programming the Timer function block.

Special case	Description
Effect of a cold restart (%S0=1)	Forces the current value to 0. Sets output %Tmi.Q to 0. The preset value is reset to the value defined during configuration.
Effect of a warm restart (%S1=1)	Has no effect on the current and preset values of the timer. The current value does not change during a power outage.
Effect of a controller stop	Stopping the controller does not freeze the current value.
Effect of a program jump	Jumping over the timer block does not freeze the timer. The timer will continue to increment until it reaches the preset value (%Tmi.P). At that point, the Done bit (%Tmi.Q) assigned to output Q of the timer block changes state. However, the associated output wired directly to the block output is not activated and not scanned by the controller.
Testing by bit %Tmi.Q (done bit)	It is advisable to test bit %Tmi.Q only once in the program.
Effect of modifying the preset %Tmi.P	Modifying the present value by using an instruction or by adjusting the value only takes effect on the next activation of the timer.

Timers with a 1 ms Time Base

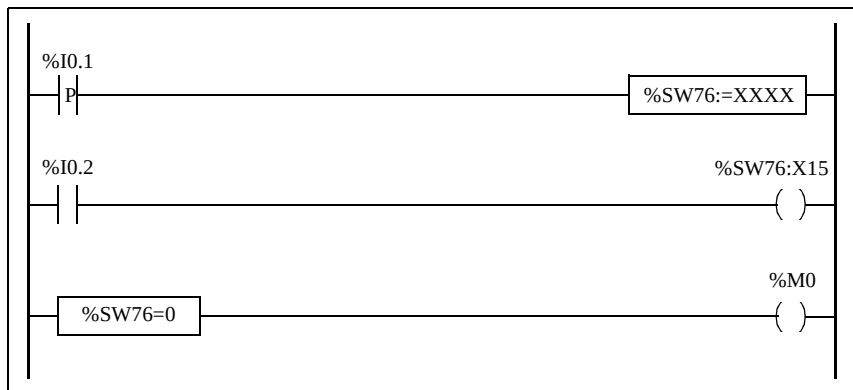
The 1 ms time base is only available with the first five timers. The four system words %SW76, %SW77, %SW78, and SW79, can be used as "hourglasses." These four words are decremented individually by the system every millisecond **if they have a positive value.**

Multiple timing can be achieved by successive loading of one of these words or by testing the intermediate values. If the value of one of these four words is less than 0, it will not be modified. A timer can be "frozen" by setting the corresponding bit 15 to 1, and then "unfrozen" by resetting it to 0.

**Programming
Example**

The following is an example of programming a timer function block.

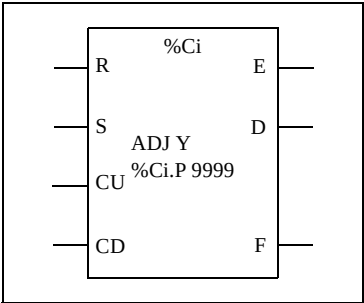
```
LDR    %I0.1      (Launching the timer on the rising edge of %I0.1)
[%SW76:=XXXX]    (XXXX = required value)
LD      %I0.2      (optional management of freeze, input I0.2 freezes)
ST      %SW76:X15
LD      [%SW76=0]  (timer end test)
ST      %M0
.....
```



Up/Down Counter Function Block (%Ci)

Introduction The Counter function block (%Ci) provides up and down counting of events. These two operations can be done simultaneously.

Illustration The following is an illustration of the up/down Counter function block.



Up/down counter function block

Parameters

The Counter function block has the following parameters:

Parameter	Label	Value
Counter number	%Ci	0 to 127
Current Value	%Ci.V	Word is incremented or decremented according to inputs (or instructions) CU and CD. Can be read and tested but not written by the program. Use the Data Editor to modify %Ci.V.
Preset value	%Ci.P	$0 \leq \%Ci.P \leq 9999$. Word can be read, tested, and written (default value: 9999).
Edit using the Animation Tables Editor	ADJ	- Y: Yes, the preset value can be modified by using the Animation Tables Editor. - N: No, the preset value cannot be modified by using the Animation Tables Editor.
Reset input (or instruction)	R	At state 1: %Ci.V = 0.
Reset input (or instruction)	S	At state 1: %Ci.V = %Ci.P.
Upcount input (or instruction)	CU	Increments %Ci.V on a rising edge.
Downcount input (or instruction)	CD	Decrements %Ci.V on a rising edge.
Downcount overflow output	E (Empty)	The associated bit %Ci.E=1, when down counter %Ci.V changes from 0 to 9999 (set to 1 when %Ci.V reaches 9999, and reset to 0 if the counter continues to count down).
Preset output reached	D (Done)	The associated bit %Ci.D=1, when %Ci.V=%Ci.P.
Upcount overflow output	F (Full)	The associated bit %Ci.F=1, when %Ci.V changes from 9999 to 0 (set to 1 when %Ci.V reaches 0, and reset to 0 if the counter continues to count up).

Operation

The following table describes the main stages of up/down counter operation.

Operation	Action	Result
Counting	A rising edge appears at the upcounting input CU (or instruction CU is activated).	The %Ci.V current value is incremented by one unit.
	The %Ci.V current value is equal to the %Ci.P preset value.	The "preset reached" output bit %Ci.D switches to 1.
	The %Ci.V current value changes from 9999 to 0.	The output bit %Ci.F (upcounting overflow) switches to 1.
	If the counter continues to count up.	The output bit %Ci.F (upcounting overflow) is reset to zero.
Downcount	A rising edge appears at the downcounting input CD (or instruction CD is activated).	The current value %Ci.V is decremented by one unit.
	The current value %Ci.V changes from 0 to 9999.	The output bit %Ci.E (downcounting overflow) switches to 1.
	If the counter continues to count down.	The output bit %Ci.F (downcounting overflow) is reset to zero.
Up/down count	To use both the upcount and downcount functions simultaneously (or to activate both instructions CD and CU), the two corresponding inputs CU and CD must be controlled simultaneously. These two inputs are then scanned in succession. If they are both at 1, the current value remains unchanged.	
Reset	Input R is set to state 1 (or the R instruction is activated).	The current value %Ci.V is forced to 0. Outputs %Ci.E, %Ci.D and %Ci.F are at 0. The reset input has priority.
Preset	If input S is set to 1 (or the S instruction is activated) and the reset input is at 0 (or the R instruction is inactive).	The current value %Ci.V takes the %Ci.P value and the %Ci.D output is set to 1.

Special Cases

The following table shows a list of special operating/configuration cases for counters.

Special case	Description
Effect of a cold restart (%S0=1)	• The current value %Ci.V is set to 0. • Output bits %Ci.E, %Ci.D, and %Ci.F are set to 0. • The preset value is initialized with the value defined during configuration.
Effect of a warm restart (%S1=1) of a controller stop	Has no effect on the current value of the counter (%Ci.V).
Effect of modifying the preset %Ci.P	Modifying the preset value via an instruction or by adjusting it takes effect when the block is processed by the application (activation of one of the inputs).

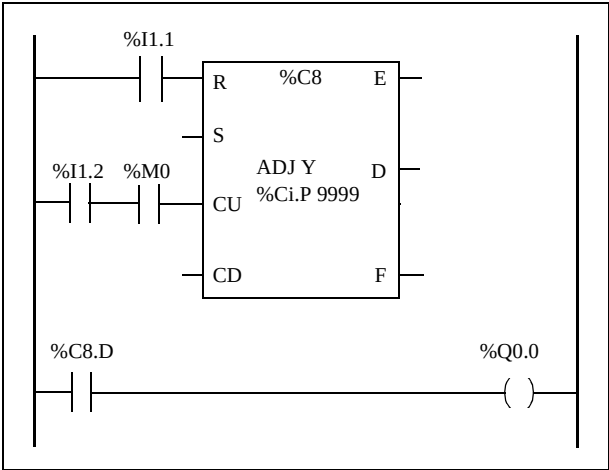
Programming and Configuring Counters

Introduction

The following example is a counter that provides a count of up to 5000 items. Each pulse on input %I1.2 (when internal bit %M0 is set to 1) increments the counter %C8 up to its final preset value (bit %C8.D=1). The counter is reset by input %I1.1.

Programming Example

The following illustration is a counter function block with examples of reversible and non-reversible programming.



Ladder diagram

BLK	%C8
LD	%I1.1
R	%C8
LD	%I1.2
AND	%M0
CU	%C8
END_BLK	
LD	%C8.D
ST	%Q0.0

Reversible Programming

LD	%I1.1
R	%C8
LD	%I1.2
AND	%M0
CU	%C8
LD	%C8.D
ST	%Q0.0

Non-Reversible programming

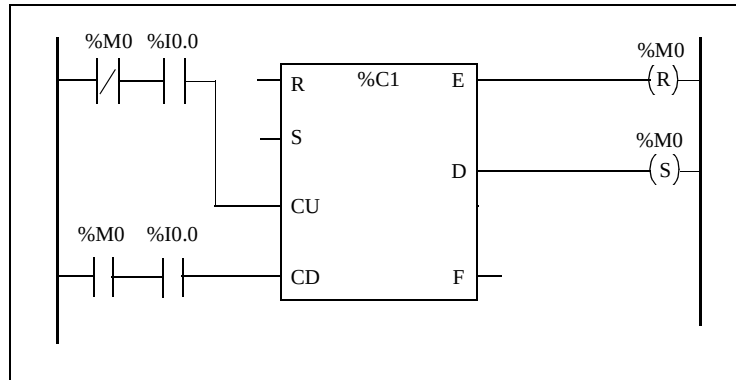
Configuration

The following parameters must be entered during configuration:

- Preset value (%Ci.P): set to 5000 in this example
- Adjust: Yes

Example of an Up/Down Counter

The following illustration is an example of an Up/Down Counter function block.



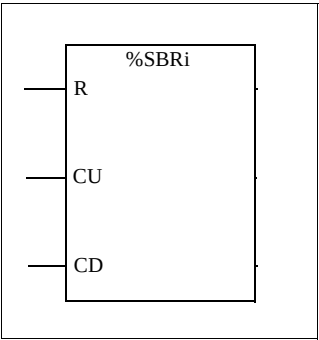
Ladder diagram

In this example, if we take %C1.P 4, the current value of the %C1.V counter will be incremented from 0 to 3, then decremented from 3 to 0. Whereas %I0.0=1 %C1.V oscillates between 0 and 3.

Shift Bit Register Function Block (%SBRi)

Introduction The Shift Bit Register function block (%SBRi) provides a left or right shift of binary data bits (0 or 1).

Illustration The following is an example of a Shift Register function block.

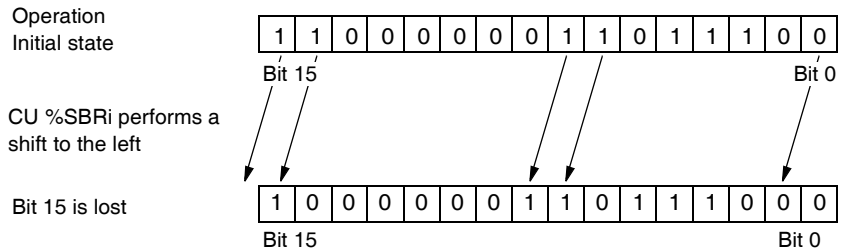


Parameters The Shift Bit Register function block has the following parameters.

Parameter	Label	Value
Register number	%SBRi	0 to 7
Register bit	%SBRi.j	Bits 0 to 15 (j = 0 to 15) of the shift register can be tested by a Test instruction and written using an Assignment instruction.
Reset input (or instruction)	R	When function parameter R is 1, this sets register bits 0 to 15 %SBRi.j to 0.
Shift to left input (or instruction)	CU	On a rising edge, shifts a register bit to the left.
Shift to right input (or instruction)	CD	On a rising edge, shifts a register bit to the right.

Operation

The following illustration shows a bit pattern before and after a shift operation.

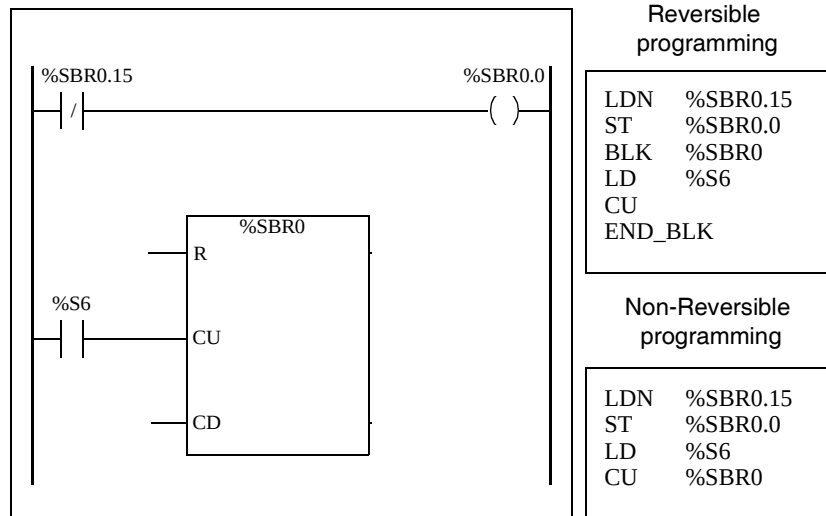


This is also true of a request to shift a bit to the right (Bit 15 to Bit 0) using the CD instruction. Bit 0 is lost.

If a 16-bit register is not adequate, it is possible to use the program to cascade several registers.

Programming

In the following example, a bit is shifted to the left every second while Bit 0 assumes the opposite state to Bit 15.



Special Cases

The following table contains a list of special cases for programming the Shift Bit Register function block.

Special Case	Description
Effect of a cold restart (%S0=1)	Sets all the bits of the register word to 0.
Effect of a warm restart (%S1=1)	Has no effect on the bits of the register word.

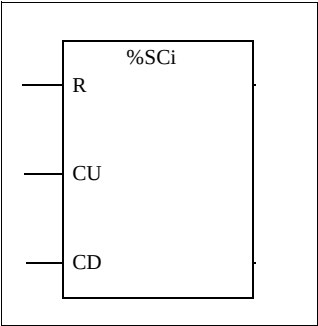
Step Counter Function Block (%SCi)

Introduction

A Step Counter function block (%SCi) provides a series of steps to which actions can be assigned. Moving from one step to another depends on external or internal events. Each time a step is active, the associated bit is set to 1. Only one step of a step counter can be active at a time.

Illustration

The following is an example of a Step Counter function block.

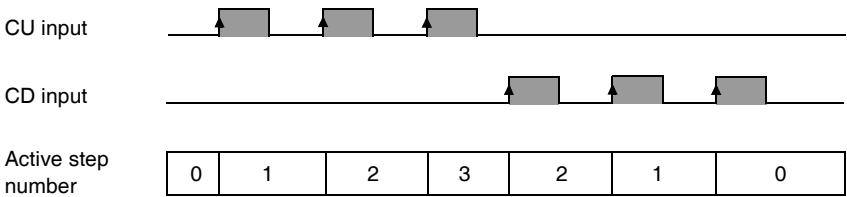


Parameters

The step function block has the following parameters:

Parameter	Label	Value
Step counter number	%SCi	0 - 7
Step Counter bit	%SCi.j	Step counter bits 0 to 255 (j = 0 to 255) can be tested by a Load logical operation and written by an Assignment instruction.
Reset input (or instruction)	R	When function parameter R is 1, this resets the step counter.
Increment input (or instruction)	CU	On a rising edge, increments the step counter by one step.
Decrement input (or instruction)	CD	On a rising edge, decrements the step counter by one step.

Timing Diagram The following timing diagram illustrates the operation of the step function block.

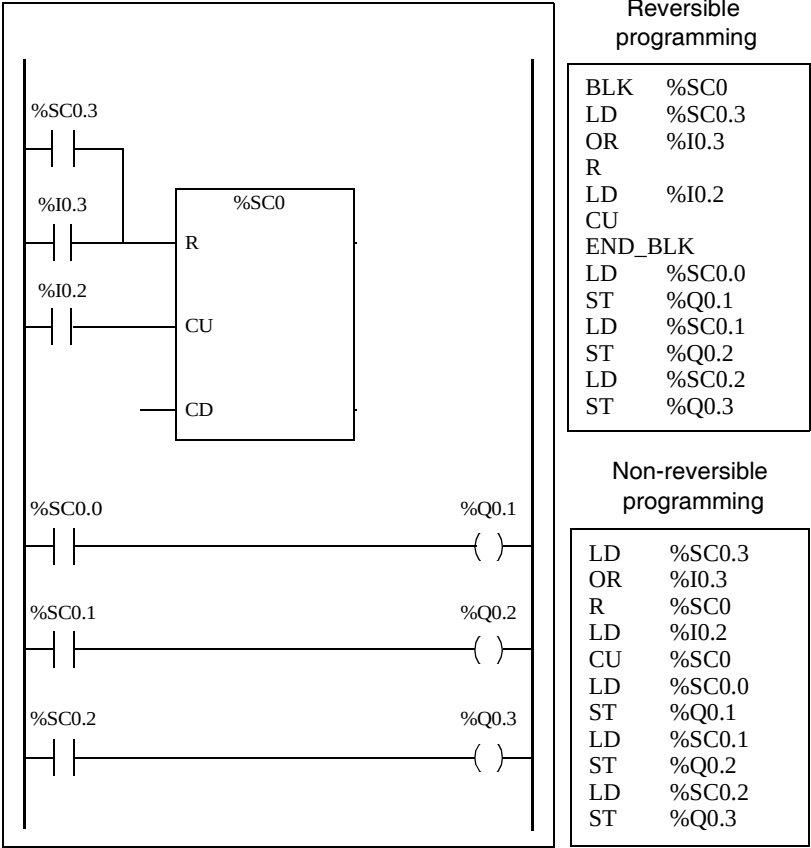


Programming

The following is an example of a Step Counter function block.

- Step Counter 0 is incremented by input %I0.2.
- Step Counter 0 is reset to 0 by input %I0.3 or when it arrives at step 3.
- Step 0 controls output %Q0.1, step 1 controls output %Q0.2, and step 2 controls output %Q0.3.

The following illustration shows both reversible and non-reversible programming for this example.



Special case

The following table contains a list of special cases for operating the Step Counter function block.

Special case	Description
Effect of a cold restart (%S0=1)	Initializes the step counter.
Effect of a warm restart (%S1=1)	Has no effect on the step counter.

16.3 Numerical Processing

At a Glance

Aim of this Section This section provides an introduction to Numerical Processing including descriptions and programming guidelines.

What's in this Section? This section contains the following topics:

Topic	Page
Introduction to Numerical Instructions	410
Assignment Instructions	411
Comparison Instructions	416
Arithmetic Instructions on Integers	418
Logic Instructions	422
Shift Instructions	423
Conversion Instructions	425
Single/double word conversion instructions	427

Introduction to Numerical Instructions

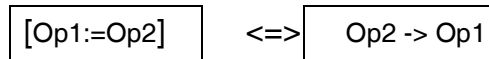
At a Glance

Numerical instructions generally apply to 16-bit words (see *Word Objects*, p. 29) and to 32-bit double words (See *Floating point and double word objects*, p. 32). They are written between square brackets. If the result of the preceding logical operation was true (Boolean accumulator = 1), the numerical instruction is executed. If the result of the preceding logical operation was false (Boolean accumulator = 0), the numerical instruction is not executed and the operand remains unchanged.

Assignment Instructions

Introduction Assignment instructions are used to load operand Op2 into operand Op1.

Assignment Syntax for Assignment instructions.



Assignment operations can be performed on:

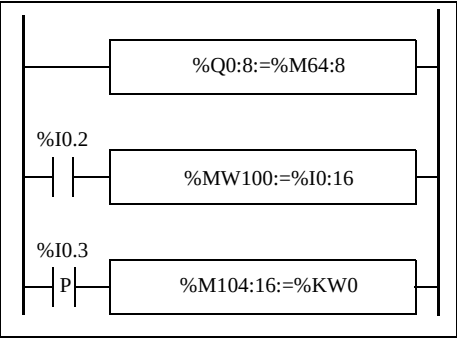
- Bit strings
- Words
- Double words
- Floating word
- Word tables
- Double word tables
- Floating word tables

Assignment of Bit Strings Operations can be performed on the following bit strings (see *Structured Objects*, p. 45):

- Bit string -> bit string (Example 1)
- Bit string -> word (Example 2) or double word (indexed)
- Word or double word (indexed) -> bit string (Example 3)
- Immediate value -> bit string

Examples

Examples of bit string assignments.



LD 1
[%Q0:8:=%M64:8]

(Ex. 1)

LD %I0.2
[%MW100:=%I0:16]

(Ex. 2)

LDR %I0.3
[%M104:16:=%KW0]

(Ex. 3)

Usage rules:

- For bit string -> word assignment: The bits in the string are transferred to the word starting on the right (first bit in the string to bit 0 in the word), and the word bits which are not involved in the transfer (length ≤16) are set to 0.
- For word -> bit string assignment: The word bits are transferred from the right (word bit 0 to the first bit in the string).

Bit String Assignments

Syntax for bit string assignments.

Operator	Syntax	Operand 1 (Op1)	Operand 2 (Op2)
:=	[Op1: = Op2] Operand 1 (Op1) assumes the value of operand 2 (Op2)	%MWi,%QWi, %QWAi,%SWi %MWi[%MWi], %MDi, %MDi[%MWi] %Mi:L, %Qi:L, %Si:L, %Xi:L	Immediate value, %MWi, %KW _i , %IW,%IWA _i , %INW _i , %QWi, %QWA _i %QNW _i , %SW _i , %BLK.x, %MWi[%MWi], %KW _i [%MWi], %MDi[%MWi], %KDi[%MWi], %Mi:L,%Qi:L, %Si:L, %Xi:L, %li:L

Note: The abbreviation %BLK.x (for example, %C0.P) is used to describe any function block word.

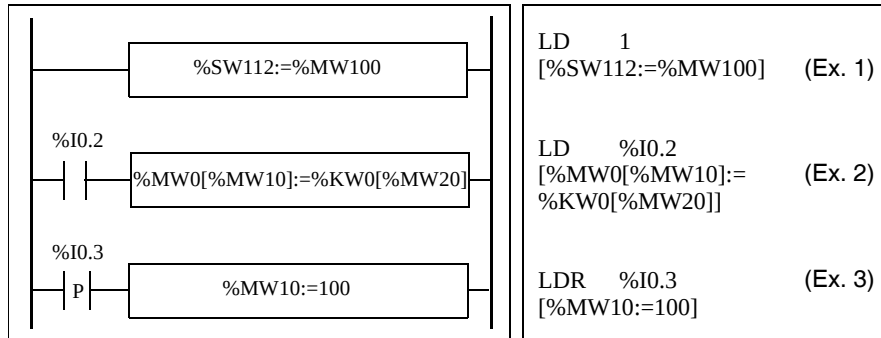
Assignment of Words

Assignment operations can be performed on the following words and double words:

- Word (indexed) -> word (2, for example) (indexed or not)
- Double word (indexed) -> double word (indexed or not)
- Immediate whole value -> word (Example 3) or double word (indexed or not)
- Bit string -> word or double word
- Floating point (indexed or not)-> floating point (indexed or not)
- Word or double word -> bit string
- Immediate floating point value -> floating point (indexed or not)

Examples

Examples of word assignments.



Syntax

Syntax for word assignments.

Operator	Syntax
:=	[Op1: = Op2] Operand 1 (Op1) assumes the value of operand 2 (Op2)

The following table gives details operands:

Type	Operand 1 (Op1)	Operand 2 (Op2)
word, double word, bit string	%BLK.x, %MWi, %QWi, %QWAI, %SWi %MWi[MWi, %MDi, %MDi[%MWj]], %Mi:L, %Qi:L, %Si:L, %Xi:L	Immediate value, %MWi, %KWi, %IW, %IWAi, %QWi, %QWAI, %SWi, %MWi[MWi], %KWi[MWi], %MDi, %MDi[%MWj], %KDi, %KDi[MWj] , %INW, %Mi:L, %Qi:L, %QNW, %Si:L, %Xi:L, %Ii:L
Floating point	%MFi, %MFi[%MWj]	Immediate floating point value, %MFi, %MFi[%MWj], %KFi, %KFi[%MWj]

Note: The abbreviation %BLK.x (for example, R3.I) is used to describe any function block word. For bit strings %Mi:L, %Si:L, and %Xi:L, the base address of the first of the bit string must be a multiple of 8 (0, 8, 16, ..., 96, ...).

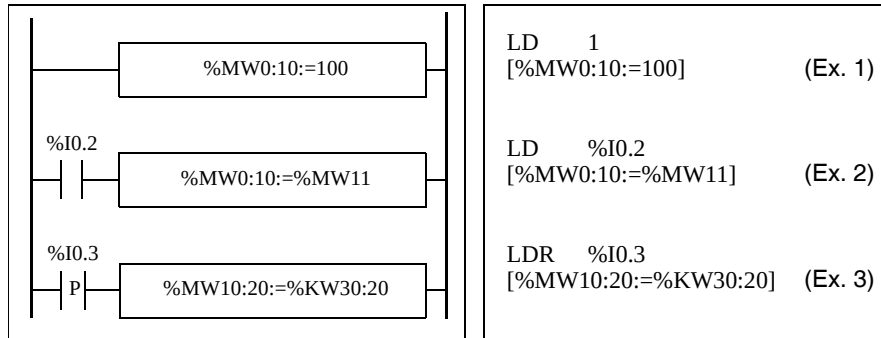
Assignment of Word, Double Word and Floating Point Tables

Assignment operations can be performed on the following object tables (see *Tables of words*, p. 46):

- Immediate whole value -> word table (Example 1) or double word table
- Word -> word table (Example 2)
- Word table -> word table (Example 3)
Table length (L) should be the same for both tables.
- Double word -> double word table
- Double word table -> double word table
Table length (L) should be the same for both tables.
- Immediate floating point value -> floating point table
- Floating point -> floating point table
- Floating point table-> floating point table
Table length (L) should be the same for both tables.

Examples

Examples of word table assignments:



Syntax

Syntax for word, double word and floating point table assignments

Operator	Syntax
:=	[Op1: = Op2] Operand 1 (Op1) assumes the value of operand 2 (Op2)

The following table gives details operands:

Type	Operand 1 (Op1)	Operand 2 (Op2)
word table	%MWi:L, %SWi:L	%MWi:L, %SWi:L, Immediate whole value, %MWi, %KW, %IW, %QW, %IWA, %QWA, %SWi, %BLK.x
Double word tables	%MDi:L	Immediate whole value, %MDi, %KDi,%MDi:L, %KDi:L
Floating word tables	%MFi:L]	Immediate floating point value, %MFi, %KFi, %MFi:L, %KFi:L

Note: The abbreviation %BLK.x (for example, R3.I) is used to describe any function block word.

Comparison Instructions

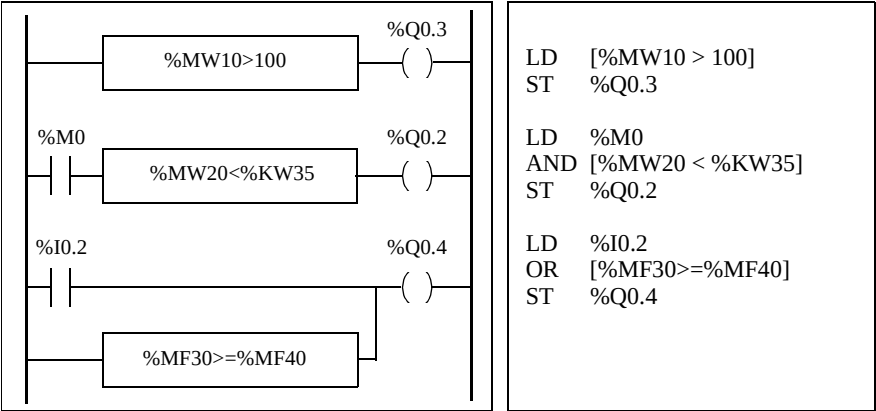
Introduction

Comparison instructions are used to compare two operands.
The following table lists the types of Comparison instructions.

Instruction	Function
>	Test if operand 1 is greater than operand 2
>=	Test if operand 1 is greater than or equal to operand 2
<	Test if operand 1 is less than operand 2
<=	Test if operand 1 is less than or equal to operand 2
=	Test if operand 1 is equal than operand 2
<>	Test if operand 1 is different from operand 2

Structure

The comparison is executed inside square brackets following instructions LD, AND, and OR. The result is 1 when the comparison requested is true.
Examples of Comparison instructions.



Syntax

Syntax for Comparison instructions:

Operator	Syntax
>, >=, <, <=, =, <>	LD [Op1 Operator Op2] AND [Op1 Operator Op2] OR [Op1 Operator Op2]

Operands:

Type	Operand 1 (Op1)	Operand 2 (Op2)
Words	%MWi, %KWi, %INWi, %IW, %IWAi, %QNW _i , %QWi, %QW _{Ai} , %QNW _i , %SW _i , %BLK.x	Immediate value, %MW _i , %KWi, %INWi, %IW, %IWAi, %QNW _i , %QW, %QW _{Ai} , %SW _i , %BLK.x, %MW _i [%MW _i], %KWi [%MW _i]
Double words	%MDi, %KDi	Immediate value, %MDi, %KDi, %MDi [%MW _i], %KD [%MW _i]
Floating word	%MFi, %KFi	Immediate floating point value, %MFi, %KFi, %MFi [%MW _i], %KFi [%MW _i]

Note: Comparison instructions can be used within parentheses.

An example of using Comparison instruction within parentheses:

```
LD      %M0
AND(    [%MF20 > 10.0]
OR      %I0.0
)
ST      %Q0.1
```

Arithmetic Instructions on Integers

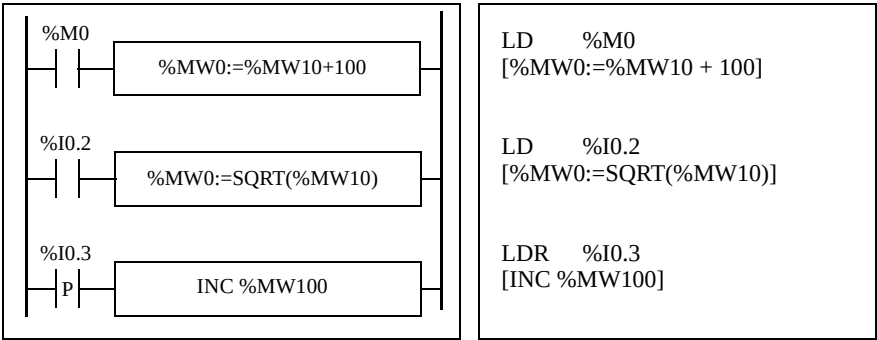
Introduction

Arithmetic instructions are used to perform arithmetic operations between two integer operands or on one integer operand.
The following table lists the types of Arithmetic instructions.

Instruction	Function
+	Add two operands
-	Subtract two operands
*	Multiply two operands
/	Divide two operands
REM	Remainder of division of the two operands
SQRT	Square root of an operand
INC	Increment an operand
DEC	Decrement an operand
ABS	Absolute value of an operand

Structure

Arithmetic operations are performed as follows:



Syntax

The syntax depends on the operators used as shown in the table below.

Operator	Syntax
+, -, *, /, REM	[Op1: = Op 2 Operator Op3]
INC, DEC	[Operator Op1]
SQRT (1)	[Op1: = SQRT(Op2)]
ABS (1)	[Op1: = ABS(Op2)]

Operands:

Type	Operand 1 (Op1)	Operands 2 and 3 (Op2 & 3) (1)
Words	%MWi, %QWi, %QWai, %SWi	Immediate value, %MWi, %KWi, %INW, %IW, %IWAi, %QNW, %QW, %QWai, %SWi, %BLK.x
Double words	%MDi	Immediate value, %MDi, %KDi

Note: (1) With this operator, Op2 cannot be an immediate value. The ABS function can only be used with double words (%MD and %KD) and floating points (%MF and %KF). Consequently, OP1 and OP2 must be double words or floating points.

Overflow and Error Conditions

Addition

- Overflow during word operation

If the result exceeds the capacity of the result word, bit %S18 (overflow) is set to 1 and the result is not significant (see Example 1, next page). The user program manages bit %S18.

Note:

For double words, the limits are -2147483648 and 2147483648.

Multiplication

- Overflow during operation

If the result exceeds the capacity of the result word, bit %S18 (overflow) is set to 1 and the result is not significant.

Division / remainder

- Division by 0

If the divisor is 0, the division is impossible and system bit %S18 is set to 1. The result is then incorrect.

- Overflow during operation

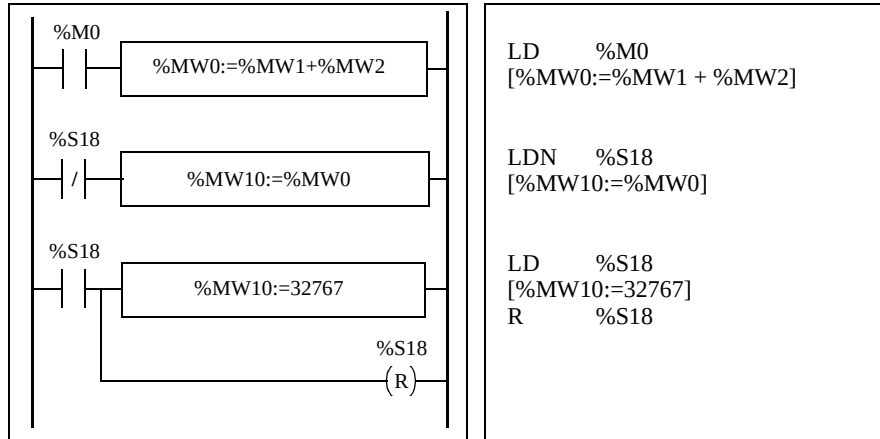
If the division quotient exceeds the capacity of the result word, bit %S18 is set to 1.

Square root extraction

- Overflow during operation

Square root extraction is only performed on positive values. Thus, the result is always positive. If the square root operand is negative, system bit %S18 is set to 1 and the result is incorrect.

Note: The user program is responsible for managing system bits %S17 and %S18. These are set to 1 by the controller and must be reset by the program so that they can be reused (see previous page for example).

Examples**Example 1: overflow during addition.**

If %MW1 =23241 and %MW2=21853, the real result (45094) cannot be expressed in one 16-bit word, bit %S18 is set to 1 and the result obtained (-20442) is incorrect. In this example when the result is greater than 32767, its value is fixed at 32767.

Logic Instructions

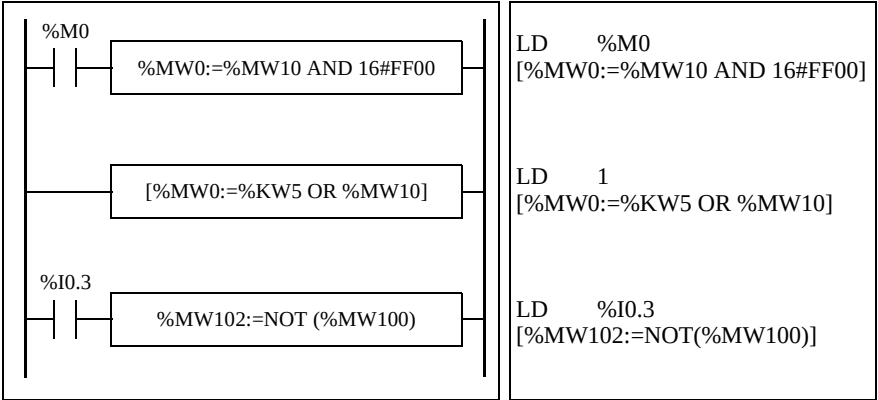
Introduction

The Logic instructions are used to perform a logical operation between two word operands or on one word operand.
The following table lists the types of Logic instructions.

Instruction	Function
AND	AND (bit-wise) between two operands
OR	Logic OR (bit-wise) between two operands
XOR	Exclusive OR (bit-wise) between two operands
NOT	Logic complement (bit-wise) of an operand

Structure

Logic operations are performed as follows:



Syntax

The syntax depends on the operators used:

Operator	Syntax	Operand 1 (Op1)	Operands 2 and 3 (Op2 & 3)
AND, OR, XOR	[Op1: = Op2 Operator Op3]	%MWi, %QWi, %QWai, %SWi	Immediate value (1), %MWi, %KWi, %IW, %IWai, %QW, %QWai, %SWi, %BLK.x
NOT	[Op1:=NOT(Op2)]		

Note: (1) With NOT, Op2 cannot be an immediate value.

Example

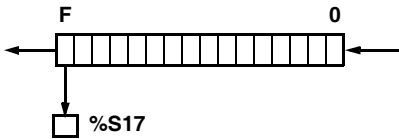
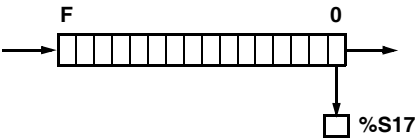
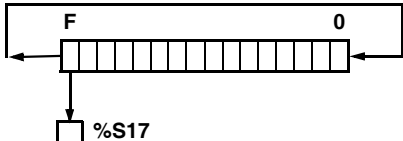
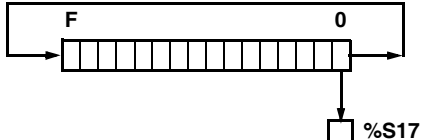
The following is an example of a logical AND instruction:
[%MW15 := %MW32 AND %MW12]

Shift Instructions

Introduction

Shift instructions move bits of an operand a certain number of positions to the right or to the left.

The following table lists the types of Shift instructions.

Instruction	Function	
Logic shift		
SHL(op2,i)	Logic shift of i positions to the left.	
SHR(op2,i)	Logic shift of i positions to the right.	
Rotate shift		
ROR(op2,i)	Rotate shift of i positions to the left.	
ROL(op2,i)	Rotate shift of i positions to the right.	

Note: System bit %S17 (See *System Bits (%S)*, p. 596) is used for capacity overrun.

Structure

Shift operations are performed as follows:

%I0.1

P

%MW0:=SHL(%MW10, 5)

%I0.2

P

%MW10:=ROR(%KW9, 8)

LDR %I0.1
[%MW0 :=SHL(%MW10, 5)]

LDR %I0.2
[%MW10 :=ROR(%KW9, 8)]

Syntax

The syntax depends on the operators used as shown in the table below.

Operator	Syntax
SHL, SHR	[Op1: = Operator (Op2,i)]
ROL, ROR	

Operands:

Types	Operand 1 (Op1)	Operand 2 (Op2)
Words	%MWi, %QWi, %QWai, %SWi	%MWi, %KWi, %IW, %IWAi, %QW, %QWai, %SWi, %BLK.x
Double word	%MDi	%MDi, %KDi

Conversion Instructions

Introduction

Conversion instructions perform conversion between different representations of numbers.

The following table lists the types of Conversion instructions.

Instruction	Function
BTI	BCD --> Binary conversion
ITB	Binary --> BCD conversion

Review of BCD Code

Binary Coded Decimal (BCD) represents a decimal digit (0 to 9) by coding four binary bits. A 16-bit word object can thus contain a number expressed in four digits (0000 - 9999), and a 32 bit double word object can therefore contain an eight-figure number.

During conversion, system bit %S18 is set to 1 if the value is not BCD. This bit must be tested and reset to 0 by the program.

BCD representation of decimal numbers:

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Examples:

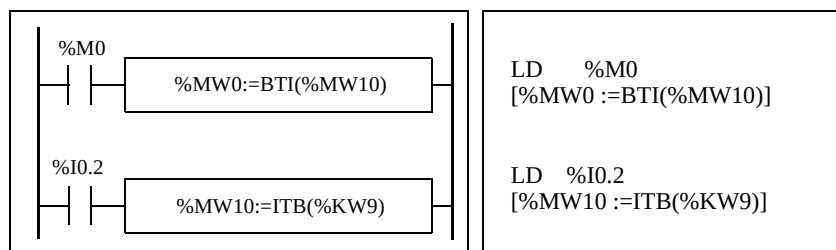
- Word %MW5 expresses the BCD value "2450" which corresponds to the binary value: 0010 0100 0101 0000
- Word %MW12 expresses the decimal value "2450" which corresponds to the binary value: 0000 1001 1001 0010

Word %MW5 is converted to word %MW12 by using instruction BTI.

Word %MW12 is converted to word %MW5 by using instruction ITB.

Structure

Conversion operations are performed as follows:



Syntax

The syntax depends on the operators used as shown in the table below.

Operator	Syntax
BTI, ITB	[Op1: = Operator (Op2)]

Operands:

Type	Operand 1 (Op1)	Operand 2 (Op2)
Words	%MWi, %QWi, %QWAI, %SWi	%MWi, %KW, %IW, %IWAi, %QW, %QWAI, %SWi, %BLK.x
Double word	%MDi	%MDi, %KDi

**Application
Example:**

The BTI instruction is used to process a setpoint value at controller inputs via BCD encoded thumb wheels.

The ITB instruction is used to display numerical values (for example, the result of a calculation, the current value of a function block) on BCD coded displays.

Single/double word conversion instructions

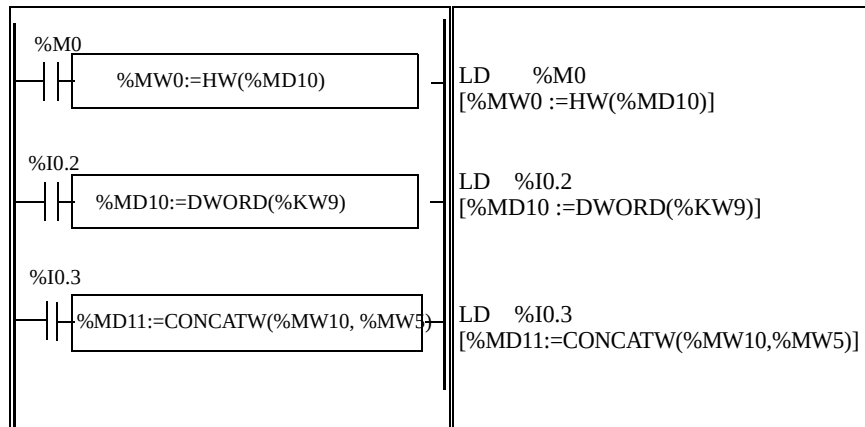
Introduction

The following table describes instructions used to perform conversions between single and double words:

Instruction	Function
LW	LSB of double word extracted to a word.
HW	MSB of double word extracted to a word.
CONCATW	Concatenates two words into a double word.
DWORD	Converts a 16 bit word into a 32 bit double word.

Structure

Conversion operations are performed as follows:



Syntax

The syntax depends on the operators used as shown in the following table: I

Operator	Syntax	Operand 1 (Op1)	Operand 2 (Op2)	Operand 3 (Op3)
LW, HW	Op1 = Operator (Op2)	%MWi	%MDi, %KDi	[-]
CONCATW	Op1 = Operator (Op2, Op3))	%MDi	%MWi, %KW _i , immediate value	%MW _i , %KW _i , immediate value
DWORD	Op1 = Operator (Op2)	%MDi	%MWi, %KW _i	[-]

16.4 Program Instructions

At a Glance

Aim of this Section This section provides an introduction to Program Instructions.

What's in this Section? This section contains the following topics:

Topic	Page
END Instructions	429
NOP Instruction	431
Jump Instructions	432
Subroutine Instructions	433

END Instructions

Introduction

The End instructions define the end of the execution of a program scan.

END, ENDC, and ENDCN

Three different end instructions are available:

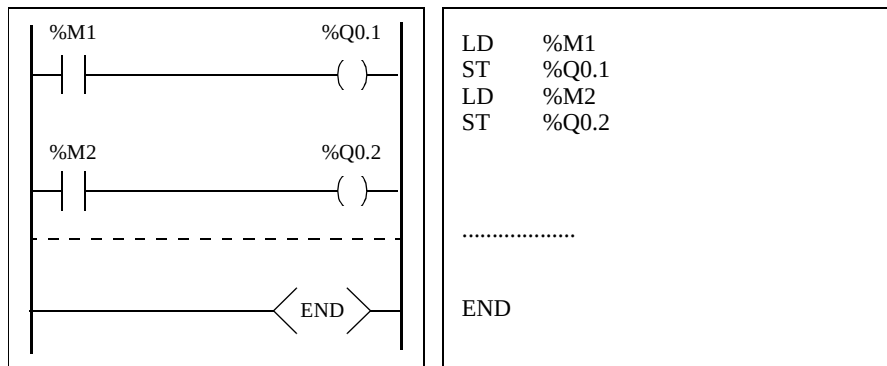
- END: unconditional end of program
- ENDC: end of program if Boolean result of preceding test instruction is 1
- ENDCN: end of program if Boolean result of preceding test instruction is 0

By default (normal mode) when the end of program is activated, the outputs are updated and the next scan is started.

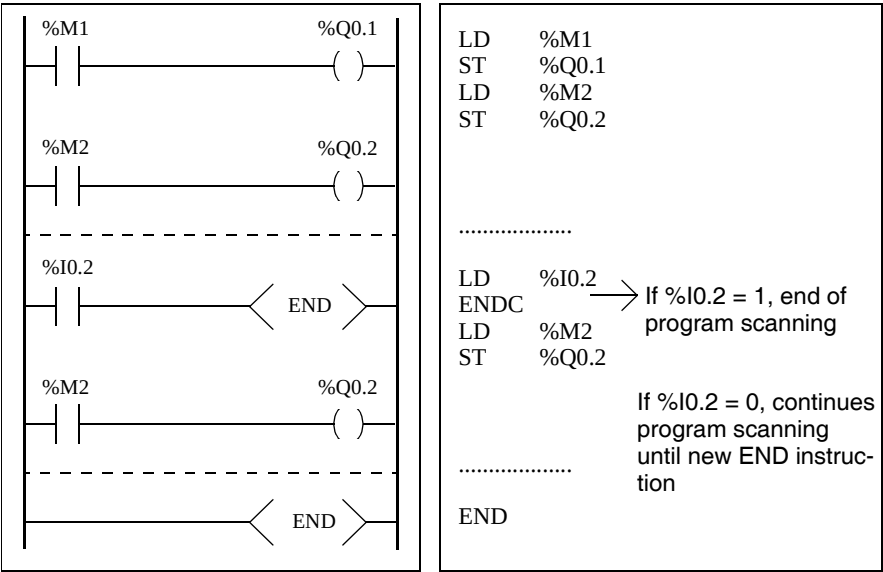
If scanning is periodic, when the end of period is reached the outputs are updated and the next scan is started.

Examples

Example of an unconditional END instruction.



Example of a conditional END instruction.



NOP Instruction

NOP

The NOP instruction does not perform any operation. Use it to "reserve" lines in a program so that you can insert instructions later without modifying the line numbers.

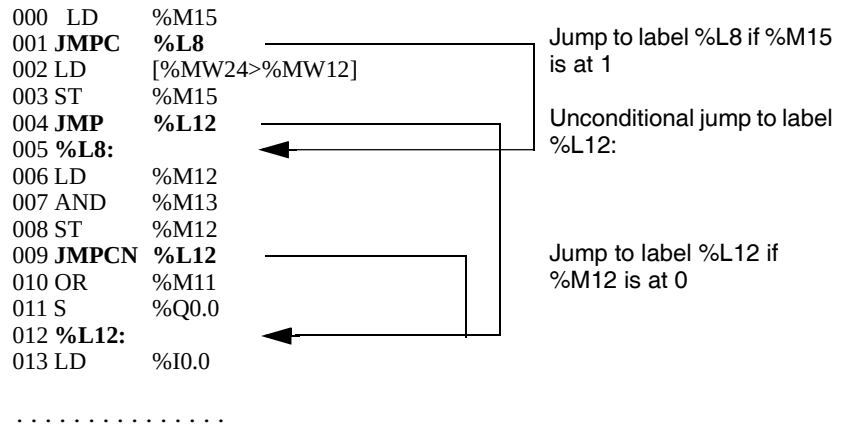
Jump Instructions

Introduction Jump instructions cause the execution of a program to be interrupted immediately and to be continued from the line after the program line containing label %Li (i = 1 to 16 for a compact and 1 to 63 for the others).

JMP, JMPC and JMPCN Three different Jump instructions are available:

- JMP: unconditional program jump
- JMPC: program jump if Boolean result of preceding logic is 1
- JMPCN: program jump if Boolean result of preceding logic is 0

Examples Examples of jump instructions.



- Guidelines**
- Jump instructions are not permitted between parentheses, and must not be placed between the instructions AND(, OR(and a close parenthesis instruction ")".
 - The label can only be placed before a LD, LDN, LDR, LDF or BLK instruction.
 - The label number of label %Li must be defined only once in a program.
 - The program jump is performed to a line of programming which is downstream or upstream. When the jump is upstream, attention must be paid to the program scan time. Extended scan time can cause triggering of the watchdog.

Subroutine Instructions

Introduction

The Subroutine instructions cause a program to perform a subroutine and then return to the main program.

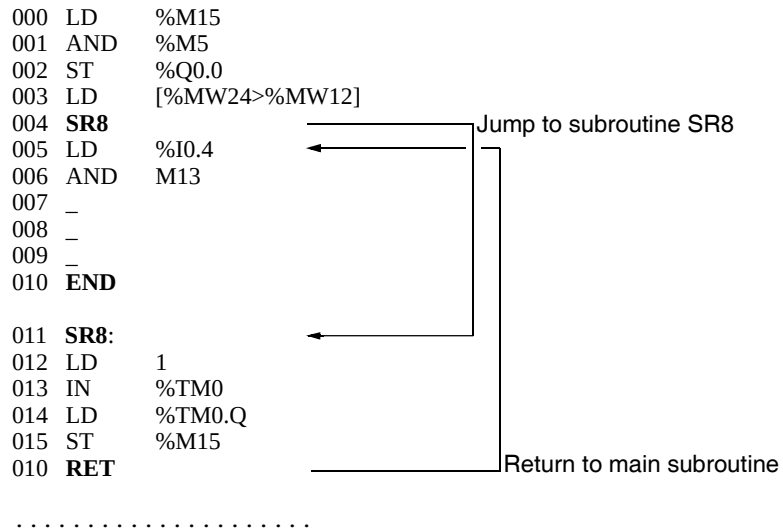
SRn, SRn: and RET.

The subroutines consist of three steps:

- The **SRn** instruction calls the subroutine referenced by label SRn, if the result of the preceding Boolean instruction is 1.
- The subroutine is referenced by a label **SRn:**, with n = 0 to 15 for TWDLCAA10DRF, TWDLCAA16DRF and 0 to 63 for all other controllers.
- The **RET** instruction placed at the end of the subroutine returns program flow to the main program.

Example

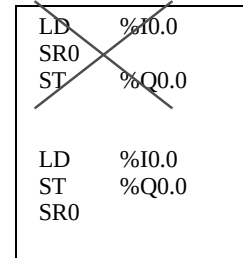
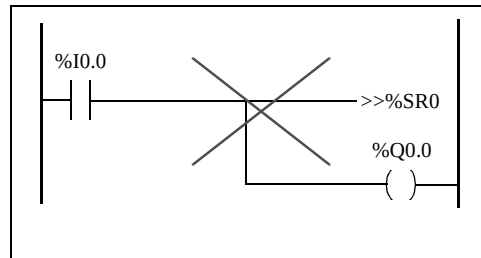
Examples of subroutine instructions.



Guidelines

- A subroutine should not call up another subroutine.
- Subroutine instructions are not permitted between parentheses, and must not be placed between the instructions AND(, OR(and a close parenthesis instruction ")".
- The label can only be placed before a LD or BLK instruction marking the start of a Boolean equation (or rung).
- Calling the subroutine should not be followed by an assignment instruction. This is because the subroutine may change the content of the boolean accumulator. Therefore upon return, it could have a different value than before the call. See the following example.

Example of programming a subroutine.



At a Glance

Subject of this Chapter

This chapter provides details about instructions and function blocks that are used to create advanced control programs for Twido programmable controllers.

What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
17.1	Advanced Function Blocks	437
17.2	Clock Functions	480
17.3	Twido PID Quick Start Guide	490
17.4	PID Function	516
17.5	Floating point instructions	567
17.6	Instructions on Object Tables	578

17.1 Advanced Function Blocks

At a Glance

Aim of this Section

This section provides an introduction to advanced function blocks including programming examples.

What's in this Section?

This section contains the following topics:

Topic	Page
Bit and Word Objects Associated with Advanced Function Blocks	438
Programming Principles for Advanced Function Blocks	440
LIFO/FIFO Register Function Block (%Ri)	443
LIFO Operation	444
FIFO,operation	445
Programming and Configuring Registers	446
Pulse Width Modulation Function Block (%PWM)	448
Pulse Generator Output Function Block (%PLS)	451
Drum Controller Function Block (%DR)	454
Drum Controller Function Block %DRi Operation	455
Programming and Configuring Drum Controllers	457
Fast Counter Function Block (%FC)	459
Very Fast Counter Function Block (%VFC)	462
Transmitting/Receiving Messages - the Exchange Instruction (EXCH)	476
Exchange Control Function Block (%MSGx)	477

Bit and Word Objects Associated with Advanced Function Blocks

Introduction

Advanced function blocks use similar types of dedicated words and bits as the standard function blocks. Advanced function blocks include:

- LIFO/FIFO registers (%R)
- Drum controllers (%DR)
- Fast counters (%FC)
- Very fast counters (%VFC)
- Pulse width modulation output (%PWM)
- Pulse generator output (%PLS)
- Shift Bit Register (%SBR)
- Step counter (%SC)
- Message control block (%MSG)

Objects Accessible by the Program

The table below contains an overview of the words and bits accessible by the program that are associated with the various advanced function blocks. Please note that write access in the table below depends on the "Adjustable" setting selected during configuration. Setting this allows or denies access to the words or bits by TwidoSoft or the operator interface.

Advanced Function Block	Associated Words and Bits		Address	Write Access
%R	Word	Register input	%Ri.I	Yes
	Word	Register output	%Ri.O	Yes
	Bit	Register output full	%Ri.F	No
	Bit	Register output empty	%Ri.E	No
%DR	Word	Current step number	%DRi.S	Yes
	Bit	Last step equals current step	%DRi.F	No
%FC	Word	Current Value	%FCi.V	Yes
	Word	Preset value	%FCi.P	Yes
	Bit	Done	%FCi.D	No

Advanced Function Block	Associated Words and Bits		Address	Write Access
%VFC	Word	Current Value	%VFCi.V	No
	Word	Preset value	%VFCi.P	Yes
	Bit	Counting direction	%VFCi.U	No
	Word	Capture Value	%VFCi.C	No
	Word	Threshold 0 Value	%VFCi.S0	Yes
	Word	Threshold Value1	%VFCi.S1	Yes
	Bit	Overflow	%VFCi.F	No
	Bit	Reflex Output 0 Enable	%VFCi.R	Yes
	Bit	Reflex Output 1 Enable	%VFCi.S	Yes
	Bit	Threshold Output 0	%VFCi.TH0	No
	Bit	Threshold Output 1	%VFCi.TH1	No
	Bit	Frequency Measure Time Base	%VFCi.T	Yes
%PWM	Word	Percentage of pulse at 1 in relationship to the total period.	%PWMi.R	Yes
	Word	Preset period	%PWMi.P	Yes
%PLS	Word	Number of pulses	%PLSi.N	Yes
	Word	Preset value	%PLSi.P	Yes
	Bit	Current output enabled	%PLSi.Q	No
	Bit	Generation done	%PLSi.D	No
%SBR	Bit	Register Bit	%SBRi.J	No
%SC	Bit	Step counter Bit	%SCi.j	Yes
%MSG	Bit	Done	%MSGi.D	No
	Bit	Error	%MSGi.E	No

Programming Principles for Advanced Function Blocks

At a Glance

All Twido applications are stored in the form of List programs, even if written in the Ladder Editor, and therefore, Twido controllers can be called List "machines." The term "reversibility" refers to the ability of TwidoSoft to represent a List application as Ladder and then back again. By default, all Ladder programs are reversible.

As with basic function blocks, advanced function blocks must also take into consideration reversibility rules. The structure of reversible function blocks in List language requires the use of the following instructions:

- **BLK**: Marks the block start and the input portion of the function block
- **OUT_BLK**: Marks the beginning of the output portion of the function block
- **END_BLK**: Marks the end of the function block

Note: The use of these reversible function block instructions is not mandatory for a properly functioning List program. For some instructions it is possible to program in List language without being reversible.

Dedicated Inputs and Outputs

The Fast Counter, Very Fast Counter, PLS, and PWM advanced functions use dedicated inputs and outputs, but these bits are not reserved for exclusive use by any single block. Rather, the use of these dedicated resources must be managed. When using these advanced functions, you must manage how the dedicated inputs and outputs are allocated. TwidoSoft assists in configuring these resources by displaying input/output configuration details and warning if a dedicated input or output is already used by a configured function block.

The following tables summarizes the dependencies of dedicated inputs and outputs and specific functions.

When used with counting functions:

Inputs	Use
%I0.0.0	%VFC0: Up/Down management or Phase B
%I0.0.1	%VFC0: Pulse input or Phase A
%I0.0.2	%FC0: Pulse input or %VFC0 pre-set input
%I0.0.3	%FC1: Pulse input or %VFC0 capture input
%I0.0.4	%FC2: Pulse input or %VFC1 capture input
%I0.0.5	%VFC1 pre-set input
%I0.0.6	%VFC1: Up/Down management or Phase B
%I0.0.7	%VFC1: Pulse input or Phase A

When used with counting or special functions:

Outputs	Use
%Q0.0.0	%PLS0 or PWM0 output
%Q0.0.1	%PLS1 or PWM1 output
%Q0.0.2	Reflex outputs for %VFC0
%Q0.0.3	
%Q0.0.4	Reflex outputs for %VFC1
%Q0.0.5	

Using Dedicated Inputs and Outputs

TwidoSoft enforces the following rules for using dedicated inputs and outputs.

- Each function block that uses dedicated I/O must be configured and then referenced in the application. The dedicated I/O is only allocated when a function block is configured and not when it is referenced in a program.
- After a function block is configured, its dedicated input and output cannot be used by the application or by another function block.
For example, if you configure %PLS0, you can not use %Q0.0.0 in %DR0 (drum controller) or in the application logic (that is, ST %Q0.0.0).
- If a dedicated input or output is needed by a function block that is already in use by the application or another function block, this function block cannot be configured.
For example, if you configure %FC0 as an up counter, you can not configure %VFC0 to use %I0.0.2 as capture input.

Note: To change the use of dedicated I/O, unconfigure the function block by setting the type of the object to "not used," and then remove references to the function block in your application.

LIFO/FIFO Register Function Block (%Ri)

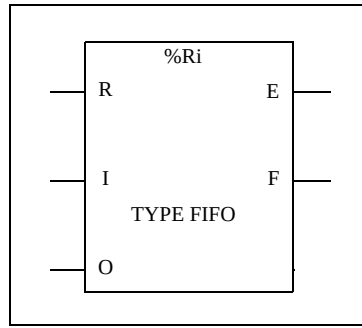
Introduction

A register is a memory block which can store up to 16 words of 16 bits each in two different ways:

- Queue (First In, First Out) known as FIFO.
- Stack (Last In, First Out) known as LIFO.

Illustration

The following is an illustration of the register function block.



Register function block

Parameters

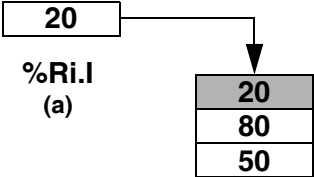
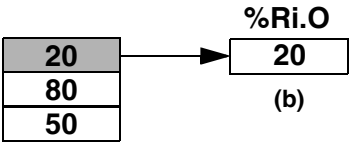
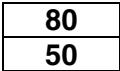
The Counter function block has the following parameters:

Parameter	Label	Value
Register number	%Ri	0 to 3.
Type	FIFO or LIFO	Queue or Stack.
Input word	%Ri.I	Register input word. Can be read, tested, and written.
Output word	%Ri.O	Register output word. Can be read, tested and written.
Storage Input (or instruction)	I (In)	On a rising edge, stores the contents of word %Ri.I in the register.
Retrieval Input (or instruction)	O (Out)	On a rising edge, loads a data word of the register into word %Ri.O.
Reset input (or instruction)	R (Reset)	At state 1, initializes the register.
Empty Output	E (Empty)	The associated bit %Ri.E indicates that the register is empty. Can be tested.
Full Output	F (Full)	The associated bit %Ri.F indicates that the register is full. Can be tested.

LIFO Operation

Introduction In LIFO operation (Last In, First Out), the last data item entered is the first to be retrieved.

Operation The following table describes LIFO operation.

Step	Description	Example
1	When a storage request is received (rising edge at input I or activation of instruction I), the contents of input word %Ri.I (which has already been loaded) are stored at the top of the stack (Fig. a). When the stack is full (output F=1), no further storage is possible.	Storage of the contents of %Ri.I at the top of the stack.  %Ri.I (a)
2	When a retrieval request is received (rising edge at input O or activation of instruction O), the highest data word (last word to be entered) is loaded into word %Ri.O (Fig. b). When the register is empty (output E=1) no further retrieval is possible. Output word %Ri.O does not change and retains its value.	Retrieval of the data word highest in the stack.  %Ri.O (b)
3	The stack can be reset at any time (state 1 at input R or activation of instruction R). The element indicated by the pointer is then the highest in the stack.	

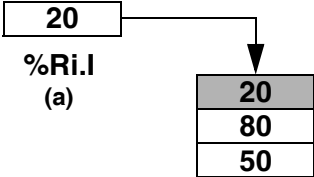
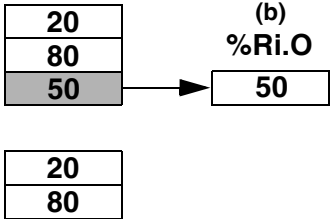
FIFO,operation

Introduction

In FIFO operation (First In, First Out), the first data item entered is the first to be retrieved.

Operation

The following table describes FIFO operation.

Step	Description	Example
1	When a storage request is received (rising edge at input I or activation of instruction I), the contents of input word %Ri.I (which has already been loaded) are stored at the top of the queue (Fig. a). When the queue is full (output F=1), no further storage is possible.	Storage of the contents of %Ri.I at the top of the queue. 
2	When a retrieval request is received (rising edge at input O or activation of instruction O), the data word lowest in the queue is loaded into output word %Ri.O and the contents of the register are moved down one place in the queue (Fig. b). When the register is empty (output E=1) no further retrieval is possible. Output word %Ri.O does not change and retains its value.	Retrieval of the first data item which is then loaded into %Ri.O. 
3	The queue can be reset at any time (state 1 at input R or activation of instruction R).	

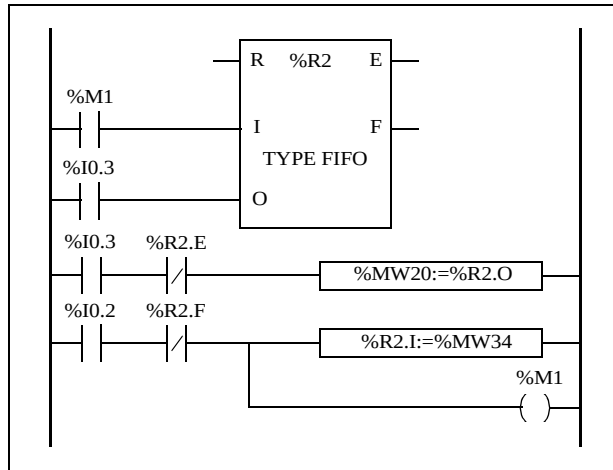
Programming and Configuring Registers

Introduction

The following programming example shows the content of a memory word (%MW34) being loaded into a register (%R2.I) on reception of a storage request (%I0.2), if register %R2 is not full (%R2.F = 0). The storage request in the register is made by %M1. The retrieval request is made by input %I0.3, and %R2.O is loaded into %MW20, if the register is not empty (%R2.E = 0).

Programming Example

The following illustration is a register function block with examples of reversible and non-reversible programming.



Ladder diagram

BLK	%R2	LD	%M1
LD	%M1	I	%R2
I		LD	%I0.3
LD	%I0.3	O	%R2
O		ANDN	%R2.E
END_BLK		[%MW20:=%R2.O]	
LD	%I0.3	LD	%I0.2
ANDN	%R2.E	ANDN	%R2.F
[%MW20:=%R2.O]		[%R2.I:=%MW34]	
LD	%I0.2	ST	%M1
ANDN	%R2.F		
[%R2.I:=%MW34]			

Reversible program

Non-reversible program

- Configuration** The only parameter that must be entered during configuration is the type of register:
- FIFO (default), or
 - LIFO

Special Cases The following table contains a list of special cases for programming the Shift Bit Register function block:

Special case	Description
Effect of a cold restart (%S0=1)	Initializes the contents of the register. The output bit %Ri.E associated with the output E is set to 1.
Effect of a warm restart (%S1=1) of a controller stop	Has no effect on the current value of the register, nor on the state of its output bits.

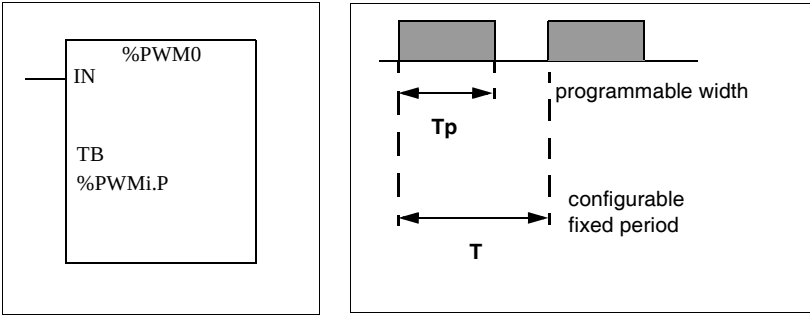
Pulse Width Modulation Function Block (%PWM)

Introduction

The Pulse Width Modulation (%PWM) function block generates a square wave signal on dedicated output channels %Q0.0.0 or %Q0.0.1, with variable width and, consequently, duty cycle. Controllers with relay outputs for these two channels do not support this function due to a frequency limitation. There are two %PWM blocks available. %PWM0 uses dedicated output %Q0.0.0 and %PMW1 uses dedicated output %Q0.0.1. The %PLS function blocks contend to use these same dedicated outputs so you must choose between the two functions.

Illustration

PWM block and timing diagram:



Parameters

The following table lists parameters for the PWM function block.

Parameter	Label	Description
Timebase	TB	0.142 ms, 0.57 ms, 10 ms, 1 s (default value)
Preselection of the period	%PWMi.P	$0 < \%PWMi.P \leq 32767$ with time base 10 ms or 1 s $0 < \%PWMi.P \leq 255$ with time base 0.57 ms or 0.142 s 0 = Function not in use
Duty cycle	%PWMi.R	This value gives the percentage of the signal in state 1 in a period. The width T_p is thus equal to: $T_p = T * (\%PWMi.R/100)$. The user application writes the value for %PWMi.R. It is this word which controls the duty cycle of the period. For T definition, see "range of periods" below. The default value is 0 and values greater than 100 are considered to be equal to 100.
Pulse generation input	IN	At state 1, the pulse width modulation signal is generated at the output channel. At state 0, the output channel is set to 0.

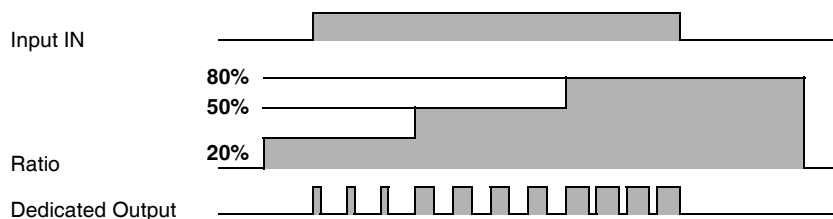
Range of Periods

The preset value and the time base can be modified during configuration. They are used to fix the signal period $T = \%PWMi.P * TB$. The lower the ratios to be obtained, the greater the selected %PWMi.P must be. The range of periods available:

- 0.142 ms to 36.5 ms in steps of 0.142 ms (27.4Hz to 7kHz)
- 0.57 ms to 146 ms in steps of 0.57 ms (6./84 Hz to 1.75 kHz)
- 10 ms to 5.45 mins in steps of 10 ms
- 1 sec to 9.1 hours in steps of 1 sec

Operation

The frequency of the output signal is set during configuration by selecting the time base TB and the preset %PWMi.P. Modifying the % PWMi.R duty cycle in the program modulates the width of the signal. Below is an illustration of a pulse diagram for the PWM function block with varying duty cycles.



Programming and Configuration

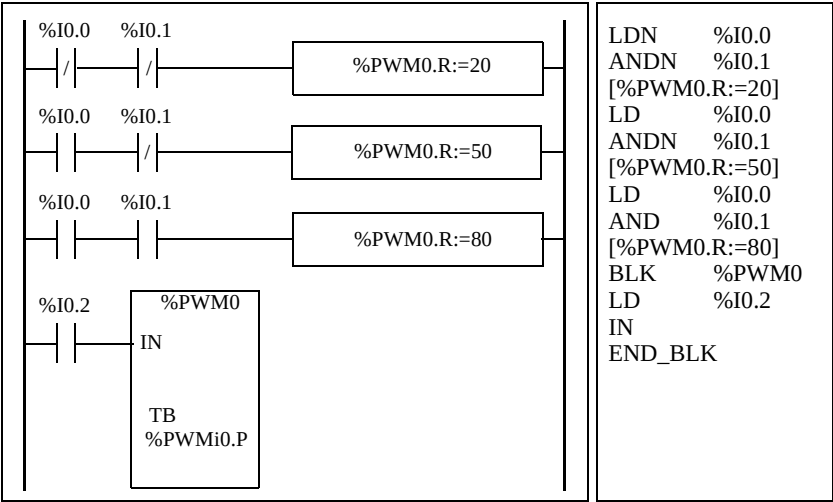
In this example, the signal width is modified by the program according to the state of controller inputs %I0.0.0 and %I0.0.1.

If %I0.0.1 and %I0.0.2 are set to 0, the %PWM0.R ratio is set at 20%, the duration of the signal at state 1 is then: 20 % x 500 ms = 100 ms.

If %I0.0.0 is set to 0 and %I0.0.1 is set to 1, the %PWM0.R ratio is set at 50% (duration 250 ms).

If %I0.0.0 and %I0.0.1 are set to 1, the %PWM0.R ratio is set at 80% (duration 400 ms).

Programming Example:



Special Cases

The following table shows a list of special operating of the PWM function block.

Special case	Description
Effect of a cold restart (%S0=1)	Sets the %PWMi.R ratio to 0. In addition, the value for %PWMi.P is reset to the configured value, and this will supersede any changes made with the Animations Table Editor or the optional Operator Display.
Effect of a warm restart (%S1=1)	Has no effect.
Effect due to the fact that outputs are dedicated to the %PWM block	Forcing output %Q0.0.0 or %Q0.0.1 using a programming device does not stop the signal generation.

Pulse Generator Output Function Block (%PLS)

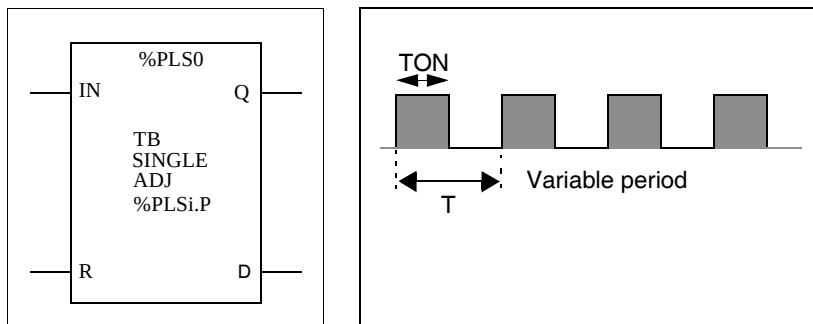
Introduction

The %PLS function block is used to generate square wave signals. There are two %PLS functions available on the dedicated output channels %Q0.0.0 or %Q0.0.1. The %PLS function block allows only a single signal width, or duty cycle, of 50%. You can choose to limit the number of pulses or the period when the pulse train is executed. These can be determined at the time of configuration and/or updated by the user application.

Note: Controllers with relay outputs for these two channels do not support %PLS function.

Representation

An example of the pulse generator function block in single-word mode:



- $TON = T/2$ for the 0.142ms and 0.57ms time bases
 $= (\%PLSi.P * TB) / 2$
- $TON = [\text{whole part}(\%PLSi.P) / 2] * TB$ for the 10ms to 1s time bases.

Specifications The table below contains the characteristics of the PLS function block:

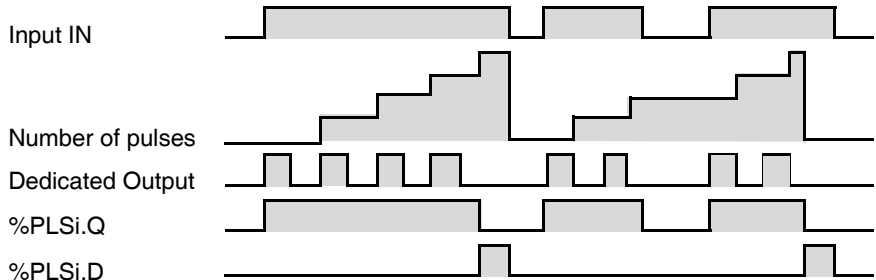
Function	Object	Description
Timebase	TB	0.142 ms, 0.57 ms, 10 ms, 1 sec
Preset period	%PLSi.P	Pulses on output %PLS1 are not stopped when %PLS1.N or %PLS1.ND* is reached for time bases 0.142 ms and 0.57 ms. • $1 < \%PLSi.P \leq 32767$ for time base 10 ms or 1 s • $0 < \%PLSi.P \leq 255$ for time base 0.57 ms or 0.142 ms • 0 = Function not in use. To obtain a good level of precision from the duty cycle with time bases of 10ms and 1s, you are recommended to have a %PLSi ≥ 100 if P is odd.
Number of pulses	%PLSi.N %PLSi.ND*	The number of pulses to be generated in period T can be limited to the range $0 \leq \%PLSi.N \leq 32767$ in standard mode or $0 \leq \%PLSi.ND \leq 4294967295$ in double word mode . The default value is set to 0. To produce an unlimited number of pulses, set %PLSi.N or %PLSi.ND to zero. The number of pulses can always be changed irrespective of the Adjustable setting.
Adjustable	Y/N	If set to Y, it is possible to modify the preset value %PLSi.P via the HMI or Animation Tables Editor. Set to N means that there is no access to the preset.
Pulse generation input	IN	At state 1, the pulse generation is produced at the dedicated output channel. At state 0, the output channel is set to 0.
Reset input	R	At state 1, outputs %PLSi.Q and %PLSi.D are set to 0. The number of pulses generated in period T is set to 0.
Current pulse output generation	%PLSi.Q	At state 1, indicates that the pulse signal is generated at the dedicated output channel configured.
Pulse generation done output	%PLSi.D	At state 1, signal generation is complete. The number of desired pulses has been reached.

Note: (*) Means a double word variable.

- Range of Periods** The preset value and the time base can be modified during configuration. They are used to fix the signal period $T = \%PLSi.P * TB$. The range of periods available:
- 0.142 ms to 36.5 ms in steps of 0.142 ms (27.4Hz to 7kHz)
 - 0.57 ms to 146 ms in steps of 0.57 ms (6.84 Hz to 1.75 kHz)
 - 20 ms to 5.45 mins in steps of 10 ms
 - 2 sec to 9.1 hours in steps of 1 sec

Operation

The following is an illustration of the %PLS function block.



Special Cases

Special case	Description
Effect of cold restart (%S0=1)	Sets the %PLSi.P to that defined during configuration
Effect of warm restart (%S1=1)	Has no effect
Effect of modifying the preset (%PLSi.P)	Takes effect immediately
Effect due to the fact that outputs are dedicated to the %PLS block	Forcing output %Q0.0.0 or %Q0.0.1 using a programming device does not stop the signal generation.

Note: %PLSx.D is set when the number of desired pulses has been reached. It is reset by either setting the IN or the R inputs to 1.

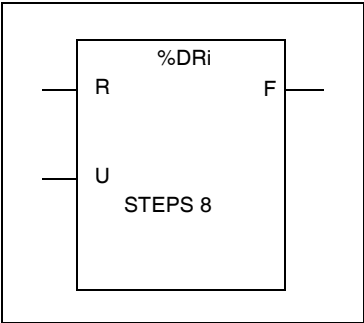
Drum Controller Function Block (%DR)

Introduction

The drum controller operates on a principle similar to an electromechanical drum controller which changes step according to external events. On each step, the high point of a cam gives a command which is executed by the controller. In the case of a drum controller, these high points are symbolized by state 1 for each step and are assigned to output bits %Qi.j or internal bits %Mi, known as control bits.

Illustration

The following is an illustration of the drum controller function block.



Drum controller function block

Parameters

The drum controller function block has the following parameters:

Parameter	Label	Value
Number	%DRi	0 to 3 Compact Controller0 to 7 Modular Controllers
Current step number	%DRi.S	0<%DRi.S<7. Word which can be read and written. Written value must be a decimal immediate value. When written, the effect takes place on the next execution of the function block.
Number of steps		1 to 8 (default)
Input to return to step 0(or instruction)	R (Reset)	At state 1, sets the drum controller to step 0.
Advance input (or instruction)	U (Upper)	On a rising edge, causes the drum controller to advance by one step and updates the control bits.
Output	F (Full)	Indicates that the current step equals the last step defined. The associated bit %DRi.F can be tested (for example, %DRi.F=1, if %DRi.S= number of steps configured - 1).
Control bits		Outputs or internal bits associated with the step (16 control bits) and defined in the Configuration Editor.

Drum Controller Function Block %DRi Operation

Introduction

The drum controller consists of:

- A matrix of constant data (the cams) organized in eight steps (0 to 7) and 16 data bits (state of the step) arranged in columns numbered 0 to F.
- A list of control bits is associated with a configured output (%Qi.j.k) or memory word (%Mi). During the current step, the control bits take on the binary states defined for this step.

The example in the following table summarizes the main characteristics of the drum controller.

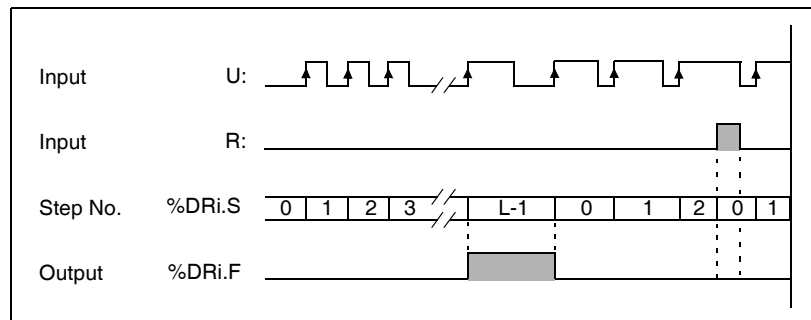
Column	0	1	2		D	O	F
Control bits	%Q0.1	%Q0.3	%Q1.5		%Q0.6	%Q0.5	%Q1.0
0 steps	0	0	1		1	1	0
1 steps	1	0	1		1	0	0
5 steps	1	1	1		0	0	0
6 steps	0	1	1		0	1	0
7 steps	1	1	1		1	0	0

Operation

In the above example, step 5 is the current step, control bits %Q0.1, %Q0.3, and %Q1.5 are set to state 1; control bits %Q0.6, %Q0.5, and %Q1.0 are set to state 0. The current step number is incremented on each rising edge at input U (or on activation of instruction U). The current step can be modified by the program.

Timing Diagram

The following diagram illustrates the operation of the drum controller.



Special Cases The following table contains a list of special cases for drum controller operation.

Special case	Description
Effects of a cold restart (%S0=1)	Resets the drum controller to step 0 (update of control bits).
Effect of a warm restart (%S1=1)	Updates the control bits after the current step.
Effect of a program jump	The fact that the drum controller is no longer scanned means the control bits are not reset.
Updating the control bits	Only occurs when there is a change of step or in the case of a warm or cold restart.

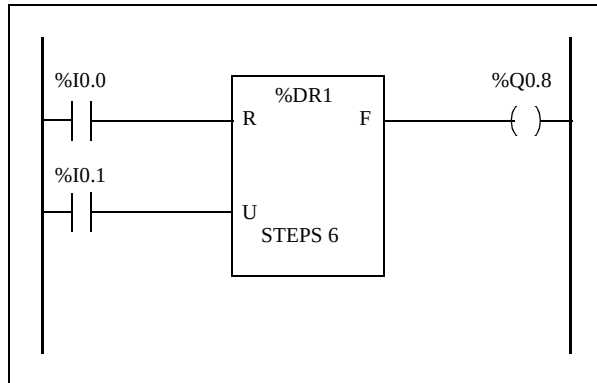
Programming and Configuring Drum Controllers

Introduction

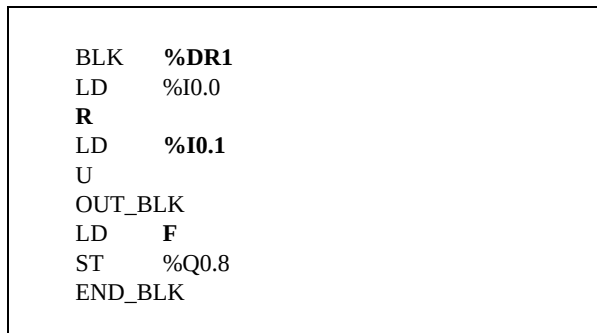
The following is an example of programming and configuring a drum controller. The first six outputs %Q0.0 to %Q0.5 are activated in succession each time input %I0.1 is set to 1. Input %I0.0 resets the outputs to 0.

Programming Example

The following illustration is a drum controller function block with examples of reversible and non-reversible programming.



Ladder diagram



Configuration

The following information is defined during configuration:

- Number of steps: 6
- The output states (control bits) for each drum controller step.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Step 1:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Step 2:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Step 3:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Step 4:	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Step 5:	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Step 6:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

- Assignment of the control bits.

1:	%Q0.0	4:	%Q0.1
2:	%Q0.2	5:	%Q0.3
3:	%Q0.4	6:	%Q0.5

Fast Counter Function Block (%FC)

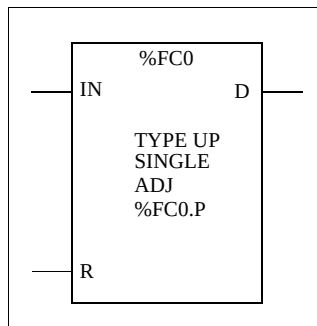
Introduction

The Fast Counter function block (%FC) serves as either an up-counter or a down-counter. It can count the rising edge of digital inputs up to frequencies of 5kHz in single word or double word computational mode. Because the Fast Counters are managed by specific hardware interrupts, maintaining maximum frequency sampling rates may vary depending on your specific application and hardware configuration.

The TWDLCA•40DRF Compact controllers can accommodate up to four fast counters, while all other series of Compact controllers can be configured to use a maximum of three fast counters. Modular controllers can only use a maximum of two. The Fast Counter function blocks %FC0, %FC1, %FC2, and %FC3 use dedicated inputs %I0.0.2, %I0.0.3, %I0.0.4 and %I0.0.5 respectively. These bits are not reserved for their exclusive use. Their allocation must be considered with the use of other function blocks for these dedicated resources.

Illustration

The following is an example of a Fast Counter function block in single-word mode.



Parameters

The following table lists parameters for the Fast Counter function block.

Parameter	Label	Description
Function	TYPE	Set at configuration, this can be set to either up-count or down-count.
Preset value	%FCi.P %FCi.PD	Initial value may be set: ->between 1 and 65535 in standard mode, ->between 1 and 4294967295 in double word mode,
Adjustable	Y/N	If set to Y, it is possible to modify the preset value %FCi.P or %FCi.PD and the current value %FCi.V or %FCi.VD with the Operator Display or Animation Tables Editor. If set to N, there is no access to the preset.
Current Value	%FCi.V %FCi.VD	The current value increments or decrements according the up or down counting function selected. For up-counting, the current counting value is updated and can reach 65535 in standard mode (%FCi.V) and 4294967295 in double word mode (%FCi.VD). For down-counting, the current value is the preset value %FCi.P or %FCi.PD and can count down to zero.
Enter to enable	IN	At state 1, the current value is updated according to the pulses applied to the physical input. At state 0, the current value is held at its last value.
Reset	%FCi.R	Used to initialize the block. At state 1, the current value is reset to 0 if configured as an up-counter, or set to %FCi.P or %FCi.PD if configured as a down-counter. The done bit %FCi.D is set back to its default value.
Done	%FCi.D	This bit is set to 1 when %FCi.V or %FCi.VD reaches the %FCi.P or %FCi.PD configured as an up-counter, or when %FCi.V or %FCi.VD reaches zero when configured as a down-counter. This read-only bit is reset only by the setting %FCi.R to 1.

Special Note

If configured to be adjustable, then the application can change the preset value %FCi.P or %FCi.PD and current value %FCi.V or %FCi.VD at any time. But, a new value is taken into account only if the input reset is active or at the rising edge of output %FCi.D. This allows for successive different counts without the loss of a single pulse.

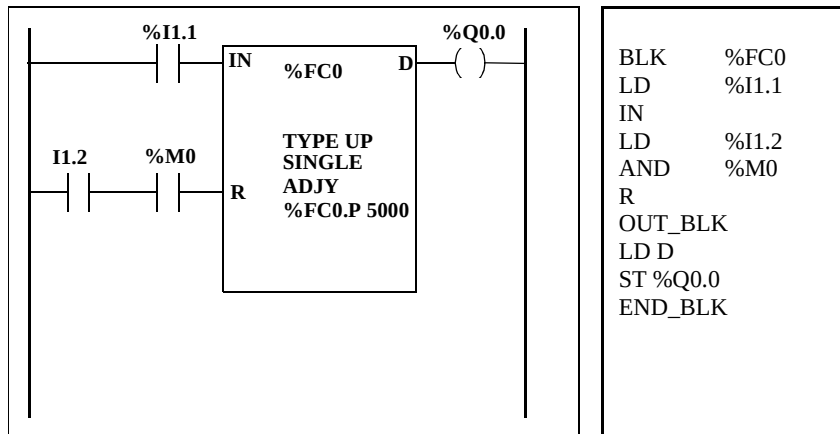
Operation

If configured to up-count, when a rising edge appears at the dedicated input, the current value is incremented by one. When the preset value %FCi.P or %FCi.PD is reached, the Done output bit %FCi.D is set to 1.

If configured to down-count, when a rising edge appears at the dedicated input, the current value is decreased by one. When the value is zero, the Done output bit %FCi.D is set to 1 and the preset value is loaded into the current value %FCi.V or %FCi.VD.

Configuration and Programming

In this example, the application counts a number of items up to 5000 while %I1.1 is set to 1. The input for %FC0 is the dedicated input %I0.0.2. When the preset value is reached, %FC0.D is set to 1 and retains the same value until %FC0.R is commanded by the result of "AND" on %I1.2 and %M0.

**Special Cases**

The following table contains a list of special operating cases for the %FC function block:

Special case	Description
Effect of cold restart (%S0=1)	Resets all the %FC attributes with the values configured by the user or user application.
Effect of warm restart (%S1=1)	Has no effect.
Effect of Controller stop	The %FC continues to count with the parameter settings enabled at the time the controller was stopped.

Very Fast Counter Function Block (%VFC)

Introduction

The Very Fast Counter function block (%VFC) can be configured by TwidoSoft to perform any one of the following functions:

- Up/down counter
- Up/down 2-phase counter
- Single Up Counter
- Single Down Counter
- Frequency Meter

The %VFC supports counting of digital input up to frequencies of 20kHz in single word or double word computational mode. The TWDLCA•40DRF Compact controllers can accommodate up to two very fast counters, while all other series of Compact controllers can configure one very fast counter (%VFC). Modular controllers can configure up to two very fast counters (%VFC).

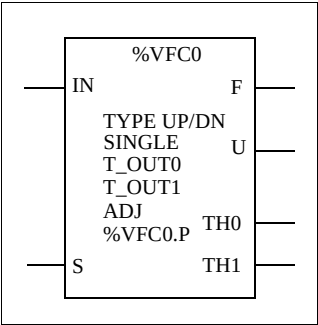
Dedicated I/O Assignments

The Very Fast Counter function blocks (%VFC) use dedicated inputs and auxiliary inputs and outputs. These inputs and outputs are not reserved for their exclusive use. Their allocation must be considered with the use of other function blocks for these dedicated resources. The following array summarizes these assignments:

		Main inputs		Auxiliary inputs		Reflex outputs	
%VFC0	Selected Use	IA input	IB input	IPres	Ica	Output 0	Output 1
	Up/down counter	%I0.0.1	%I0.0.0 (UP=0/DO=1)	%I0.0.2 (1)	%I0.0.3 (1)	%Q0.0.2 (1)	%Q0.0.3 (1)
	Up/Down 2-Phase Counter	%I0.0.1	%I0.0.0 (Pulse)	%I0.0.2 (1)	%I0.0.3 (1)	%Q0.0.2 (1)	%Q0.0.3 (1)
	Single Up Counter	%I0.0.1	(2)	%I0.0.2 (1)	%I0.0.3 (1)	%Q0.0.2 (1)	%Q0.0.3 (1)
	Single Down Counter	%I0.0.1	(2)	%I0.0.2 (1)	%I0.0.3 (1)	%Q0.0.2 (1)	%Q0.0.3 (1)
	Frequency Meter	%I0.0.1	(2)	(2)	(2)	(2)	(2)
%VFC1	Selected Use	IA input	Input IB)	IPres	Ica	Output 0	Output 1
	Up/down counter	%I0.0.7	%I0.0.6 (UP = 0/DO = 1)	%I0.0.5 (1)	%I0.0.4 (1)	%Q0.0.4 (1)	%Q0.0.5 (1)
	Up/Down 2-Phase Counter	%I0.0.7	%I0.0.6 (Pulse)	%I0.0.5 (1)	%I0.0.4 (1)	%Q0.0.4 (1)	%Q0.0.5 (1)
	Single Up Counter	%I0.0.7	(2)	%I0.0.5 (1)	%I0.0.4 (1)	%Q0.0.4 (1)	%Q0.0.5 (1)
	Single Down Counter	%I0.0.7	(2)	%I0.0.5 (1)	%I0.0.4 (1)	%Q0.0.4 (1)	%Q0.0.5 (1)
	Frequency Meter	%I0.0.7	(2)	(2)	(2)	(2)	(2)
Comments: (1) = optional (2) = not used IPres = preset input Ica = Catch input When not used, the input or output remains a normal digital I/O available to be managed by the application in the main cycle. If %I0.0.2 is used %FC0 is not available. If %I0.0.3 is used %FC2 is not available. If %I0.0.4 is used %FC3 is not available.							

Illustration

Here is a block representation of the Very Fast Counter (%VFC) in single-word mode:



Specifications The following table lists characteristics for the very fast counter (%VFC) function block.

Function	Description	Values	%VFC Use	Run-time Access
Current Value (%VFCi.V) (%VFCi.VD*)	Current value that is increased or decreased according to the physical inputs and the function selected. This value can be preset or reset using the preset input (%VFCi.S).	%VFCi.V: 0 -> 65535 %VFCi.VD: 0 -> 4294967295	CM	Read
Preset value (%VFCi.P) (%VFCi.PD*)	Only used by the up/down counting function and single up or down counting.	%VFCi.P: 0 -> 65535 %VFCi.PD: 0 -> 4294967295	CM or FM	Read and Write (1)
Capture Value (%VFCi.C) (%VFCi.CD*)	Only used by the up/down counting function and single up or down counting.	%VFCi.C: 0 -> 65535 %VFCi.CD: 0 -> 4294967295	CM	Read
Counting direction (%VFCi.U)	Set by the system, this bit is used by the up/down counting function to indicate to you the direction of counting: As a single phase up or down counter, %I0.0.0 decides the direction for %VFC0 and %I0.0.6 for %VFC1. For a two-phase up/down counter, it is the phase difference between the two signals that determines the direction. For %VFC0, %I0.0 is dedicated to IB and %I0.1 to IA. For %VFC1, %I0.6 is dedicated to IB and %I0.7 to IA.	0 (Down counting) 1 (Up counting)	CM	Read
Enable Reflex Output 0 (%VFCi.R)	Validate Reflex Output 0	0 (Disable) 1 (Enable)	CM	Read and Write (2)
Enable Reflex Output 1 (%VFCi.S)	Validate Reflex Output 1	0 (Disable) 1 (Enable)	CM	Read and Write (2)
Threshold Value S0 (%VFCi.S0) (%VFCi.S0D*)	This word contains the value of threshold 0. The meaning is defined during configuration of the function block. Note: This value must be less than %VFCi.S1.	%VFCi.S0: 0 -> 65535 %VFCi.S0D: 0 -> 4294967295	CM	Read and Write (1)
Threshold Value S1 (%VFCi.S1) (%VFCi.S1D*)	This word contains the value of threshold 0. The meaning is defined during configuration of the function block. Note: This value must be greater than %VFCi.S0.	%VFCi.S1: 0 -> 65535 %VFCi.S1D: 0 -> 4294967295	CM	Read and Write (1)

Function	Description	Values	%VFC Use	Run-time Access
Frequency Measure Time Base (%VFCi.T)	Configuration item for 100 or 1000 millisecond time base.	1000 or 100	FM	Read and Write (1)
Adjustable (Y/N)	Configurable item that when selected, allows the user to modify the preset, threshold, and frequency measure time base values while running.	N (No) Y (Yes)	CM or FM	No
Enter to enable (IN)	Used to validate or inhibit the current function.	0 (No)	CM or FM	Read and Write (3)
Preset input (S)	Depending on the configuration, at state 1: - Up/Down or Down Counting: initializes the current value with the preset value. - Single Up Counting: resets the current value to zero. In addition, this also initializes the operation of the threshold outputs and takes into account any user modifications to the threshold values set by the Operator Display or user program.	0 or 1	CM or FM	Read and Write
Overflow output (F)	0 to 65535 or from 65535 to 0 in standard mode 0 to 4294967295 or from 4294967295 to 0 in double word mode	0 or 1	CM	Read
Threshold Bit 0 (%VFCi.TH0)	Set to 1 when the current value is greater than or equal to the threshold value %VFCi.S0. It is advisable to test this bit only once in the program because it is updated in real time. The user application is responsible for the validity of the value at its time of use.	0 or 1	CM	Read
Threshold Bit 1 (%VFCi.TH1)	Set to 1 when the current value is greater than or equal to the threshold value %VFCi.S1. It is advisable to test this bit only once in the program because it is updated in real time. The user application is responsible for the validity of the value at its time of use.	0 or 1	CM	Read

(*)Means a 32-bit double word variable. The double word option is available on all controllers with the exception of the Twido TWDLC•A10DRF controllers.

(1) Writable only if Adjust is set to one.

(2) Access available only if configured.

(3) Read and write access only through the application. Not the Operator Display or Animation Tables Editor.

CM = Counting Mode

FM = Frequency Meter Mode

Counting Function Description

The very fast counting function (%VFC) works at a maximum frequency of 20 kHz, with a range of 0 to 65535 in standard mode and 0 to 4294967295. The pulses to be counted are applied in the following way:

Table:

Function	Description	%VFC0		%VFC1	
		IA	IB	IA	IB
Up/Down Counter	The pulses are applied to the physical input, the current operation (upcount/downcount) is given by the state of the physical input IB.	%I0.0.1	%I0.0.0	%I0.0.7	%I0.0.6
Up/Down 2-Phase Counter	The two phases of the encoder are applied to physical inputs IA and IB.	%I0.0.1	%I0.0.0	%I0.0.7	%I0.0.6
Single Up Counter	The pulses are applied to the physical input IA. IB is not used.	%I0.0.1	ND	%I0.0.7	ND
Single Down Counter	The pulses are applied to the physical input IA. IB is not used.	%I0.0.1	ND	%I0.0.7	ND

Notes on Function Blocks

Upcount or downcount operations are made on the rising edge of pulses, and only if the counting block is enabled.

There are two optional inputs used in counting mode: Ica and IPres. Ica is used to capture the current value (%VFCi.V or %VFCi.VD) and stored it in %VFCi.C or %VFCi.CD. The Ica inputs are specified as %I0.0.3 for %VFC0 and %I0.0.4 for %VFC1 if available.

When IPres input is active, the current value is affected in the following ways:

- For up counting, %VFCi.V or %VFCi.VD is reset to 0
- For downcounting, %VFCi.V or %VFCi.VD is written with the content of %VFCi.P or %VFCi.PD, respectively.
- For frequency counting, %VFCi.V or %VFCi.PD is set to 0

Warning: %VFCi.F is also set to 0. The IPres inputs are specified as %I0.0.2 for %VFC0 and %I0.0.5 for %VFC1 if available.

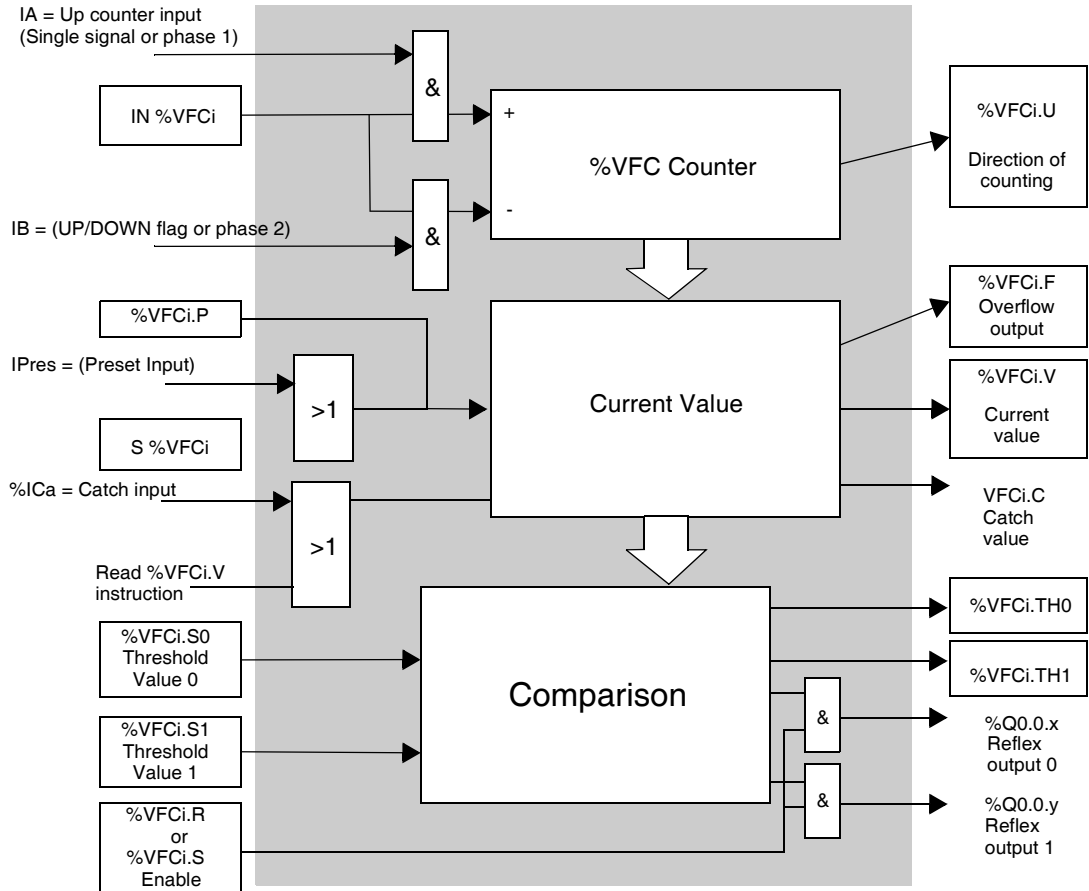
Notes on Function Block Outputs

For all functions, the current value is compared to two thresholds (%VFCi.S0 or %VFCi.S0D and % VFCi.S1 or %VFCi.S1D). According to the result of this comparison two bit objects (%VFCi.TH0 and %VFCi.TH1) are set to 1 if the current value is greater or equal to the corresponding threshold, or reset to 0 in the opposite case. Reflex outputs (if configured) are set to 1 in accordance with these comparisons. Note: None, 1 or 2 outputs can be configured.

%VFC.U is an output of the FB, it gives the direction of the associated counter variation (1 for UP, 0 for DOWN).

Counting Function Diagram

The following is a counting function diagram in standard mode (in double word mode, you will use the double word function variables, accordingly):



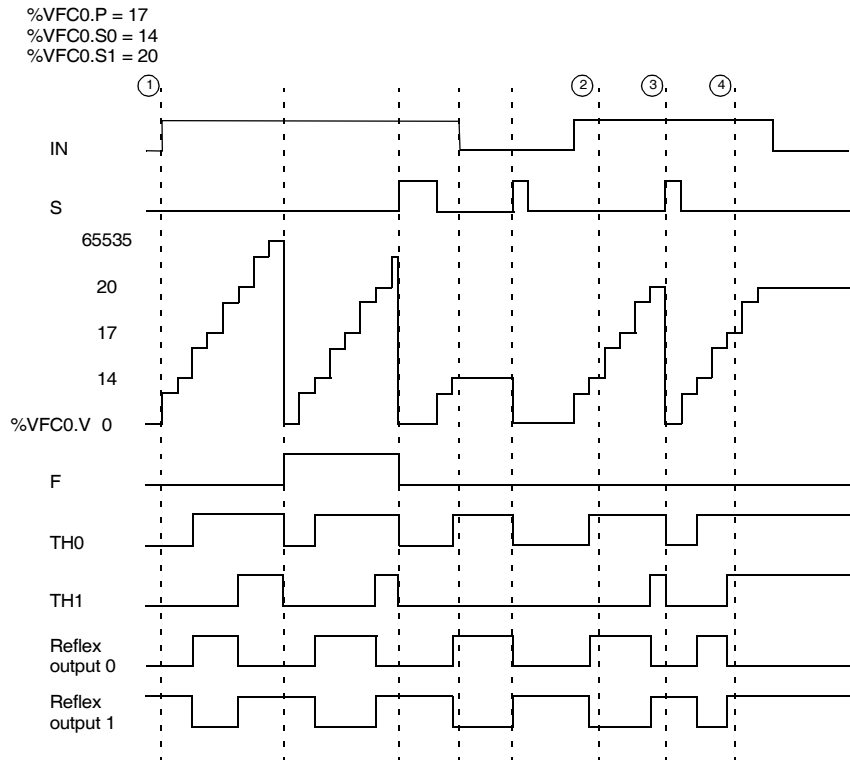
Note: Outputs are managed independently from the controller cycle time. The response time is between 0 and 1ms.

Single Up Counter Operation

The following is an example of using %VFC in a single up counter mode. The following configuration elements have been set for this example:
 %VFC0.P preset value is 17, while the %VFC0.S0 lower threshold value is 14, and the %VFC0.S1 upper threshold is 20.

Reflex Output	value < %VFC.S0	%VFC0.S0 <= value < %VFC0.S1	value >= %VFC0.S1
%Q0.0.2		X	
%Q0.0.3	X		X

A timing chart follows:



- ① : %VFC0.U = 1 because %VFC is an up-counter
- ② : change %VFC0.S1 to 17
- ③ : S input active makes threshold S1 new value to be granted in next count
- ④ : a catch of the current value is made, so %VFC0.C = 17

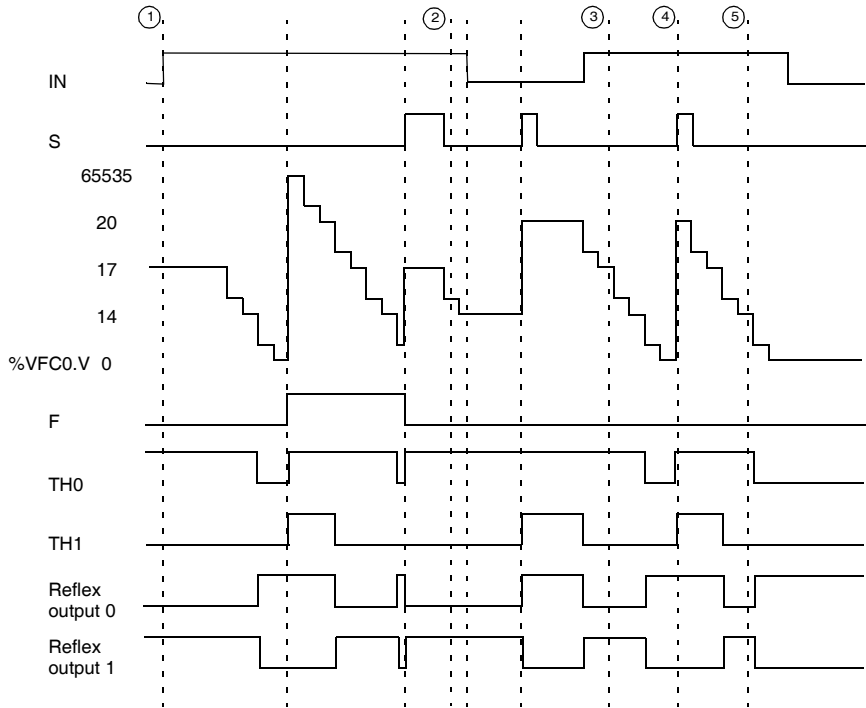
**Single Down
Counter
Operation**

The following is an example of using %VFC in a single down counter mode. The following configuration elements have been set for this example: %VFC0.P preset value is 17, while the %VFC0.S0 lower threshold value is 14, and the %VFC0.S1 upper threshold is 20.

Reflex Output	value < %VFC.S0	%VFC0.S0 <= value < %VFC0.S1	value >= %VFC0.S1
%Q0.0.2	X		X
%Q0.0.3		X	

Example:

%VFC0.P = 17
 %VFC0.S0 = 14
 %VFC0.S1 = 20



- ① : %VFC0.U = 0 because %VFC is a down-counter
- ② : change %VFC0.P to 20
- ③ : change %VFC0.S1 to 17
- ④ : S input active makes threshold S1 new value to be granted in next count
- ⑤ : a catch of the current value is made, so %VFC0.C = 17

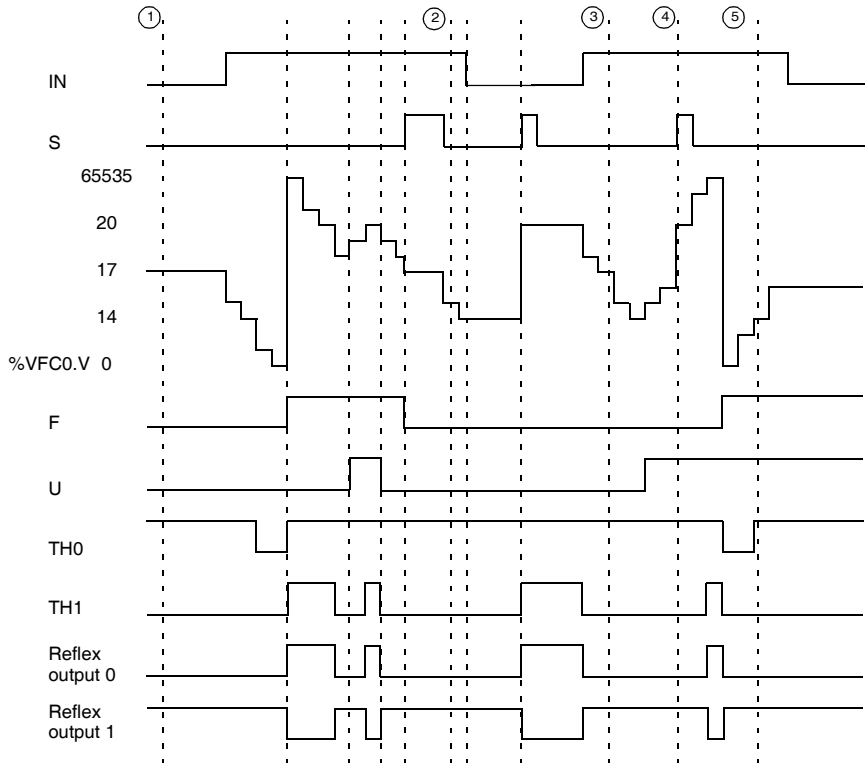
**Up-Down
Counter
Operation**

The following is an example of using %VFC in an up-down counter mode. The following configuration elements have been set for this example: %VFC0.P preset value is 17, while the %VFC0.S0 lower threshold value is 14, and the %VFC0.S1 upper threshold is 20.

Reflex Output	value < %VFC.S0	%VFC0.S0 <= value < %VFC0.S1	value >= %VFC0.S1
%Q0.0.2			X
%Q0.0.3	X	X	

Example:

%VFC0.P = 17
 %VFC0.S0 = 14
 %VFC0.S1 = 20



- ① : Input IN is set to 1 and input S set to 1
- ② : change %VFC0.P to 20
- ③ : change %VFC0.S1 to 17
- ④ : S input active makes threshold S1 new value to be granted in next count
- ⑤ : a catch of the current value is made, so %VFC0.C = 17

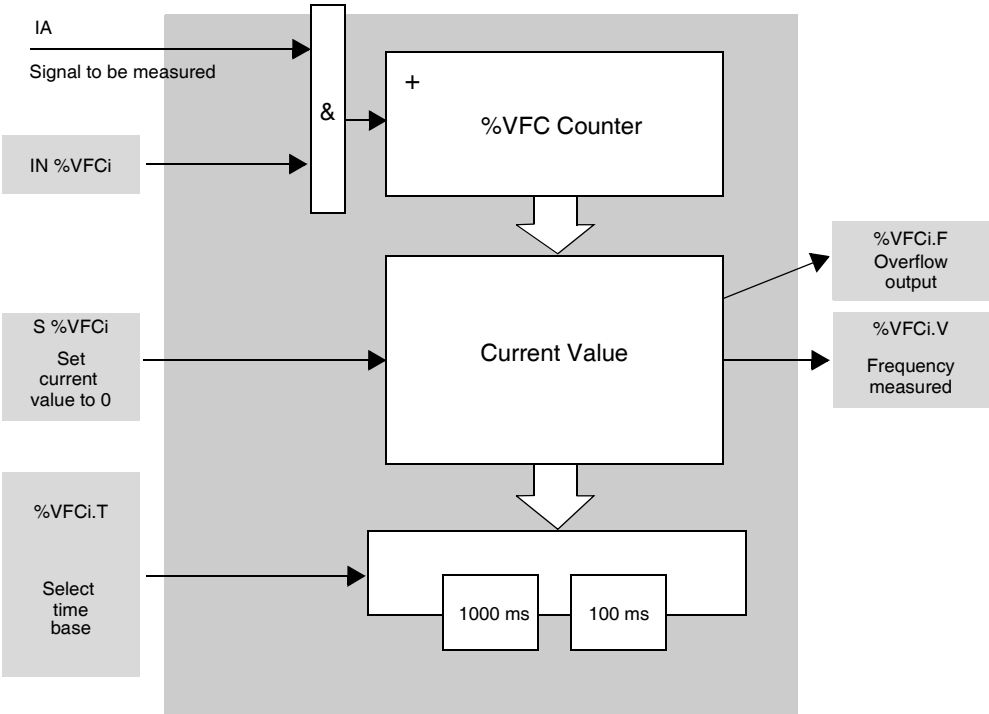
**Frequency Meter
Function
Description**

The frequency meter function of a %VFC is used to measure the frequency of a periodic signal in Hz on input IA. The frequency range which can be measured is from 10 to 20kHz. The user can choose between 2 time bases, the choice being made by a new object %VFC.T (Time base). A value of 100 = time base of 100 ms and a value of 1000 = time base of 1 second.

Time Base	Measurement range	Accuracy	Update
100 ms	100 Hz to 20 kHz	0.05 % for 20 kHz, 10 % for 100 Hz	10 times per second
1 s	10 Hz to 20 kHz	0.005 % for 20 kHz, 10 % for 10 Hz	Once per second

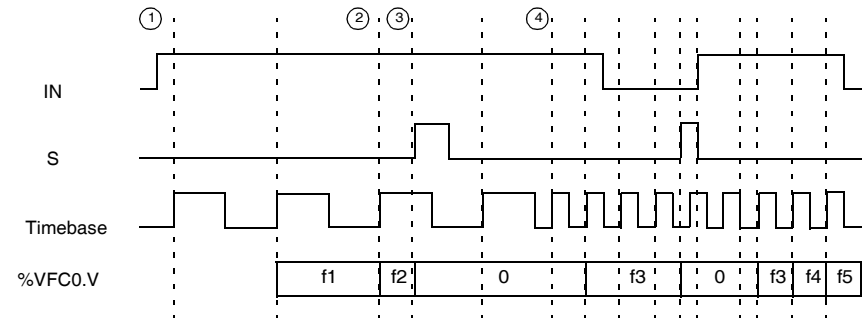
**Frequency Meter
Function
Diagram**

The following is a frequency meter function diagram:



Frequency Meter Operation

The following is a timing diagram example of using %VFC in a frequency meter mode.



- ① : The first frequency measurement starts here.
- ② : The current frequency value is updated.
- ③ : Input IN is 1 and input S is 1
- ④ : Change %VFC0.T to 100 ms: this change cancels the current measurement and starts another one.

Special Cases

The following table shows a list of special operating of the %VFC function block.

Special case	Description
Effect of cold restart (%S0=1)	Resets all the %VFC attributes with the values configured by the user or user application.
Effect of warm restart (%S1=1)	Has no effect
Effect of Controller stop	The %VFC stops its function and the outputs stay in their current state.

Transmitting/Receiving Messages - the Exchange Instruction (EXCH)

Introduction

A Twido controller can be configured to communicate with Modbus slave devices or can send and/or receive messages in character mode (ASCII).

TwidoSoft provides the following functions for these communications:

- EXCH instruction to transmit/receive messages
- Exchange control function block (%MSG) to control the data exchanges

The Twido controller uses the protocol configured for the specified port when processing an EXCH instruction. Each communication port can be assigned a different protocol. The communication ports are accessed by appending the port number to the EXCH or %MSG function (EXCH1, EXCH2, %MSG1, %MSG2). In addition, TWDLCAE40DRF series controllers implement Modbus TCP messaging over the Ethernet network by using the EXCH3 instruction and %MSG3 function.

EXCH Instruction

The EXCH instruction allows a Twido controller to send and/or receive information to/from ASCII devices. The user defines a table of words (%MWi:L) containing the data to be sent and/or received (up to 250 data bytes in transmission and/or reception). The format for the word table is described in the paragraphs about each protocol. A message exchange is performed using the EXCH instruction.

Syntax

The following is the format for the EXCH instruction:

[EXCHx %MWi:L]

Where: x = serial port number (1 or 2); x = Ethernet port (3); L = total number of words of the word table (maximum 121). Values of the internal word table %MWi:L are such as $i+L \leq 255$.

The Twido controller must finish the exchange from the first EXCHx instruction before a second exchange instruction can be started. The %MSG function block must be used when sending several messages.

Note: To find out more information about the Modbus TCP messaging instruction EXCH3, please refer to *TCP Modbus Messaging*, p. 180.

Exchange Control Function Block (%MSGx)

Introduction

Note: The "x" in %MSGx signifies the controller port: "x = 1 or 2"

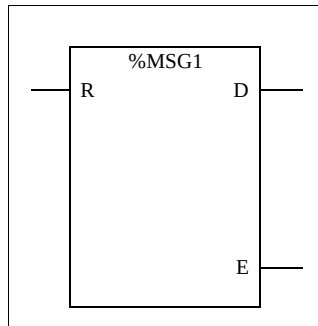
- x = 1 or 2, signifies the serial port 1 or 2 of the controller, respectively;
- x = 3, signifies the Ethernet network port of the controller (on TWDLCAE40DRF controllers only). For more information about the %MSG3 function, please refer to *TCP Modbus Messaging*, p. 180.

The %MSGx function block manages data exchanges and has three functions:

- Communications error checking:
Error checking verifies that the block length (word table) programmed with the EXCH instruction is large enough to contain the length of the message to be sent (compare with length programmed in the least significant byte of the first word of the word table).
 - Coordination of multiple messages:
To ensure coordination when sending multiple messages, the %MSGx function block provides the information required to determine when a previous message is complete.
 - Transmission of priority messages:
The %MSGx function block allows the current message transmission to be stopped, in order to allow the immediate sending of an urgent message.
- The programming of the %MSGx function block is optional.

Illustration

The following is an example of the %MSGx function block.



Parameters

The following table lists parameters for the %MSGx function block.

Parameter	Label	Value
Reset input (or instruction)	R	At state 1, reinitializes communication: %MSGx.E = 0 and %MSGx.D = 1.
Comm. done output	%MSGx.D	State 1, comm. done, if: ● End of transmission (if transmission) ● End of reception (end character received) ● Error ● Reset the block State 0, request in progress.
Fault (Error) output	%MSGx.E	State 1, comm. done, if: ● Bad command ● Table incorrectly configured ● Incorrect character received (speed, parity, etc.) ● Reception table full (not updated) State 0, message length OK, link OK.

If an error occurs when using an EXCH instruction, bits %MSGx.D and %MSGx.E are set to 1, and system word %SW63 contains the error code for Port 1, and %SW64 contains the error code for Port 2. See *System Words (%SW)*, p. 604.

Reset Input (R)

When Reset Input set to 1:

- Any messages that are being transmitted are stopped.
- The Fault (Error) output is reset to 0.
- The Done bit is set to 1.

A new message can now be sent.

Fault (Error) Output (%MSGx.E)

The error output is set to 1 either because of a communications programming error or a message transmission error. The error output is set to 1 if the number of bytes defined in the data block associated with the EXCH instruction (word 1, least significant byte) is greater than 128 (+80 in hexadecimal by FA).

The error output is also set to 1 if a problem exists in sending a Modbus message to a Modbus device. In this case, the user should check wiring, and that the destination device supports Modbus communication.

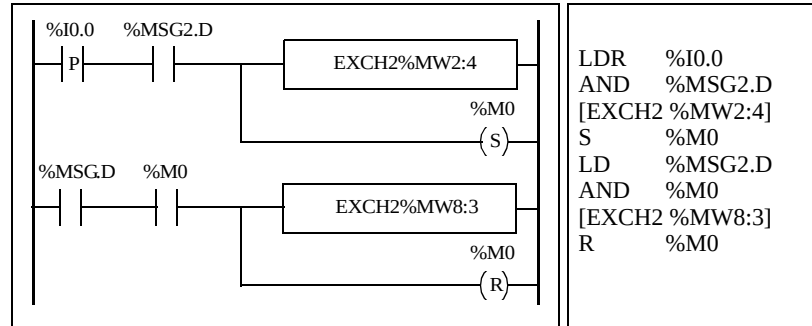
Communications Done output (%MSGx.D)

When the Done output is set to 1, the Twido controller is ready to send another message. Use of the %MSGx.D bit is recommended when multiple messages are sent. If it is not used, messages may be lost.

Transmission of Several Successive Messages

Execution of the EXCH instruction activates a message block in the application program. The message is transmitted if the message block is not already active (%MSGx.D = 1). If several messages are sent in the same cycle, only the first message is transmitted. The user is responsible for managing the transmission of several messages using the program.

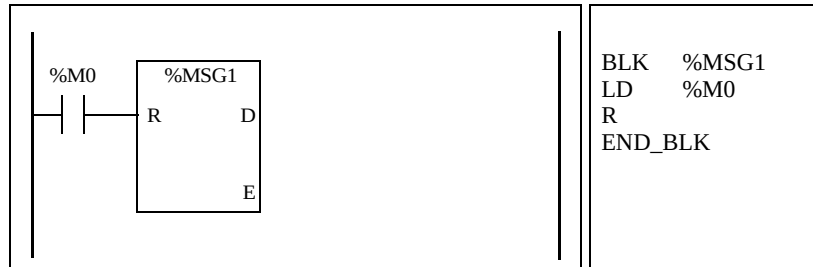
Example of a transmission of two messages in succession on port 2:



Reinitializing Exchanges

An exchange is cancelled by activating the input (or instruction) R. This input initializes communication and resets output %MSGx.E to 0 and output %MSGx.D to 1. It is possible to reinitialize an exchange if a fault is detected.

Example of reinitializing an exchange:



Special Cases

The following table the special operating cases for the %MSGx function block.

Special Case	Description
Effect of a cold restart (%S0=1)	Forces a reinitialization of the communication.
Effect of a warm restart (%S1=1)	Has no effect.
Effect of a controller stop	If a message transmission is in progress, the controller stops its transfer and reinitializes the outputs %MSGx.D and %MSGx.E.

17.2 Clock Functions

At a Glance

Aim of this Section This section describes the time management functions for Twido controllers.

What's in this Section? This section contains the following topics:

Topic	Page
Clock Functions	481
Schedule Blocks	482
Time/Date Stamping	485
Setting the Date and Time	487

Clock Functions

Introduction

Twido controllers have a time-of-day clock function, which requires the Real-Time Clock option (RTC) and provides the following:

- **Schedule blocks** are used to control actions at predefined or calculated times.
- **Time/date stamping** is used to assign time and dates to events and measure event duration.

The Twido time-of-day clock can be accessed by selecting **Schedule Blocks** from from the TwidoSoft **Software** menu. Additionally, the time-of-day clock can be set by a program. Clock settings continue to operate for up to 30 days when the controller is switched off, if the battery has been charged for at least six consecutive hours before the controller is switched off.

The time-of-day clock has a 24-hour format and takes leap years into account.

RTC Correction Value

The RTC Correction value is necessary for the correct operation of the RTC. Each RTC unit has its own correction value written on the unit. This value is configurable in TwidoSoft by using the **Configure RTC** option from the **Controller Operations** dialog box.

Schedule Blocks

Introduction

Schedule Blocks are used to control actions at a predefined month, day, and time. A maximum of 16 schedule blocks can be used and do not require any program entry.

Note: Check system bit %S51 and system word %SW118 to confirm that the Real-Time Clock (RTC) option is installed see *System Bits (%S)*, p. 596. The RTC option is required for using schedule blocks.

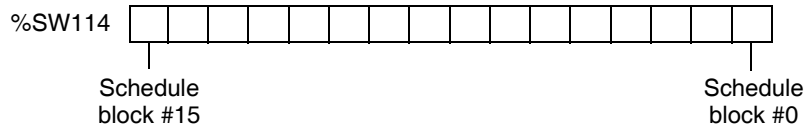
Parameters

The following table lists parameters for a schedule block:

Parameter	Format	Function/Range
Schedule block number	n	n = 0 to 15
Configured	Check box	Check this box to configure the selected schedule block number.
Output bit	%Qx.y.z	Output assignment is activated by schedule block: %Mi or %Qj.k. This output is set to 1 when the current date and time are between the setting of the start of the active period and the setting of the end of the active period.
Start month	January to December	The month to start the schedule block.
End month	January to December	The month to end the schedule block.
Start date	1 - 31	The day in the month to start the schedule block.
End date	1 - 31	The day in the month to end the schedule block.
Start time	hh:mm	The time-of-day, hours (0 to 23) and minutes (0 to 59), to start the schedule block.
Stop time	hh:mm	The time-of-day, hours (0 to 23) and minutes (0 to 59), to end the schedule block.
Day of week	Monday - Sunday	Check boxes that identify the day of the week for activation of the schedule block.

**Enabling
Schedule Blocks**

The bits of system word %SW114 enable (bit set to 1) or disable (bit set to 0) the operation of each of the 16 schedule blocks.
Assignment of schedule blocks in %SW114:



By default (or after a cold restart) all bits of this system word are set to 1. Use of these bits by the program is optional.

**Output of
Schedule Blocks**

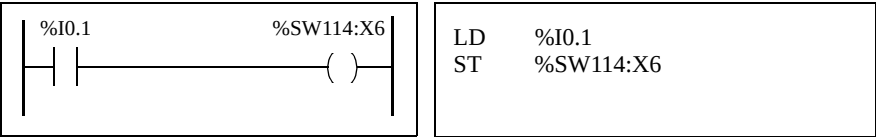
If the same output (%Mi or %Qj.k) is assigned by several blocks, it is the OR of the results of each of the blocks which is finally assigned to this object (it is possible to have several "operating ranges" for the same output).

Example

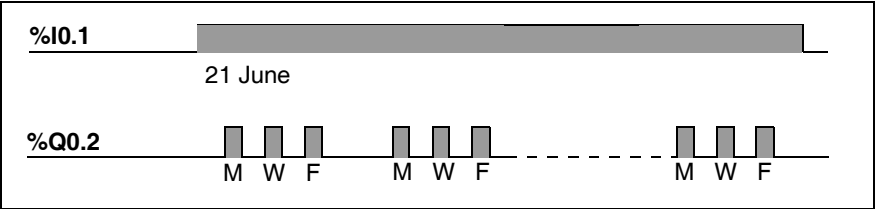
The following table shows the parameters for a summer month spray program example:

Parameter	Value	Description
Schedule block	6	Schedule block number 6
Output bit	%Q0.2	Activate output %Q0.2
Start month	June	Start activity in June
End month	September	Stop activity in September
Start date	21	Start activity on the 21st day of June
End date	21	Stop activity on the 21st day of September
Day of week	Monday, Wednesday, Friday	Run activity on Monday, Wednesday and Friday
Start time	21:00	Start activity at 21:00
Stop time	22:00	Stop activity at 22:00

Using the following program, the schedule block can be disabled through a switch or a humidity detector wired to input %I0.1.



The following timing diagram shows the activation of output %Q0.2.



Time Dating by Program

Date and time are both available in system words %SW50 to %SW53 (see *System Words (%SW)*, p. 604). It is therefore possible to perform time and date stamping in the controller program by making arithmetic comparisons between the current date and time and the immediate values or words %MWi (or %KWi), which can contain setpoints.

Time/Date Stamping

Introduction

System words %SW49 to %SW53 contain the current date and time in BCD format (see *Review of BCD Code*, p. 425, which is useful for display on or transmission to a peripheral device. These system words can be used to store the time and date of an event (see *System Words (%SW)*, p. 604).

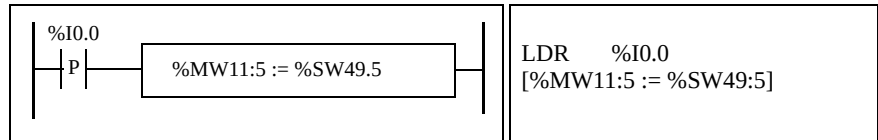
Note: Date and time can also be set by using the optional Operator Display (see *Time of Day Clock*, p. 318).

Dating an Event

To date an event it is sufficient to use assignment operations, to transfer the contents of system words to internal words, and then process these internal words (for example, transmission to display unit by EXCH instruction).

Programming Example

The following example shows how to date a rising edge on input %I0.1.



Once an event is detected, the word table contains:

Encoding	Most significant byte	Least significant byte
%MW11		Day of the week ¹
%MW12	00	Second
%MW13	Hour	Minute
%MW14	Month	Day
%MW15	Century	Year

Note: (1) 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday, 7 = Sunday.

Example of Word Table

Example data for 13:40:30 on Monday, 19 April, 2002:

Word	Value (hex)	Meaning
%MW11	0001	Monday
%MW12	0030	30 seconds
%MW13	1340	13 hours, 40 minutes
%MW14	0419	04 = April, 19th
%MW15	2002	2002

Date and time of the last stop

System words %SW54 to %SW57 contain the date and time of the last stop, and word %SW58 contains the code showing the cause of the last stop, in BCD format (see *System Words (%SW)*, p. 604).

Setting the Date and Time

Introduction

You can update the date and time settings by using one of the following methods:

- TwidoSoft
Use the **Set Time** dialog box. This dialog box is available from the **Controller Operations** dialog box. This is displayed by selecting **Controller Operations** from the **Controller** menu.
- System Words
Use system words %SW49 to %SW53 or system word %SW59.

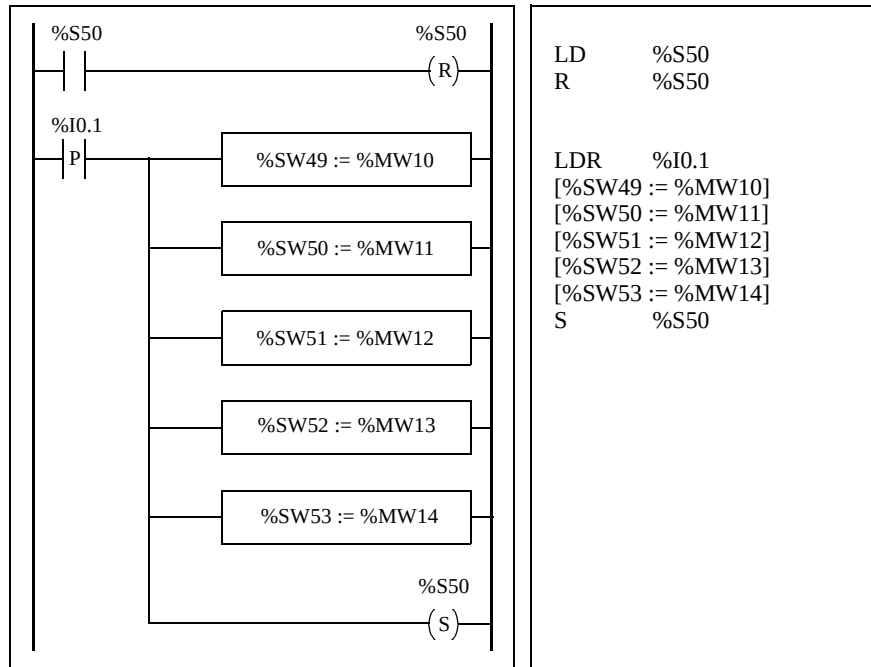
The date and time settings can only be updated when the RTC option cartridge (TWDXCPRTC) is installed on the controller. Note that the TWDLCA•40DRF series of compact controllers have RTC onboard.

Using %SW49 to %SW53

To use system words %SW49 to %SW53 to set the date and time, bit %S50 must be set to 1. This results in the following:

- Cancels the update of words %SW49 to %SW53 via the internal clock.
- Transmits the values written in words %SW49 to %SW53 to the internal clock.

Programming Example:



Words %MW10 to %MW14 will contain the new date and time in BCD format (see *Review of BCD Code, p. 425*) and will correspond to the coding of words %SW49 to %SW53.

The word table must contain the new date and time:

Encoding	Most significant byte	Least significant byte
%MW10		Day of the week ¹
%MW11		Second
%MW12	Hour	Minute
%MW13	Month	Day
%MW14	Century	Year

Note: (1) 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday, 7 = Sunday.
--

Example data for Monday, 19 April, 2002:

Word	Value (hex)	Meaning
%MW10	0001	Monday
%MW11	0030	30 seconds
%MW12	1340	13 hours, 40 minutes
%MW13	0419	04 = April, 19th
%MW14	2002	2002

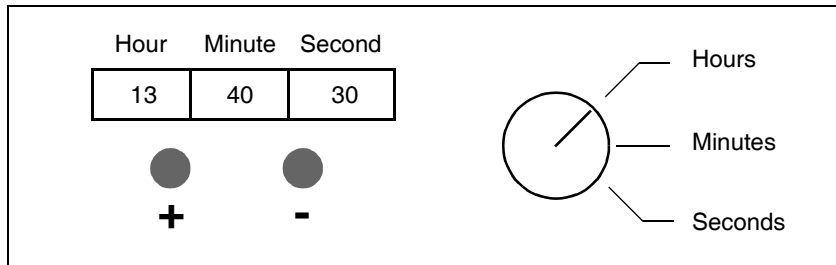
Using %SW59

Another method of updating the date and time is to use system bit %S59 and date adjustment system word %SW59.

Setting bit %S59 to 1 enables adjustment of the current date and time by word %SW59 (see *System Words (%SW), p. 604*). %SW59 increments or decrements each of the date and time components on a rising edge.

Application Example

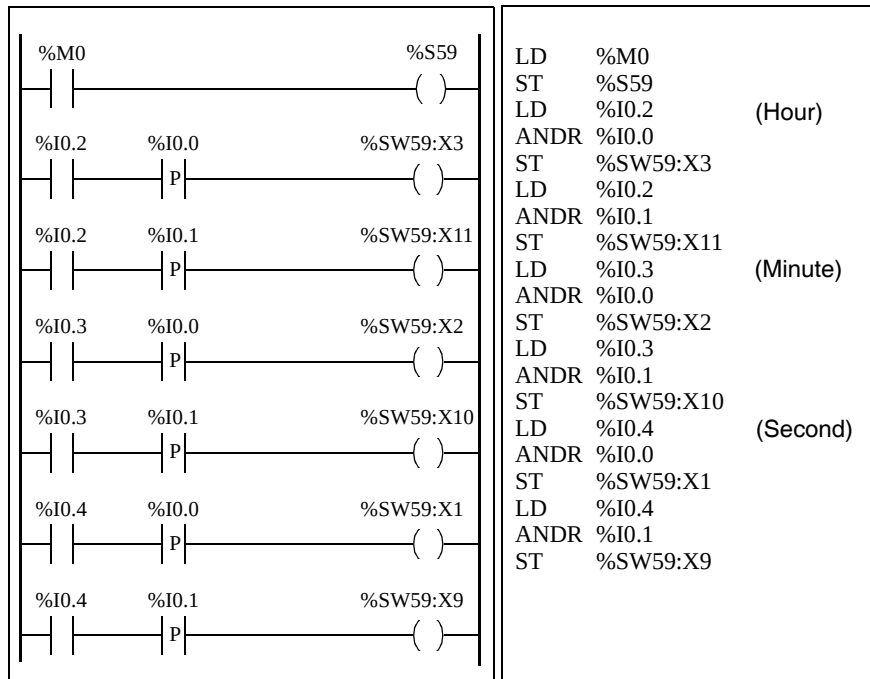
The following front panel is created to modify the hour, minutes, and seconds of the internal clock.



Description of the Commands:

- The Hours/Minutes/Seconds switch selects the time display to change using inputs %I0.2, %I0.3, and %I0.4 respectively.
- Push button "+" increments the selected time display using input %I0.0.
- Push button "-" decrements the selected time display using input %I0.1.

The following program reads the inputs from the panel and sets the internal clock.



17.3 Twido PID Quick Start Guide

At a Glance

Overview This section contains information for getting started with the PID control and Auto-Tuning functions available on Twido controllers.

What's in this Section? This section contains the following topics:

Topic	Page
Purpose of Document	491
Step 1 - Configuration of Analog Channels Used for Control	493
Step 2 - Prerequisites for PID Configuration	495
Step 3 - Configuring the PID	497
Step 4 - Initialization of Control Set-Up	504
Step 5 - Control Set-Up AT + PID	509
Step 6 - Debugging Adjustments	513

Purpose of Document

Introduction

This quick start guide aims to guide you, by providing examples, through all the steps required to correctly configure and set up your Twido controller's PID control functions.

Note: Implementing the PID function on a Twido does not require any specific knowledge but does demand a certain degree of rigor to ensure that you retain optimum results in the shortest possible time.

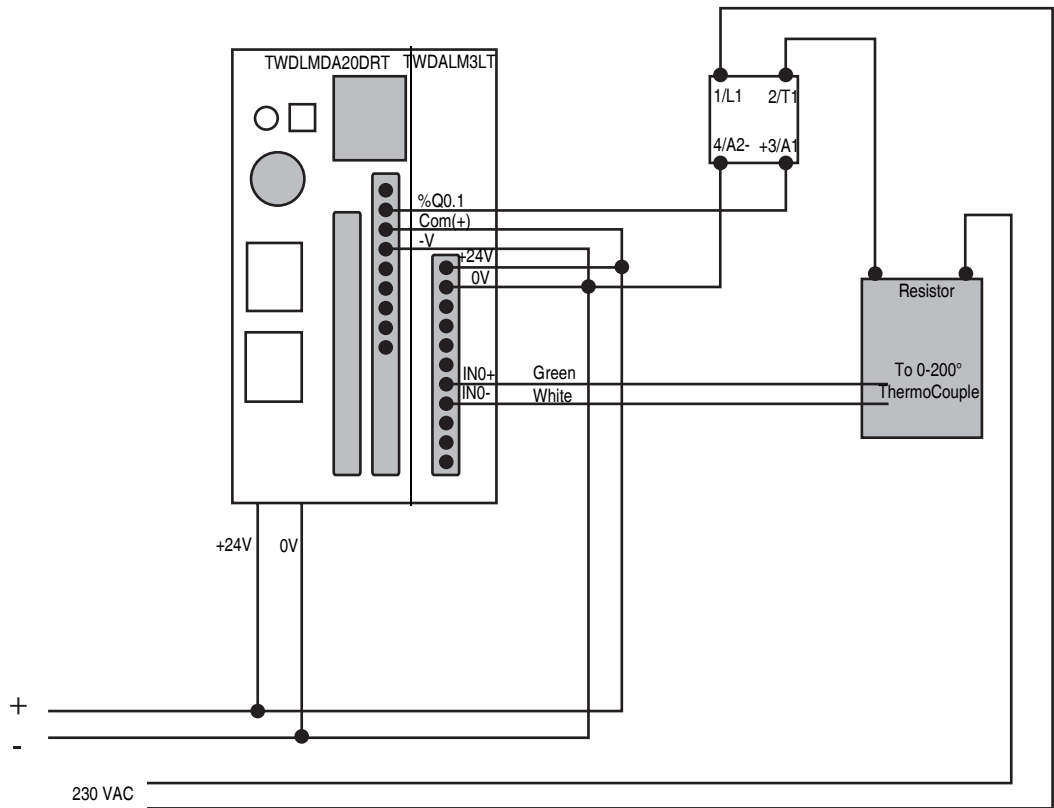
This document contains:

This document explains the following steps:

Step	Description
1	Configuration of analog channels used for control
2	Prerequisites for PID configuration
3	PID configuration
4	Initialization of control setup.
5	AT + PID control setup
6	Debugging and adjustments

Concerning the example used in this guide

For this example, we have chosen a Type K ThermoCouple (0-200°). We will use transistor control with the output therefore being a base controller output controlled directly by the PID controller using PWM (see *Step 3 - Configuring the PID*, p. 497.) The diagram below shows the experimental setup used in the example:



Step 1 - Configuration of Analog Channels Used for Control

Introduction

In general, a PID controller uses an analog feedback signal (known as the "process value") to measure the value to be adjusted.

This value can be a level, a temperature, a distance, or another value for other applications.

Example of an Analog Measurement Signal

Let us take the example of a temperature measurement.

The sensor used sends an analog measurement which depends on the measured value back to the controller. For temperature and with sensors like PT100s or Thermocouples, the measured signal increases with an increase in current temperature.

How to Add an Analog Card (Expansion Module)

In offline mode, once you have selected the base controller, add the analog card as a base extension. The numbering of the channels will depend on its configuration slot.

How to Configure Analog Input Channels

The following table describes the procedure for configuring the analog channels of the expansion module:

Step	Action
1	Right-click on Expansion Bus → Add Module .
2	Select the desired card from the list. For example, TWDALM3LT for measuring temperature using a PT100 or Thermocouple.
3	Click on Add then Finish if the configuration is only for a single module.
4	Right-click on the card you have added then select Configure .
5	In the Type column, select the input type corresponding to the type of sensor used (ThermoCouple K, if the sensor is of this type).
6	In the Range column, select the measurement unit for the sensor. For temperature sensors it is easier to select Celsius , as this makes the number of counts sent back by the analog card a direct factor of the real measurement.
7	Provide an address for the input symbol of the configured analog card. It will be used to complete the PID fields (%IW1.0, for this example).
8	Do the same for an analog output if an output must be used to drive the control system.

Example of
Analog Channel
Configuration

Several types of configuration are possible depending on the type of measurement used, as indicated below:

- For the application in the example used in this document, we have chosen a **Type K ThermoCouple** (0-200°). The process value read will be directly comprehensible (2000 counts = 200° as the unit factor is 0.1).
- For other types of measurement, you choose **0-10V** or **4-20 mA** in the **Type** column, or **Custom** in the **Range** column. Then adjust the value scale (enter **0** in the **Minimum** column and **10000** in the **Maximum** column) to be able to read the process value directly (10 V = 10000 counts).

The example below shows a configuration for a ThermoCouple K analog channel:

Configure Module - TWDALM3LT [Position 1]

Description

Expansion module with 2 Analog Inputs (RTD - Th) and 1 Output (0 - 10V, 4 - 20mA), 12 bits, removable screw terminal. K, J, T thermocouple and 3-wire PT100. (50mA)

	Symbol	Type	Range	Minimum	Maximum	Units
%IW1.0		Thermocouple K	Celsius	0	13000	0.1 °C
%IW1.1		Not used	Normal	0	4095	None
%QW1.0		Not used	Normal	0	4095	None

OK

Cancel

Reset

Help

Step 2 - Prerequisites for PID Configuration

Introduction

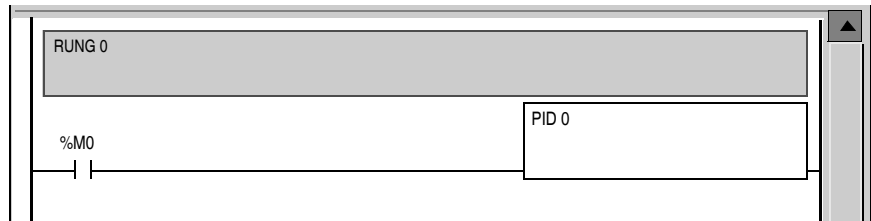
Before configuring the PID, ensure that the following phases have been performed:

Phase	Description
1	PID enabled in the program.
2	Scan period configured

Enabling PID in the Program

The PID controller must be enabled in the program by an instruction. This instruction can be permanent or be a condition of an input or internal bit. In the following example, the PID is enabled by the instruction %M0:

- In Ladder:



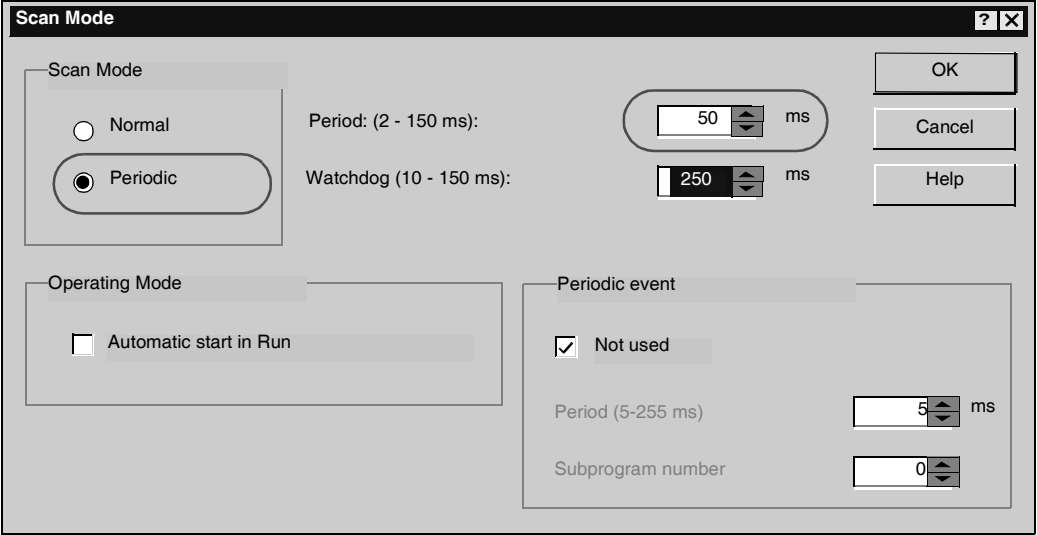
- In Instruction List:

```
-----  
0 LD      %M0  
1 [ PID 0 ]
```

Note: Ensure that you use correct syntax:

Check that there is a space between "PID" and the PID number (e.g. PID<space>0).

Configuration of Scan Period When using PID controllers, you are strongly advised to configure the scan mode of the PLC cycle to periodic. The table below describes the procedure for configuring the scan mode.

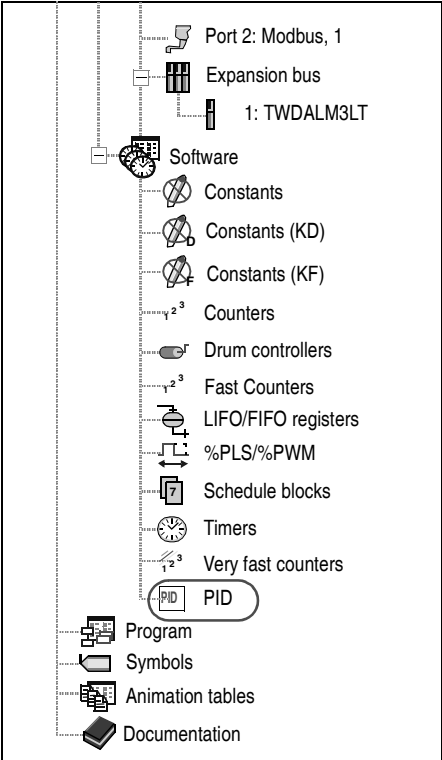
Step	Action
1	From the TwidoSoft menu bar, select Program → Edit Scan Mode .
2	Check the Periodic box.
3	Set the cycle time as shown in the screen below: 
	Note: The cycle time should be adjusted to the size of the program and desired performance. (A time of 50ms is a good compromise).

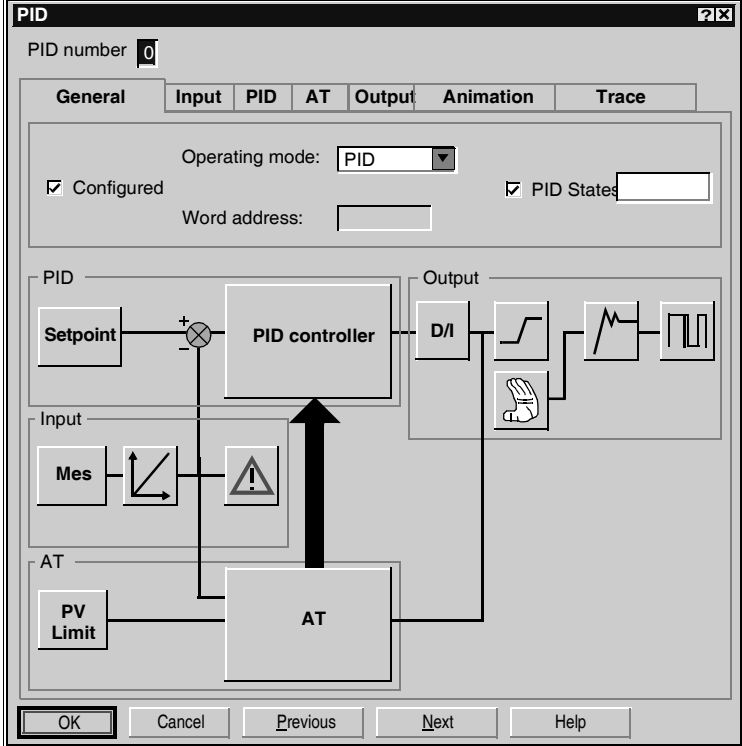
Step3 – Configuring the PID

Introduction	For this example, we have chosen to implement the majority of the PID controller functions for Twido. Some selections are not essential and can be simplified.
Auto-Tuning (AT)	The PID controller has an Auto-Tuning function that simplifies the regulation loop setting (this function is referred to as AT in the rest of the document).
Operating Modes	The Twido PLC PID controller offers four distinct operating modes, configurable in the General tab in the PID dialog box: ● PID = Simple PID controller. ● AT + PID = The Auto-Tuning function is active when the PID starts up and automatically enters the gain values Kp, Ti, Td (PID tab) and the type of PID action (Output tab). At the end of the Auto-Tuning, sequence, the controller switches to PID mode for the adjusted setpoint, and using the parameters set by AT. ● AT = The Auto-Tuning function is active when the PID starts up and automatically enters the gain values Kp, Ti, Td (PID tab) and the type of PID action (Output tab). At the end of the sequence the PID stops and waits. The gain values Kp, Ti, Td (PID tab) and the type of PID action (Output tab) are entered. ● Word address = The selection of PID operating mode can be controlled by the program by assigning the desired value to the word address associated to this selection: ● %MWxx=1: The controller operates in simple PID mode. ● %MWxx=2: The controller operates in AT + PID. ● %MWxx=3: The controller operates in AT mode only. This type of configuration via the word address enables the user to manage the PID controller operating mode via the application program, thus making it possible to adapt to the final requirements.

Launching the PID Dialog Box

The table below shows the PID dialog box and the procedure for accessing the different PID settings configuration tabs:

Step	Action
1	Double click on the PID item in the configuration browser in the left-hand side of the TwidoSoft window as shown in the figure below: A screenshot of the TwidoSoft configuration browser. It shows a hierarchical tree of configuration items. The 'PID' item is highlighted with a red oval. The items listed are: Port 2: Modbus, 1; Expansion bus; 1: TWDALM3LT; Software; Constants; Constants (KD); Constants (KF); Counters; Drum controllers; Fast Counters; LIFO/FIFO registers; %PLS/%PWM; Schedule blocks; Timers; Very fast counters; PID; Program; Symbols; Animation tables; and Documentation.

Step	Action
2	The PID dialog box appears in the foreground and is used to enter the different controller settings as shown in the figure below. In offline mode, this displays several tabs: General, Input, PID, AT, Output:  Important: The tabs must be entered in the order in which they appear in the PID dialog box: first General, Input, PID, AT then Output. Note: In online mode, this screen displays two more tabs, Animation and Trace, used respectively for the diagnostics and display of the controller operation.

Dynamic Modification of Parameters

For the dynamic modification of the PID parameters (in operation and in online mode), it is advised to enter the memory addresses in the associated fields, thus avoiding switching to offline mode to make on-the-fly changes to values.

General Tab Setting

The following table shows how to set the **General** tab in the PID dialog box:

Step	Action
1	In the General tab, check the Configured box to activate the PID and set the following tabs.
2	In the Operating mode drop-down list, select the type of operation desired (See <i>Operating Modes</i>, p. 497). In the example: We will select the Memory address mode and enter the word %MW17 in the associated field. The PID operating mode will then be linked to the value in %MW17.

Input Tab Setting

The following table shows how to set the **Input** tab in the PID dialog box:

Step	Action
1	In the Input tab, enter the analog channel used as a measurement in the associated field. In the example: We have chosen %IW1.0 as this is used as a temperature measurement.
2	Where necessary, set alarms on the low and high measurement thresholds by checking the boxes and filling in the associated fields. Note: The values entered may be fixed values (entered in the associated fields) or modifiable values (by filling in the fields associated with the memory addresses: %MWxx).

PID Tab Setting

The following table shows how to set the **PID** tab in the PID dialog box:

Step	Action
1	In the PID tab, enter the value to be used to set the controller setpoint. In general, this value is a memory address or setpoint of an analog input. In the example: We have entered %MW0, which will be used as a setpoint word.
2	Set the Kp, Ti, Td parameters. Important: If the AT or AT+PID mode is selected, it is essential that the Kp, Ti and Td fields be completed with memory addresses , to enable the Auto-Tuning function to automatically fill in the values found. In the example: We have entered %MW10 for Kp, %MW11 for Ti and %MW12 for Td. Note: In principle, it is rather difficult to determine the optimal adjustment values of Kp, Ti and Td for an application that has not yet been created. Consequently, we strongly recommend you enter the memory words addresses in these fields, in order to enter these values in online mode thus avoiding switching to offline mode to make on-the-fly changes to values.
3	Enter the PID Sampling period . This value is used by the controller to acquire measurements and update outputs. In the example: We have set the PID sampling period to 100, or 1s. Given that the adjusted system has a time constant of several minutes, this sampling period value seems correct. Important: We advise you set the sampling period to a multiple of the controller scan period, and a value consistent with the adjusted system.

Tab Setting for AT

The following table shows how to set the **AT** tab in the PID dialog box:

Step	Action
1	In the AT tab, check the Authorize box if you want to use AT.
2	Enter the Measurement limit value. This is the limit value that the measurement must not exceed during AT.
3	Enter the Output setpoint value which is the controller output value sent to generate AT.
Special Note	For further details about setting these values refer to the AT tab of PID function, p. 532 section.
Advice	We strongly recommend you enter the memory words addresses in these fields, in order to enter these values in online mode thus avoiding switching to offline mode to make on-the-fly changes to values.

Output Tab Setting

The following table shows how to set the **Output** tab in the PID dialog box:

WARNING

RISK OF SYSTEM OVERLOAD

You are reminded that manual mode has a direct effect on the controller output. Consequently, sending a manual setpoint (**Output** field) acts directly on the open controlled system. You should therefore proceed with care in this operating mode.

Failure to follow this instruction can result in death, serious injury, or equipment damage.

Step	Action
1	In the Output tab, enter the selection from the Action drop-down list. This selection depends on the configured system: ● Direct action: the controller output decreases as the variation value (setpoint - measurement) increases (cold controller). ● Inverse action: the controller output decreases as the variation value (setpoint - measurement) increases (hot controller). Important: When using the AT function, this list automatically selects Bit address. The operating mode is determined by the AT function, and in this case entered in the bit associated with this field.
2	Where necessary, enter the threshold values of the controller output in the Alarms field. This function may be necessary in certain applications for managing process alarms where thresholds are exceeded.
3	Set the Manual mode operating mode. The drop-down list offers several choices: ● Inhibit = no manual mode. ● Authorize = the controller operates in manual mode only. ● Bit address = the value of the bit is used to change the operation of manual mode (bit set to 0 = automatic mode, bit set to 1 = manual mode). In the example: Here we select %M2 to activate the choice, and %MW18 to adjust the value of the manual setpoint.
4	Adjust the Digital output word. This word is used by the controller to send the control setpoint. It can be sent directly to an analog output channel (%QW..) or to a memory word (%MWxx) for additional processing. Important: When using the PWM function, enter a memory address (%MWxx) in this field.

Step	Action
5	Set the PWM output if required by the system: 1. Check the Authorize box if you intend to control the system via a PWM actuator. 2. Enter the PWM control Period in the associated field. 3. Enter the Output used to control the PWM actuator. We recommend you use the base controller transistor outputs for this function (for example, %Q0.0 or %Q0.1 for the TWDLMDA20DRT base controller).
6	Confirm the controller configuration by clicking OK in the bottom left of the screen.
7	To configure several PID controllers, click Next to increment the number of the PID to be set.

PID Configuration Editor

Once you have confirmed the PID configuration, you must then confirm the PID configuration editor, which summarizes all the parameters of each PID configured. To confirm the configuration editor screen, click the **Accept** icon in the toolbar, as shown below:

PID 0 : configured		GENERAL	
Operating mode :	%MW17	INPUT	
Measurement :	%IW1.0	Min :	Max :
Conversion :	Inhibit	Low :	Output :
Alarms :	Inhibit	High :	Output :
Conversion :	Inhibit	PID	
Setpoint :	%MW0	Ti :	%MW11
Kp :	%MW10	Td :	%MW12
Sampling period :	100	AT	
Mode AT :	Authorize	Setpoint :	%MW13
Action :	%M1	Output :	%MW14
Limits :	Inhibit	Min :	Max :
Manual mode :	%M2	Output :	%MW18
Digital output :	%MW15	Period :	%MW16
PWM :	Authorize	Output :	%Q0.1

Step 4 - Initialization of Control Set-Up

Prerequisites for Set-Up

Before set-up, you must follow the steps below:

Step	Action
1	Connect the PC to the controller and transfer the application.
2	Switch the controller to RUN mode.

Note: Before switching the controller to RUN mode, check that the machine's operating conditions allow this for the rest of the application.

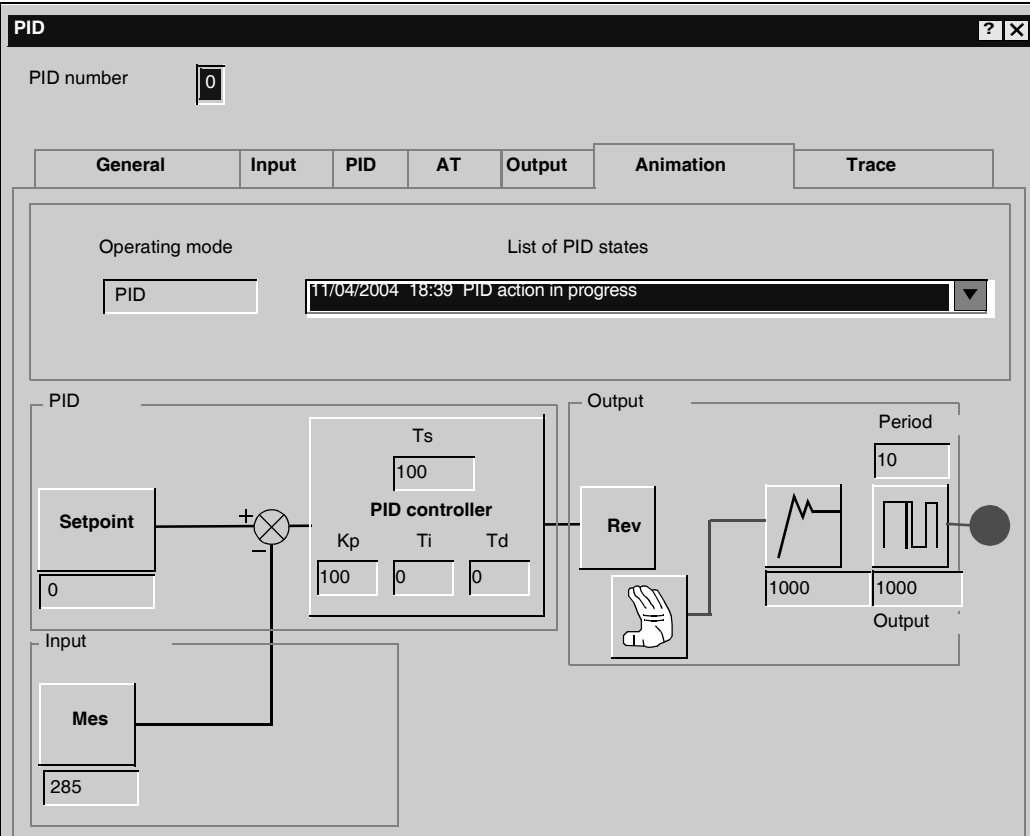
Procedure

The steps below must be followed to initialize control set-up:

Step	Action
1	Create an animation table containing the main objects needed for diagnostics. In the example: ● %MW0: loop controller setpoint, ● %IW1.0: measurement, ● %M0: enabling of loop controller, ● %M1: loop controller action type (set by the AT function), ● %M2: selection of Automatic or Manual mode, ● %MW10 to %MW12: PID loop controller coefficients, ● %MW13: measurement limit not to be exceeded in AT mode, ● %MW14: loop controller output setpoint in AT mode, ● %MW15: discrete output of the PID loop controller (entered by the controller), ● %MW16: setting of the PWM period, ● %MW17: operating mode selection for the PID controller, ● %MW18: manual setpoint associated with the %M2 bit selection.

Step	Action
2	Check the consistency of the value measured in the %IW1.0 field. In the example: 1. A measurement of 248 counts is obtained when the system stable and cold. 2. This seems consistent, as we have a multiplication coefficient of 10 between the temperature and the value read. We can also influence the measurement externally to make sure the reading is consistent (increase the temperature around the probe to check the measurement also increases). Note: This test is quite important, as the operation of the controller depends essentially on the accuracy of the measurement. 3. If you have any doubt about the accuracy of the measurement, set the controller to STOP mode and check the wiring to the inputs of the analog card (voltmeter or ammeter for inputs 0-10V / 4-20mA, ohmmeter for the PT100 (100 ohms at 20°) or Thermocouple (a few tens of ohms): • First disconnect the probe from the analog card terminals. • Check there is no wiring reversal (the colors of the wires connected to the inputs, compensation cable for the PT100). Warning: IN0 and IN1 input channels have a shared potential at the terminals (-). • Check that the analog card is powered by a 24 VDC supply to the first two terminals. • Check that the 4-20 mA input sensors are supplied. The Twido analog input cards are not a source of current.
3	To power up the loop controller, start by controlling the PID controller in Manual mode in order to increase the limit values needed by the AT function. To set the controller to Manual mode: 1. Switch the controller to RUN mode. 2. Enter the memory addresses with the following values in the animation table: • %M2: Manual mode selection = 1, (M2=1 => Manual Mode, M2=0 => Automatic Mode), • %MW16: PWM period setting = 10, • %MW17: Operating mode selection for the PID controller = 1 (PID only), • %MW18: Manual setpoint associated with the %M2 bit selection = 1000. This setpoint value can be selected several times, on condition that the system be left to return to its initial state. In the example: We have selected the value 1000, which corresponds to an average temperature increase value (for information, 2000 counts = 200°). When cold, the system starts at a value of 250 counts.
4	Check that the controller is RUN mode. (%M0: controller validation = 1, to be entered in the animation table.)
5	Double click on the PID item in the configuration browser.

Step	Action
2	Check the consistency of the value measured in the %IW1.0 field. In the example: 1. A measurement of 248 counts is obtained when the system stable and cold. 2. This seems consistent, as we have a multiplication coefficient of 10 between the temperature and the value read. We can also influence the measurement externally to make sure the reading is consistent (increase the temperature around the probe to check the measurement also increases). Note: This test is quite important, as the operation of the controller depends essentially on the accuracy of the measurement. 3. If you have any doubt about the accuracy of the measurement, set the controller to STOP mode and check the wiring to the inputs of the analog card (voltmeter or ammeter for inputs 0-10V / 4-20mA, ohmmeter for the PT100 (100 ohms at 20°) or Thermocouple (a few tens of ohms): • First disconnect the probe from the analog card terminals. • Check there is no wiring reversal (the colors of the wires connected to the inputs, compensation cable for the PT100). Warning: IN0 and IN1 input channels have a shared potential at the terminals (-). • Check that the analog card is powered by a 24 VDC supply to the first two terminals. • Check that the 4-20 mA input sensors are supplied. The Twido analog input cards are not a source of current.
3	To power up the loop controller, start by controlling the PID controller in Manual mode in order to increase the limit values needed by the AT function. To set the controller to Manual mode: 1. Switch the controller to RUN mode. 2. Enter the memory addresses with the following values in the animation table: • %M2: Manual mode selection = 1, (M2=1 => Manual Mode, M2=0 => Automatic Mode), • %MW16: PWM period setting = 10, • %MW17: Operating mode selection for the PID controller = 1 (PID only), • %MW18: Manual setpoint associated with the %M2 bit selection = 1000. This setpoint value can be selected several times, on condition that the system be left to return to its initial state. In the example: We have selected the value 1000, which corresponds to an average temperature increase value (for information, 2000 counts = 200°). When cold, the system starts at a value of 250 counts.
4	Check that the controller is RUN mode. (%M0: controller validation = 1, to be entered in the animation table.)
5	Double click on the PID item in the configuration browser.

Step	Action
6	Activate the Animation tab for the required PID number and check that the animation matches the screen below:  Note: The screens of the PID controller are only refreshed if the controller is enabled (and API set to RUN).
7	Activate the Trace tab for the required PID number, then: 1. Set the time elapse drop-down list to 15 min to see a trace of the measurement signal's progress. 2. Check that the measurement value remains within the acceptable values for the system. The increase in the measurement can be checked in the Trace tab. When this has stabilized, read the value corresponding to the stabilization of the measurement graph (for example, 350 counts corresponding to 35°, or an increase of 10° compared with the initial state).
8	Set the time elapse scroll list to 15min to see a trace of the measurement signal's progress. Check that the measurement value remains within the acceptable values for the system. We can view the increase in the measurement from the Trace tab. When this has stabilized, read the value corresponding to the stabilization of the measurement graph (for example, 350 counts corresponding to 35°, or an increase of 10° compared with the initial state).

Step	Action
9	If we see that the actuator is not controlled, check the output circuit: ● For an analog output, check the output voltage or current from the analog card. ● For a PWM output, check: ● the LED of the output concerned is lit (%Q0.1, in this example), ● the wiring of the supplies and 0V circuit for the TWDLMDA20DRT base outputs, ● the actuator power supply.
10	Close the PID display screen and stop the manual mode by entering the following values in the animation table: ● %M0: Enable loop controller = 0 (stop the loop controller) ● %M2: Selection of Automatic or Manual mode = 0 (stop manual mode) ● %MW17: Operating mode selection for the PID controller = 0 ● %MW18: Manual setpoint associated with %M2 bit selection = 0.

Step 5 - Control Set-Up AT + PID

Introduction

In this section, we will be looking at how to configure the controller to start operation in AT+PID. mode. In this operating mode, the controller will automatically adjust the controller to coefficients Kp, Ti, Td.

Note: During the sequence, the system should not be subject to any disturbance due to external variations that would affect the final adjustments. Also, before launching the AT sequence, make sure the system is stabilized.

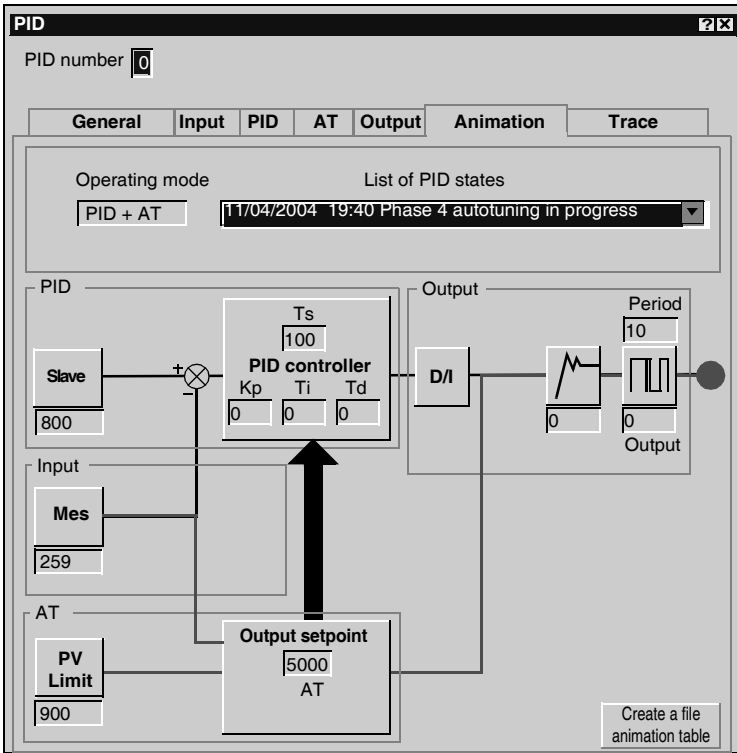
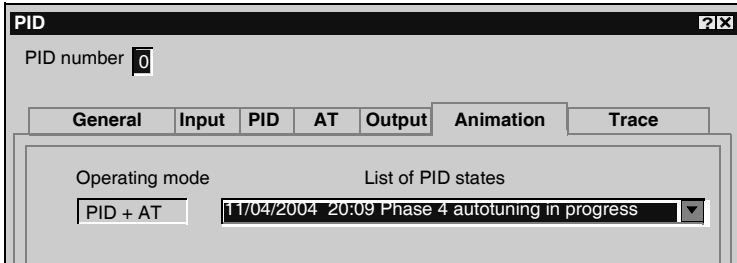
Reminder of Kp, Ti and Td Settings

For operation in AT+PID mode to be possible, the following two conditions must be met:

- The **Kp, Ti, Td** coefficients must be configured as **memory addresses** (%MWxx).
 - The **Action** type in the **Output tab** must be set to a **memory bit address** (%Mxx).
-

To set the controller to AT+PIDmode, proceed as follows:

Step	Action
1	Enter or check the memory addresses with the following values in the animation table: ● %M2: selection of Automatic or Manual mode = 0, ● %MW0: loop controller setpoint = 600 (in this example, the setpoint is active after the AT sequence and the controller maintains a temperature of 60°), ● %MW10 to %MW12: coefficients of the PID controller (leave at 0, the AT sequence will fill them in), ● %MW13: measurement limit not to be exceeded in AT mode = 900 (in the example, if 90° is exceeded an error will occur inAT), ● %MW14: controller output setpoint in AT mode = 2000 (from the test in manual mode). This is the step change value applied to the process. In AT mode, the output setpoint is directly applied at the controller output. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. The value must therefore be between 0 and 10,000. Note: The output autotuning setpoint must always be greater than the last output applied to the process. ● %MW15: discrete output of the PID loop controller (entered by the controller), ● %MW16: PWM period setting (leave 10, as set previously), ● %MW17: operating mode selection for the PID controller = 2 (AT + PID), ● %MW18: manual setpoint associated with the %M2 bit selection = 0.
2	Configure the Twido controller so that it scans in Periodic mode .
3	Set the Time of the Twido controller scanning period so that the Sampling period (Ts) value of the PID controller is an exact multiple. Note: For further details on how to determine the sampling period, see <i>Auto-Tuning Requirements</i>, p. 549 and <i>Methods for Determining the Sampling Period (Ts)</i>, p. 550.
4	Check that the controller is RUN mode.
5	Enter the memory bit %M0. %M0: controller validation = 1 in the animation table.
6	Double click on the PID item in the configuration browser.

Step	Action
7	Activate the Animation tab for the required PID number and check that the animation matches the screen below:  Note: The screens of the PID controller are only refreshed if the controller is enabled (and API set to RUN).
8	Click on the Trace tab and wait for the system to start AT.  Note: The waiting time may last for 10-20 minutes before the AT procedure changes.

**Storage of
Calculated Kp, Ti
and Td
Coefficients**

Once the Auto-Tuning sequence is complete, the memory words assigned to the Kp, Ti and Td coefficients are completed with the calculated values. These values are written to the RAM memory and saved in the controller as long as the application is valid (power-down of less than 30 days) and no cold-start is performed (%S0).

Note: If the system is not influenced by outside fluctuations, the values may be **hard** written in the settings of the PID controller and the controller switched to PID mode only.

Repetition of AT

The Auto-Tuning sequence is repeated on every switch to RUN or cold start (%S0). You should therefore test the diagnostics words using the program what to do in the event of a restart.

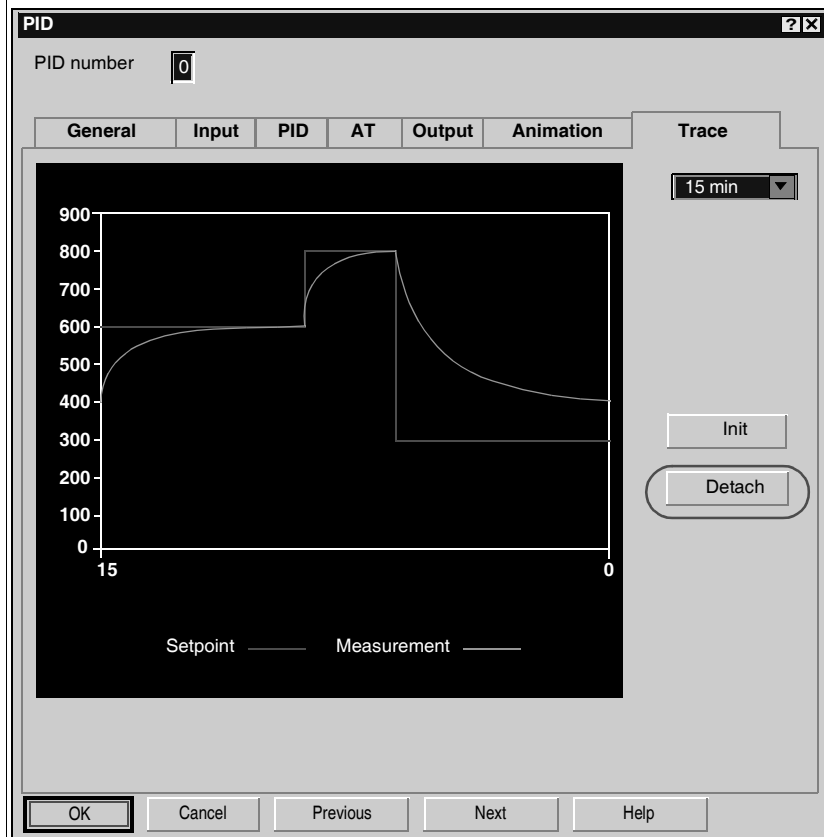
Step 6 - Debugging Adjustments

Accessing the Animation Table

To make it easier to debug the system, the animation table can be accessed at any time when the PID controller screens are in the foreground.

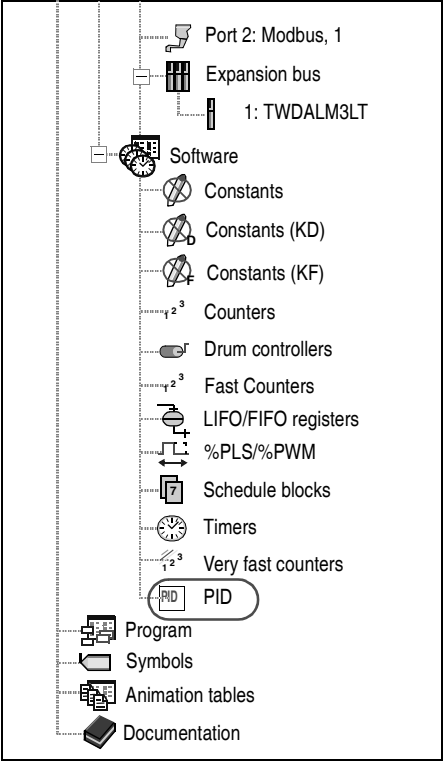
Note:

When viewing only the setpoint and process value graphs using the **Detach** button in the **Trace** tab (see Trace tab window below), the animation table can then be accessed via the menu **Window** → **Animation Tables Editor - Animation...**



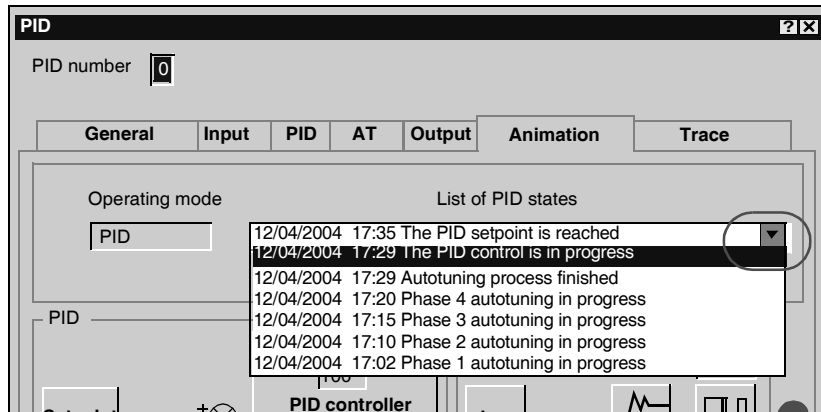
Returning to PID Screens

To return to the PID controller screens without losing the graph trace history, proceed as follows:

Step	Action
1	Double click on the PID item in the browser in the left-hand part of the TwidoSoft screen (see browser window shown below): 
2	When the PID controller window appears, select the desired PID number in the General tab.

History of PID States

In the **Animation** tab for the PID controllers, you can access the last 15 states of the current controller by making your selection from the drop-down list as shown below:



Note: The PID states are stored when the PC and TwidoSoft are in online mode.

17.4 PID Function

At a Glance

Aim of this Section

This section describes the behavior, functionalities and implementation of the PID function.

Note: To find out quick setup information about your PID controller as well as the PID autotuning, please refer to the *Twido PID Quick Start Guide*, p. 490.

What's in this Section?

This section contains the following topics:

Topic	Page
Overview	517
Principal of the Regulation Loop	518
Development Methodology of a Regulation Application	519
Compatibilities and Performances	520
Detailed characteristics of the PID function	521
How to access the PID configuration	524
General tab of PID function	525
Input tab of the PID	528
PID tab of PID function	530
AT tab of PID function	532
Output tab of the PID	537
How to access PID debugging	540
Animation tab of PID function	541
Trace tab of PID function	543
PID States and Errors Codes	545
PID Tuning With Auto-Tuning (AT)	549
PID parameter adjustment method	557
Role and influence of PID parameters	559
Appendix 1: PID Theory Fundamentals	563
Appendix 2: First-Order With Time Delay Model	565

Overview

General

The PID regulation function is a TwidoSoft programming language function. It allows programming of PID regulation loops on controllers compatible with TwidoSoft version 2.0 or higher.

This function is particularly adapted to:

- Answering the needs of the sequential process which need the auxiliary adjustment functions (examples: plastic film packaging machine, finishing treatment machine, presses, etc.)
- Responding to the needs of the simple adjustment process (examples: metal furnaces, ceramic furnaces, small refrigerating groups, etc.)

It is very easy to install as it is carried out in the:

- Configuration
- and Debug

screens associated with a program line (operation block in Ladder Language or by simply calling the PID in Instruction List) indicating the number of the PID used.

Example of a program line in Ladder Language:



Note: In any given Twido automation application, the maximum number of configurable PID functions is 14.

Key Features

The key features are as follows:

- Analog input,
- Linear conversion of the configurable measurement,
- High or low configurable input alarm,
- Analog or PWM output,
- Cut off for the configurable output,
- Configurable direct or inverse action.

Principal of the Regulation Loop

At a Glance

The working of a regulation loop has three distinct phases:

- The acquisition of data:
 - Measurements from the process' sensors (analog, encoders)
 - Setpoint(s) generally from the controller's internal variables or from data from a TwidoSoft animation table
- Execution of the PID regulation algorithm
- The sending of orders adapted to the characteristics of the actuators to be driven via the discrete (PWM) or analog outputs

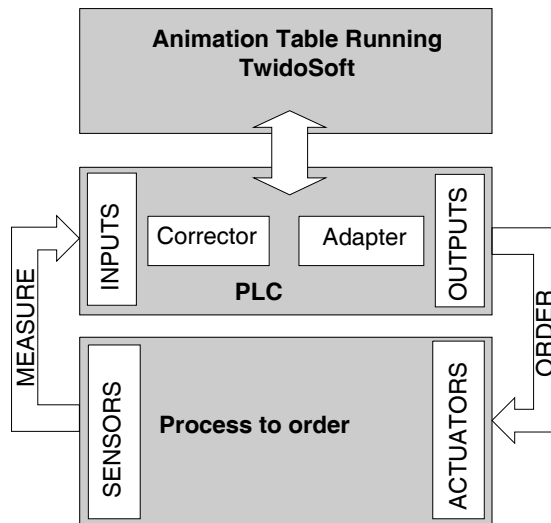
The PID algorithm generates the command signal from:

- The measurement sampled by the input module
- The setpoint value fixed by either the operator or the program
- The values of the different corrector parameters

The signal from the corrector is either directly handled by a controller analog output card linked to the actuator, or handled via a PWM adjustment on a discrete output of the controller.

Illustration

The following diagram schematizes the principal of a regulation loop.

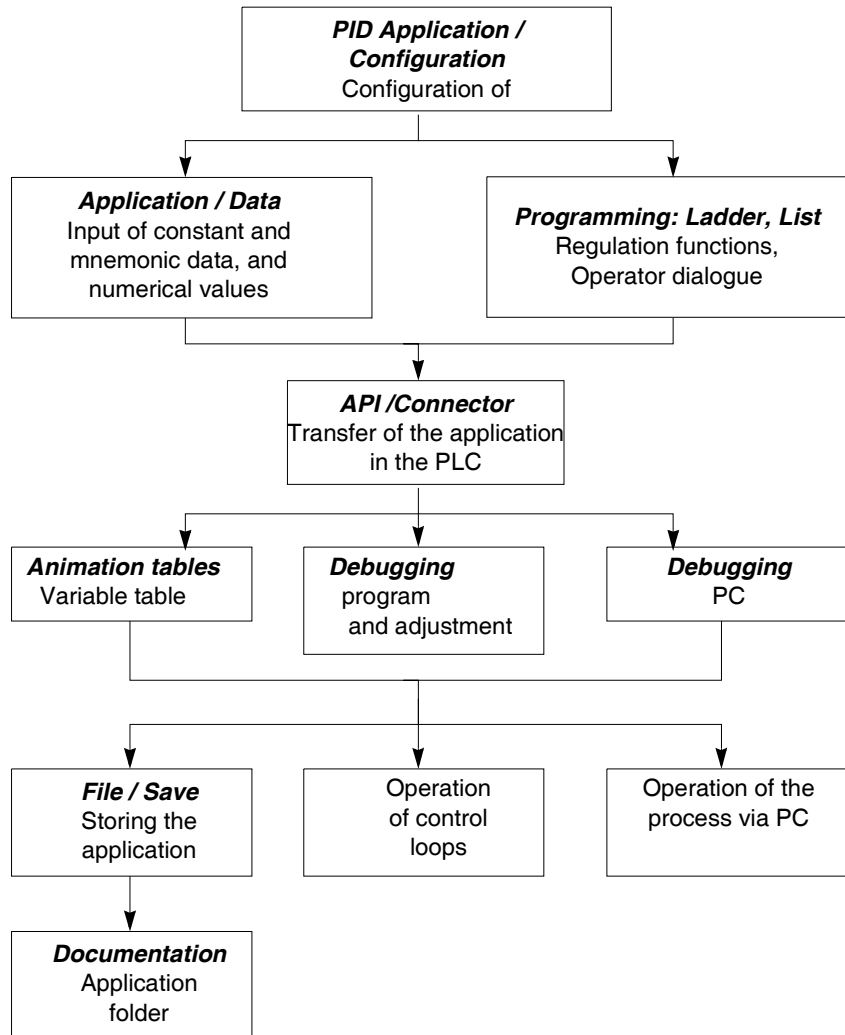


Development Methodology of a Regulation Application

Diagram of the Principal

The following diagram describes all of the tasks to be carried out during the creation and debugging of a regulation application.

Note: The order defined depends upon your own work methods, and is provided as an example.



Compatibilities and Performances

At a Glance The Twido PID function is a function that is available for controllers compatible with TwidoSoft version 2.0 or higher, which is why its installation is subject to a number of hardware and software compatibilities described in the following paragraphs. In addition, this function requires the resources presented in the **Performances** paragraph.

Compatibility The Twido PID function is available on Twido controllers version 2.0 or higher software.
If you have Twidos with an earlier version of the software, you can update your firmware in order to use this PID function.

Note: The version 1.0 analog input/output modules can be used as PID input/output modules without needing to be updated.

In order to configure and program a PID on these different hardware versions, you must have version **2.0 or higher of the TwidoSoft software**.

Performance The PID regulation loops have the following performances:

Description	Time
Loop execution time	0.4 ms

Detailed characteristics of the PID function

General

The PID function completes a PID correction via an analog measurement and setpoint in the default [0-10000] format and provides an analog command in the same format or a Pulse Width Modulation (PWM) on a digital output.

All the PID parameters are explained in the windows used to configure them. Here, we will simply summarize the functions available, indicate measurement values and describe how they integrate into PID in a functional flow diagram.

Note: For use at full scale (optimum resolution), you can configure your analog input connected to the PID's measurement branch in 0-10000 format. However, if you use the default configuration (0-4095), the controller will function correctly.

Note: In order for regulation to operate correctly, it is essential that the Twido PLC **is in periodic mode**. The PID function is then executed periodically on each cycle, and the PID input data sampling complies with the period set in configuration (see table below).

Details of Available Functions

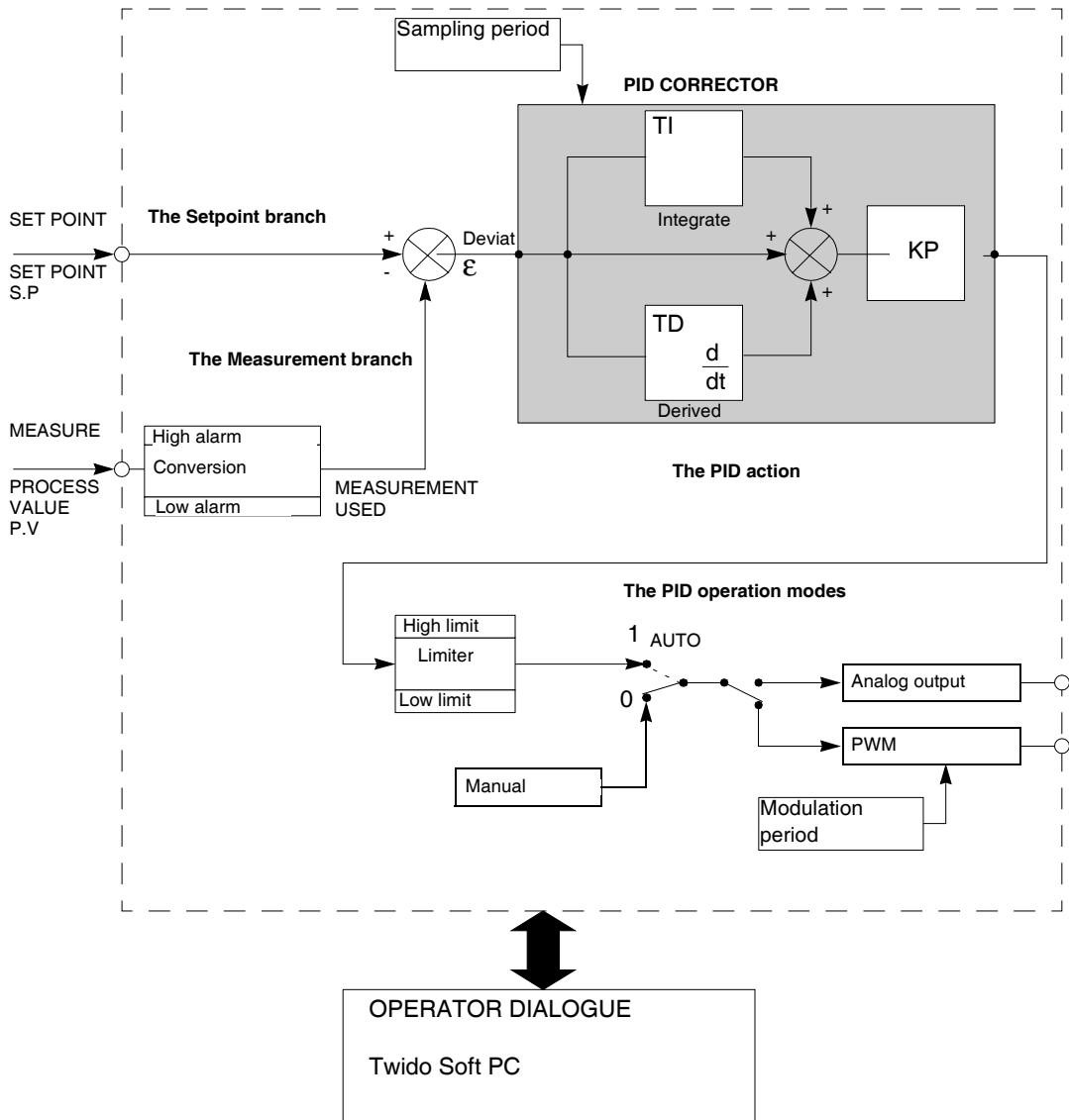
The following table indicates the different functions available and their scale:

Function	Scale and comment
Linear conversion of input	Allows you to convert a value in 0 to 10000 format (analog input module resolution) to a value between -32768 and 32767
Proportional gain	Using a factor of 100, its value is between 1 and 10000. This corresponds to a gain value varying between 0.01 and 100. Note: If you enter an invalid value of gain (negative or null gain), TwidoSoft ignores this user-setting and automatically assigns the default value of 100 to this factor.
Integral time	Using a timebase of 0.1 seconds, its value is between 0 and 20000. This corresponds to an integral time of between 0 and 2000.0 seconds.
Derivative time	Using a timebase of 0.1 seconds, its value is between 0 and 10000. This corresponds to a derivative time of between 0 and 1000.0 seconds.
Sampling period	Using a timebase of 0.01 seconds, its value is between 1 and 10000. This corresponds to a sampling period of between 0.01 and 100 seconds.
PWM output	Using a timebase of 0.1 seconds, its value is between 1 and 500. This corresponds to a modulation period of between 0.1 and 50 seconds.
Analog output	Value between 0 and +10000
High level alarm on process variable	This alarm is set after conversion. It is set to a value between -32768 and 32767 if conversion is activated and to 0 and 10000 if it is not.
Low level alarm on process variable	This alarm is set after conversion. It is set to a value between -32768 and 32767 if conversion is activated and to 0 and 10000 if it is not.
High limit value on output	This limit value is between 0 and 10000 for an analog output value. When PWM is active, the limit corresponds to a percentage of the modulated period. 0% for 0 and 100% for 10000.
Low limit value on output	This limit value is between 0 and 10000 for an analog output value. When PWM is active, the limit corresponds to a percentage of the modulated period. 0% for 0 and 100% for 10000.
Manual mode	When manual mode is active the output is assigned a fixed value set by the user. This output value is between 0 and 10000 (0 to 100% for PWM output).
Direct or inverse action	Direct or inverse is available and acts directly on the output.
Auto-Tuning (AT)	This function provides automatic tuning of the Kp, Ti, Td and Direct/Reverse Action parameters to achieve optimum convergence of the control process.

Note: For a more in-depth explanation of how each of the functions described in the above table works, refer to the diagram below.

Operating Principles

The following diagram presents the operating principle of the PID function.



Note: The parameters used are described in the table on the page above and in the configuration screens.

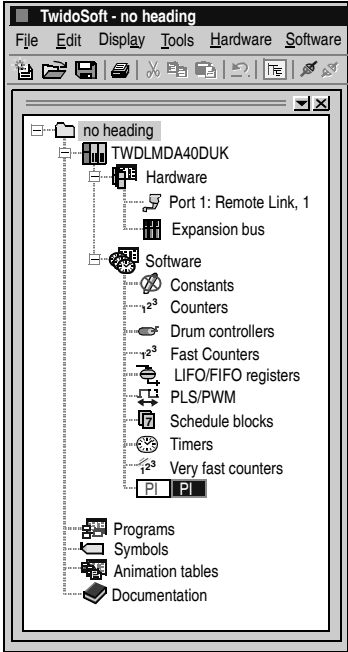
How to access the PID configuration

At a Glance

The following paragraphs describe how to access the PID configuration screens on TWIDO controllers.

Procedure

The following table describes the procedure for accessing the PID configuration screens:

Step	Action
1	Check that you are in offline mode.
2	Open the browser. Result: 
3	Double-click on PID. Result: The PID configuration window opens and displays the General (See <i>General tab of PID function</i>, p. 525) tab by default. Note: You can also right-click on PID and select the Edit option or select Software → PID from the menu or use the Program → Configuration Editor → PID Icon menu or, if using the latter method, select the PID and click on the Magnifying glass icon to select a specific PID.

General tab of PID function

At a Glance

When you open PID from the browser, you open the PID configuration window. This window allows you to:

- configure each TWIDO PID,
- debug each TWIDO PID,

When you open this screen, if you are:

- in offline mode: you will go to the **General** tab by default and will have access to the configuration parameters,
- in online mode: you will go to the **Animation** tab and will have access to the debugging and adjustment parameters.

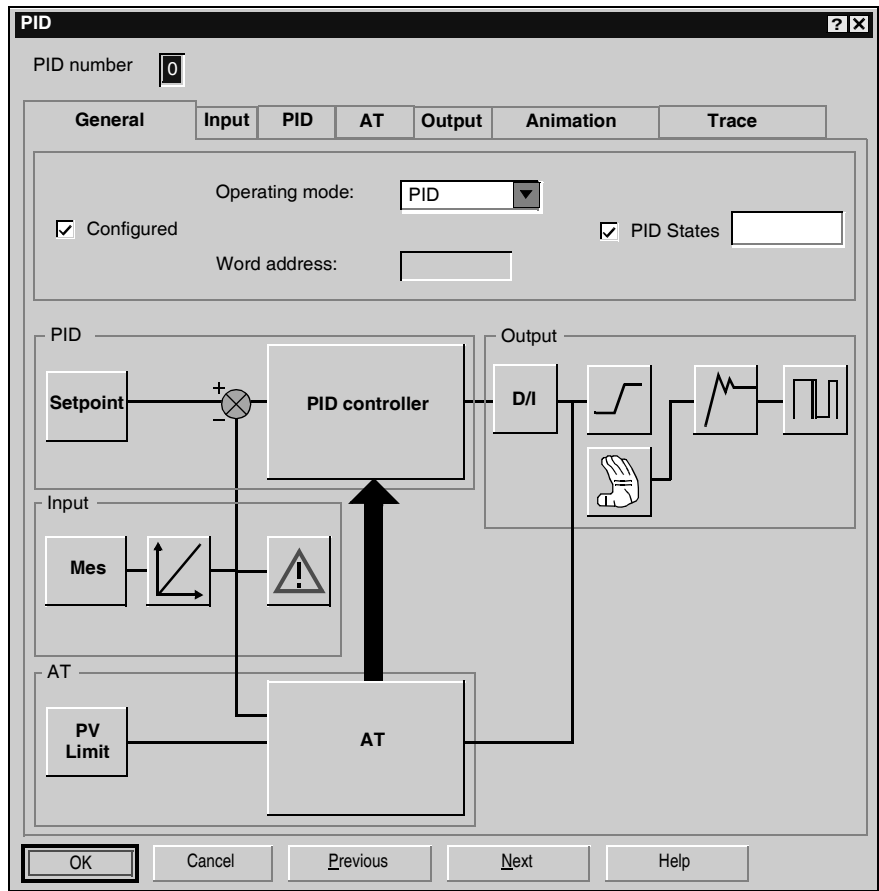
Note: In some cases, the grayed-out tabs and fields may not be accessible for any of the two reasons listed below: The "PID only" operating mode is selected, which prevents access to the AT tab parameters that are no longer needed.

- The operating mode (offline or online) which is currently active does not allow you to access these parameters.
- The "PID only" operating mode is selected, which prevents access to the AT tab parameters that are no longer needed.

The following paragraphs describe the **General** tab.

General Tab of the PID Function

The screen below is used to enter the general PID parameters.



Description

The table below describes the settings that you may define.

Field	Description
PID number	Specify the PID number that you wish to configure here. The value is between 0 and 13, 14 PID maximum per application.
Configured	To configure the PID, this box must be checked. Otherwise no action can be performed in these screens and the PID, though it exists in the application, cannot be used.
Operating mode	Specify the desired operating mode here. You may choose from three operating modes and a word address, as follows: ● PID ● AT ● AT+PID ● Word address
Word address	You may provide an internal word in this text box (%MW0 to %MW2999) that is used to programmatically set the operating mode. The internal word can take three possible values depending on the operating mode you wish to set: ● %MWx = 1 (to set PID only) ● %MWx = 2 (to set AT + PID) ● %MWx = 3 (to set AT only)
PID States	If you check to enable this option, you may provide a memory word in this text box (%MW0 to %MW2999) that is used by the PID controller to store the current PID state while running the PID controller and/or the autotuning function (for more details, please refer to <i>PID States and Errors Codes</i> , p. 545.)
Diagram	The diagram allows you to view the different possibilities available for configuring your PID.

Input tab of the PID

At a Glance

The tab is used to enter the PID input parameters.

Note: It is accessible in offline mode.

Input tab of the PID Function

The screen below is used to enter the PID input parameters.

PID

PID number 0

General

Input

PID

AT

Output

Animation

Trace

Measure

%IW1.0

Conversion

☐ Authorize

Min value:

Max value:

Alarms

☐ Authorize

Low:

High:

Output:

Output:

PID

Setpoint

+

×

PID controller

Input

Mes

Output

D/I

OK

Cancel

Previous

Next

Help

Description

The table below describes the settings that you may define.

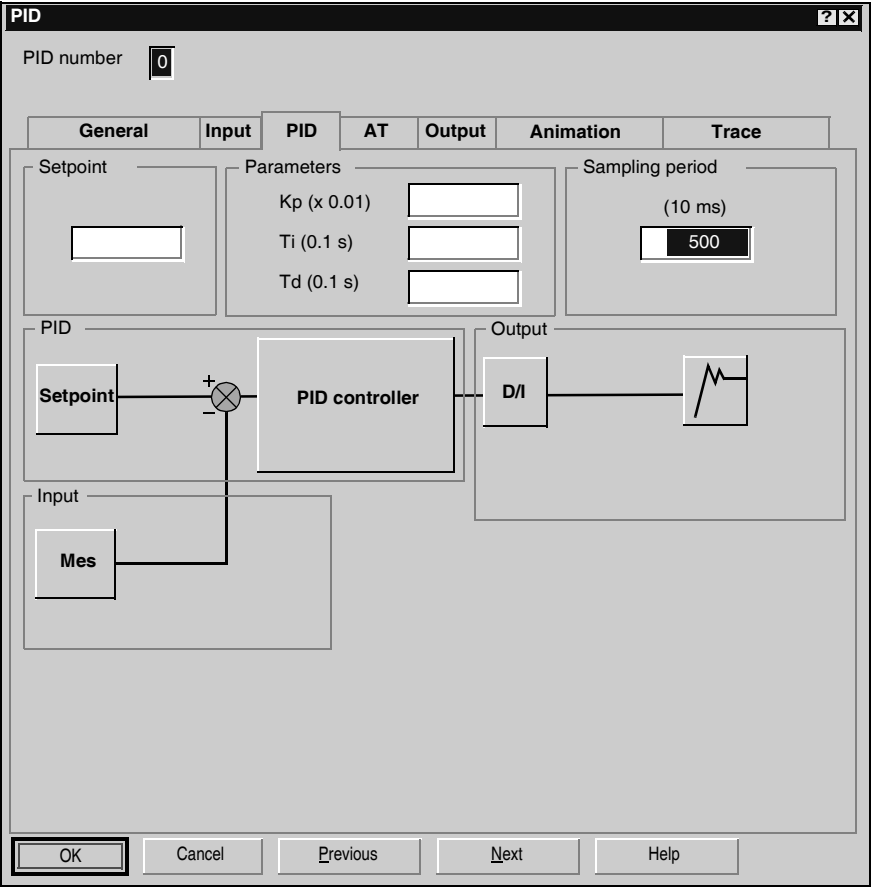
Field	Description
PID number	Specify the PID number that you wish to configure here. The value is between 0 and 13, 14 PID maximum per application.
Measurement	Specify the variable that will contain the process value to be controlled here. The default scale is 0 to 10000. You can enter either an internal word (%MW0 to %MW2999) or an analog input (%IWx.0 to %IWx.1).
Conversion	Check this box if you wish to convert the process variable specified as a PID input. If this box is checked, both the Min value and Max value fields are accessible. The conversion is linear and converts a value between 0 and 10,000 into a value for which the minimum and maximum are between -32768 and +32767.
Min value Max value	Specify the minimum and maximum of the conversion scale. The process variable is then automatically rescaled within the [Min value to Max value] interval. Note: The Min value must always be less than the Max value . Min value or Max value can be internal words (%MW0 to %MW2999), internal constants (%KW0 to %KW255) or a value between -32768 and +32767.
Alarms	Check this box if you wish to activate alarms on input variables. Note: The alarm values should be determined relative to the process variable obtained after the conversion phase. They must therefore be between Min value and Max value when conversion is active. Otherwise they will be between 0 and 10000 .
Low Output	Specify the high alarm value in the Low field. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. Output must contain the address of the bit which will be set to 1 when the lower limit is reached. Output can be either an internal bit (%M0 to %M255) or an output (%Qx.0 to %Qx.32).
High Output	Specify the low alarm value in the High field. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. Output must contain the address of the bit which will be set to 1 when the upper limit is reached. Output can be either an internal bit (%M0 to %M255) or an output (%Qx.0 to %Qx.32).
Diagram	The diagram allows you to view the different possibilities available for configuring your PID.

PID tab of PID function

At a Glance The tab is used to enter the internal PID parameters.

Note: It is accessible in offline mode.

PID tab of the PID Function The screen below is used to enter the internal PID parameters.



Description

The table below describes the settings that you may define.

Field	Description
PID number	Specify the PID number that you wish to configure here. The value is between 0 and 13, 14 PID maximum per application.
Setpoint	Specify the PID setpoint value here. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. This value must therefore be between 0 and 10000 when conversion is inhibited. Otherwise it must be between the Min value and the Max value for the conversion.
Kp * 100	Specify the PID proportional coefficient multiplied by 100 here. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. The valid range for the Kp parameter is: $0 < Kp < 10000$. Note: If Kp is mistakenly set to 0 ($Kp \leq 0$ is invalid), the default value $Kp=100$ is automatically assigned by the PID function.
Ti (0.1 sec)	Specify the integral action coefficient here for a timebase of 0.1 seconds. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. It must be between 0 and 20000. Note: To disable the integral action of the PID, set this coefficient to 0.
Td (0.1 sec)	Specify the derivative action coefficient here for a timebase of 0.1 seconds. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. It must be between 0 and 10000. Note: To disable the derivative action of the PID, set this coefficient to 0.
Sampling period	Specify the PID sampling period here for a timebase of 10^{-2} seconds (10 ms). This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. It must be between 1 (0.01 s) and 10000 (100 s).
Diagram	The diagram allows you to view the different possibilities available for configuring your PID.

Note: When AT is enabled, Kp, Ti and Td parameters are no longer set by the user for they are automatically and programmatically set by the AT algorithm. In this case, you must enter in these fields an **internal word** only (%MW0 to %MW2999).
Caution: Do not enter an internal constant or a direct value when AT is enable, for this will trigger an error when running your PID application.

AT tab of PID function

At a Glance

The setting of correct PID parameters may be tedious, time-consuming and error-prone. All these make process control difficult to setup for the yet experienced, but not necessarily process control professional user. Thus, optimum tuning may sometimes be difficult to achieve.

The PID Auto-Tuning algorithm is designed to determine automatically and adequately the following four PID terms:

- Gain factor,
- Integral value,
- Derivative value, and
- Direct or Reverse action.

Thus, the AT function can provide rapid and optimum tuning for the process loop.

AT Requirements

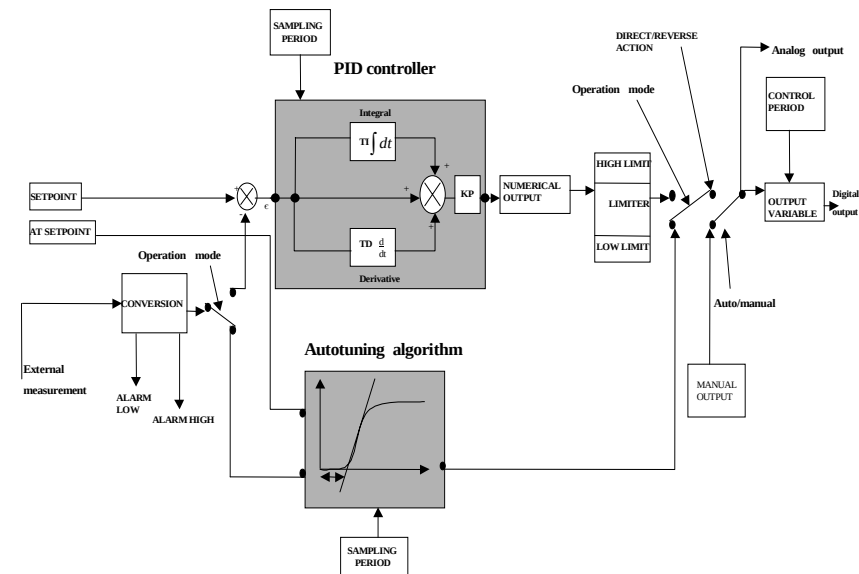
PID Auto-tuning is particularly suited for temperature control processes.

In a general manner, the processes that the AT function can be used to control must meet the following requirements:

- the process is mostly linear over the entire operating range,
 - the process response to a level change of the analog output follows a transient asymptotic pattern, and
 - there is little disturbance in process variables. (In the case of a temperature control process, this implies there is no abnormally high rate of heat exchange between the process and its environment.)
-

AT Operating Principle

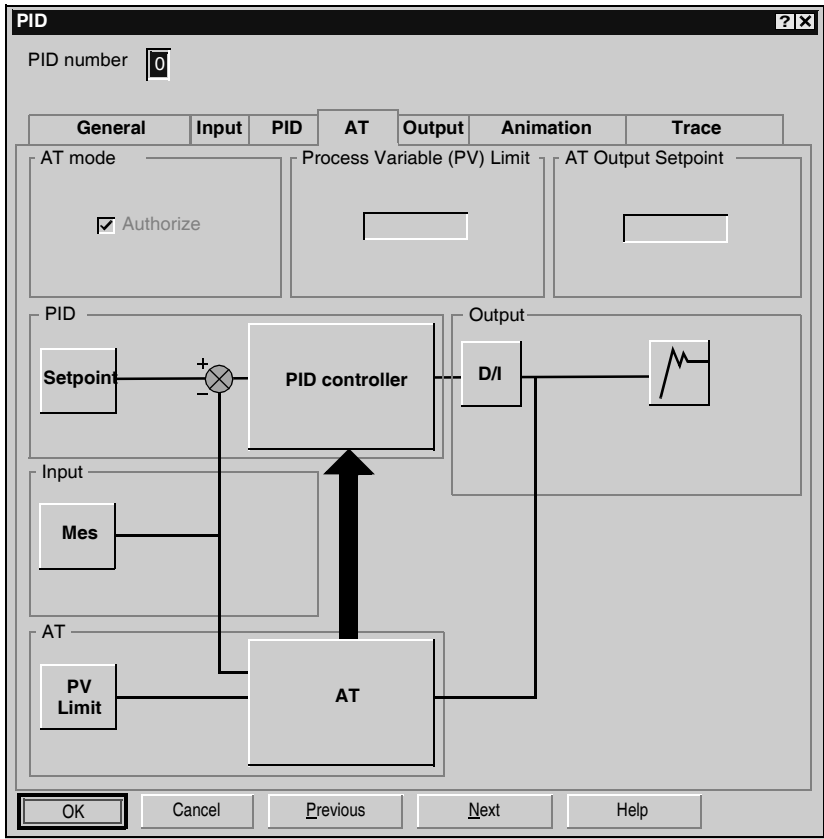
The following diagram describes the operating principle of the AT function and how it interacts with the PID loops:



AT Tab of the PID function

The screen below is used to enable/disable the AT function and enter the AT parameters.

Note: It is accessible in offline mode only.



Description

 WARNING
THE PROCESS VARIABLE (PV) LIMIT AND THE OUTPUT SETPOINT VALUES MUST BE SET CAREFULLY.

PID Auto-Tuning is an open-loop process that is acting directly on the control process without regulation or any limitation other than provided by the Process Variable (PV) Limit and the Output Setpoint. Therefore, both values must be carefully selected within the allowable range as specified by the process to prevent potential process overload.

Failure to follow this instruction can result in death, serious injury, or equipment damage.

The table below describes the settings that you may define.

Field	Description
Authorize	Check this box if you wish to enable the AT mode. There are two ways to use this checkbox, depending on whether you set the operating mode manually or via a word address in the General tab of the PID function: ● If you set the Operating mode to PID+AT or AT from the General tab (see <i>General tab of PID function, p. 525</i>), then the Authorize option is automatically checked and grayed out (it cannot be unchecked). ● If you set the operating mode via a word address %MWx (%MWx = 2: PID+AT; %MWx = 3: AT), then you must check the Authorize option manually to allow configuring the AT parameters. Result: In either of the above cases, all the fields in this AT tab configuration screen become active and you must fill in the Setpoint and Output fields with the appropriate values.
Process Variable (PV) Limit	Specify the limit that the measured process variable shall not exceed during the AT process. This parameter provides safety to the control system, as AT is an open loop process. This value can be an internal word (%MW0 to a maximum of %MW2999, depending on amount of system memory available), an internal constant (%KW0 to %KW255) or a direct value. This value must therefore be between 0 and 10000 when conversion is inhibited. Otherwise it must be between the Min value and the Max value for the conversion.

Field	Description
AT Output setpoint	Specify the AT output value here. This is the value of the step-change that is applied to the process. This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value. This value must therefore be between 0 and 10000. Note: The AT Output Setpoint must always be larger than the output last applied to the process.

Note: When the AT function is enabled, constants (%KWx) or direct values are no longer allowed, only memory words are allowed in the following set of PID fields:

- **Kp, Ti** and **Td** parameters must be set as **memory words** (%MWx) in the PID tab;
- **Action** field is automatically set to "**Address bit**" in the OUT tab;
- **Bit** box must be filled in with an adequate **memory bit** (%Mx) in the OUT tab.

Calculated Kp, Ti, Td Coefficients

Once the AT process is complete, the calculated Kp, Ti and Td PID coefficients:

- are stored in their respective memory words (%MWx), and
- can be viewed in the **Animation** tab, in TwidoSoft online mode only.

Output tab of the PID

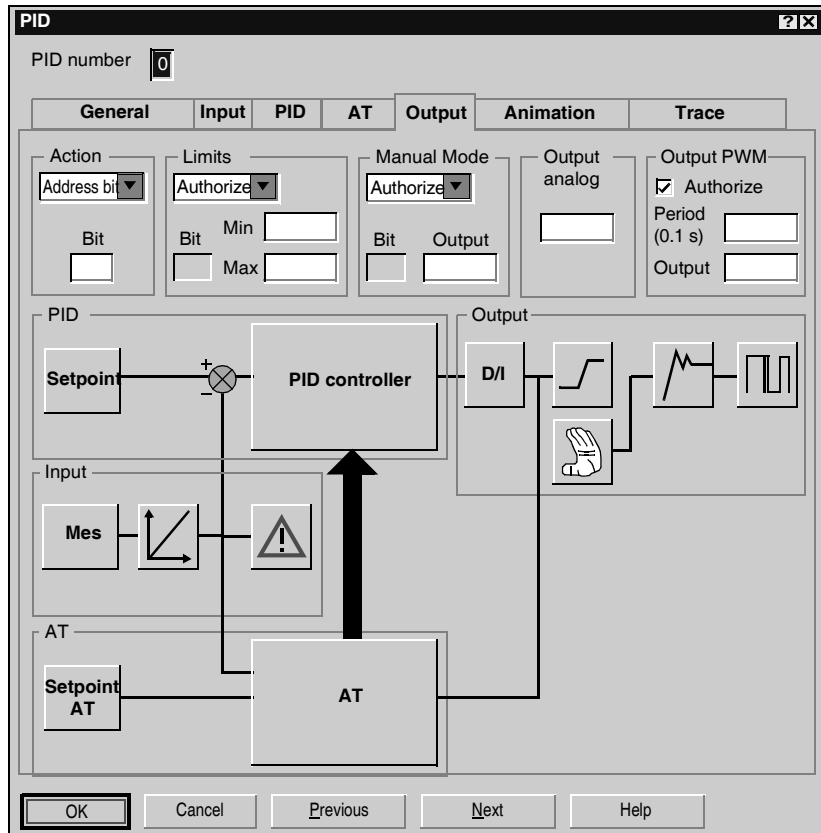
At a Glance

The tab is used to enter the PID output parameters.

Note: It is accessible in offline mode.

Output Tab of the PID Function

The screen below is used to enter the internal PID parameters.



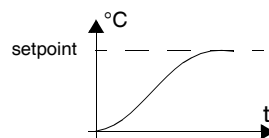
Description

The table below describes the settings that you may define.

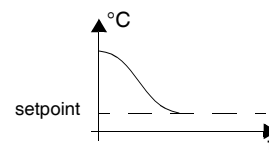
Field	Description
PID number	Specify the PID number that you wish to configure here. The value is between 0 and 13, 14 PID maximum per application.
Action	Specify the type of PID action on the process here. Three options are available: Reverse , Direct or bit address . If you have selected bit address , you can modify this type via the program, by modifying the associated bit which is either an internal bit (%M0 to %M255) or an input (%Ix.0 to %Ix.32). Action is direct if the bit is set to 1 and reverse if it is not. Note: When AT is enabled, the Auto-Tuning algorithm automatically determines the correct type of action direct or reverse for the control process. In this case, only one option is available from the Action dropdown list: Address bit . You must then enter in the associated Bit textbox an internal word (%MW0 to %MW2999). Do not attempt to enter an internal constant or a direct value in the Bit textbox, for this will trigger an execution error.
Limits Bit	Specify here whether you want to place limits on the PID output. Three options are available: Enable , Disable or bit address . If you have selected bit address , you can enable (bit to 1) or disable (bit to 0) limit management by the program, by modifying the associated bit which is either an internal bit (%M0 to %M255) or an input (%Ix.0 to %Ix.32).
Min. Max.	Set the high and low limits for the PID output here. Note: The Min. must always be less than the Max. . Min. or Max. can be internal words (%MW0 to %MW2999), internal constants (%KW0 to %KW255) or a value between 1 and 10000.
Manual mode Bit Output	Specify here whether you want to change the PID to manual mode. Three options are available: Enable , Disable or bit address . If you have selected bit address , you can switch to manual mode (bit to 1) or switch to automatic mode (bit to 0) using the program, by modifying the associated bit which is either an internal bit (%M0 to %M255) or an input (%Ix.0 to %Ix.32). The Output of manual mode must contain the value that you wish to assign to the analog output when the PID is in manual mode. This Output can be either a word (%MW0 to %MW2999) or a direct value in the [0-10000] format.
Analog output	Specify the PID output in auto mode here. This Analog output can be %MW-type (%MW0 to %MW2999) or %QW-type (%QWx.0).

Field	Description
PWM output enabled Period (0.1s) Output	Check this box if you want to use the PWM function of PID. Specify the modulation period in Period (0.1s) . This period must be between 1 and 500 and can be an internal word (%MW0 to %MW2999) or an internal constant (%KW0 to %KW255). Specify the PWM output bit as the value in Output . This can be either an internal bit (%M0 to %M255) or an output (%Qx.0 to %Qx.32).
Diagram	The diagram allows you to view the different possibilities available for configuring your PID.

Note: The term Reverse in the action field is used to reach a high setpoint (e.g.: for heating)



The term Direct in the action field is used to reach a low setpoint (e.g.: for cooling)



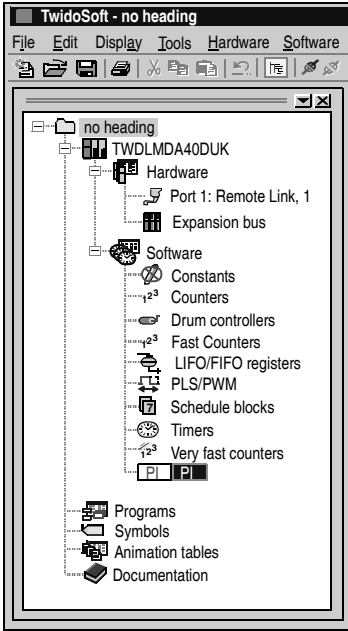
How to access PID debugging

At a Glance

The following paragraphs describe how to access the PID debugging screens on TWIDO controllers.

Procedure

The following table describes the procedure for accessing the PID debugging screens:

Step	Action
1	Check that you are in online mode.
2	Open the browser. Result: 
3	Double-click on PID. Result: The PID configuration window opens and displays the Animation (See <i>Animation tab of PID function, p. 541</i>) tab by default. Note: You can also right-click on PID and select the Edit option or select Software → PID from the menu or use the Program → Configuration Editor → PID Icon menu or, if using the latter method, select the PID and click on the Magnifying glass icon to select a specific PID.

Animation tab of PID function

At a Glance

The tab is used to debug the PID.

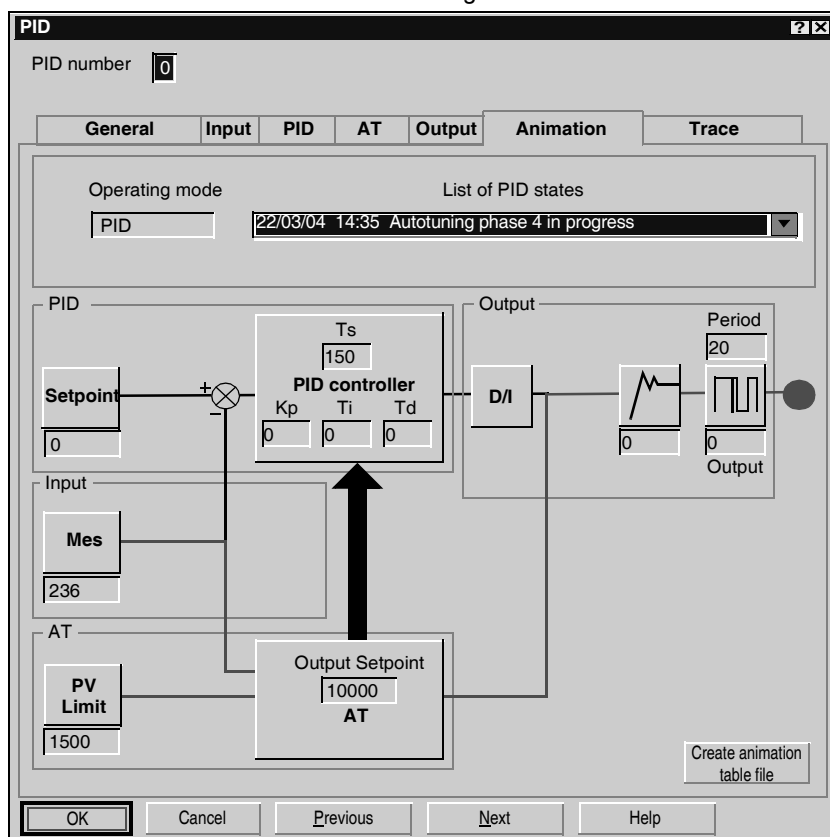
The diagram depends on the type of PID control that you have created. Only configured elements are shown.

The display is dynamic. Active links are shown in red and inactive links are shown in black.

Note: It is accessible in online mode.

Animation Tab of PID Function

The screen below is used to view and debug the PID.



Description

The following table describes the different zones of the window.

Field	Description
PID number	Specify the PID number that you wish to debug here. The value is between 0 and 13, 14 PID maximum per application.
Operating mode	This field shows the current PID operating mode.
List of PID states	This dropdown list allows you to view the last 15 PID states in real time. This list is updated with each change of state indicating the date and time of the change as well as the current state.
Create an Animation Table	Click on Create an Animation Table , to create a file containing all the variables shown in the diagram to enable you modify them online and debug your PID.

Trace tab of PID function

At a Glance

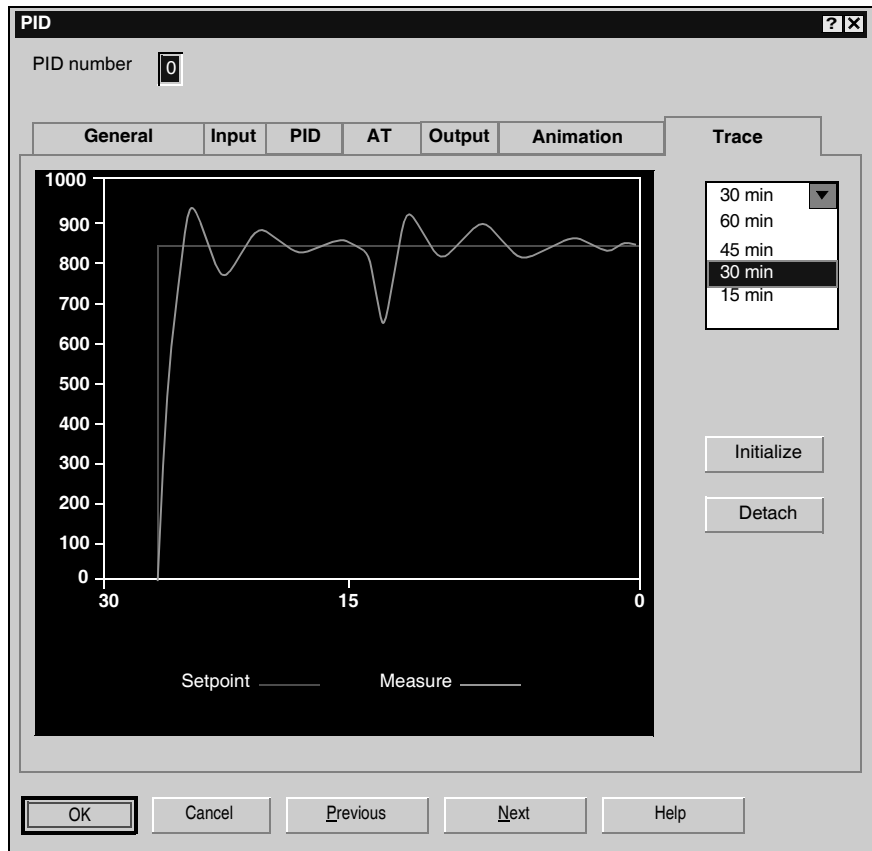
This tab allows you to view PID operation and to make adjustments to the way it behaves.

The graphs begin to be traced as soon as the debug window is displayed.

Note: It is accessible in online mode.

Animation Tab of PID Function

The screen below is used to view the PID control.



Description

The following table describes the different zones of the window.

Field	Description
PID number	Specify the PID number that you wish to view here. The value is between 0 and 13, 14 PID maximum per application.
Chart	This zone displays the setpoint and process value graphs. The scale on the horizontal axis (X) is determined using the menu to the top right of the window. The scale on the vertical axis is determined using the PID input configuration values (with or without conversion). It is automatically optimized so as to obtain the best view of the graphs.
Horizontal axis scale menu	This menu allows you to modify the scale of the horizontal axis. You can choose from 4 values: 15, 30, 45 or 60 minutes.
Initialize	This button clears the chart and restarts tracing the graphs.

PID States and Errors Codes

At a Glance

In addition to the **List of PID States** available from the **Animation** dialog box (see *Animation tab of PID function, p. 541*) that allows to view and switch back to one of the 15 latest PID states, the Twido PID controller also has the ability to record the current state of both the PID controller and the AT process to a user-defined memory word.

To find out how to enable and configure the **PID state memory word** (%MWi) see *General tab of PID function, p. 525*.

PID State Memory Word

The PID state memory word can record any of three types of PID information, as follows:

- Current state of the PID controller (PID State)
- Current state of the autotuning process (AT State)
- PID and AT error codes

Note: The PID state memory word is read-only.
--

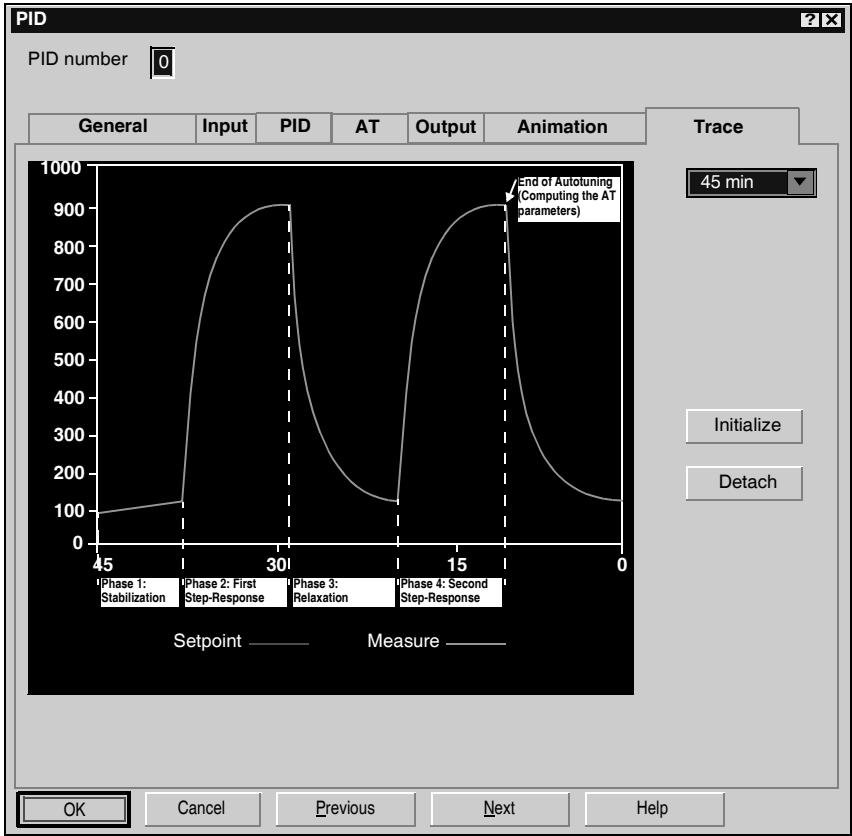
PID State Memory Word

The following is the PID controller state versus memory word hexadecimal coding concordance table:

PID State hexadecimal notation	Description
0000h	PID control is not active
2000h	PID control in progress
4000h	PID setpoint has been reached

Description of AT State

The autotuning process is divided into 4 consecutive phases. Each phase of the process must be fulfilled in order to bring the autotuning to a successful completion. The following process response curve and table describe the 4 phases of the Twido PID autotuning:



The autotuning phases are described in the following table:

AT Phase	Description
1	Phase 1 is the stabilization phase. It starts at the time the user launches the AT process. During this phase, the Twido autotuning performs checks to ensure that the process variable is in steady state. Note: The output last applied to the process before start of the autotuning is used as both the starting point and the relaxation point for the autotuning process.
2	Phase 2 applies the first step-change to the process. It produces a process step-response similar to the one shown in the above figure.

AT Phase	Description
3	Phase 3 is the relaxation phase that starts when the first step-response has stabilized. Note: Relaxation occurs toward equilibrium that is determined as the output last applied to the process before start of the autotuning.
4	Phase 4 applies the second step-change to the process in the same amount and manner as in Phase 2 described above. The autotuning process ends and the AT parameters are computed and stored in their respective memory words upon completion of Phase 4. Note: After this phase is complete, the process variable is restored to the output level last applied to the process before start of the autotuning.

AT State Memory Word

The following is the PID controller state versus memory word hexadecimal coding concordance table:

AT State hexadecimal notation	Description
0100h	Autotuning phase 1 in progress
0200h	Autotuning phase 2 in progress
0400h	Autotuning phase 3 in progress
0800h	Autotuning phase 4 in progress
1000h	Autotuning process complete

PID and AT Error Codes

The following table describes the potential execution errors that may be encountered during both PID control and autotuning processes:

PID/AT Processes	Error code (hexadecimal)	Description
PID Error	8001h	Operating mode value out of range
	8002h	Linear conversion min and max equal
	8003h	Upper limit for digital output lower than lower limit
	8004h	Process variable limit out of linear conversion range
	8005h	Process variable limit less than 0 or greater than 10000
	8006h	Setpoint out of linear conversion range
	8007h	Setpoint less than 0 or greater than 10000
	8008h	Control action different from action determined at AT start
Autotuning Error	8009h	Autotuning error: the process variable (PV) limit has been reached
	800Ah	Autotuning error : due to either oversampling or output setpoint too low
	800Bh	Autotuning error: Kp is zero
	800Ch	Autotuning error: the time constant is negative
	800Dh	Autotuning error: delay is negative
	800Eh	Autotuning error: error calculating Kp
	800Fh	Autotuning error: time constant over delay ratio > 20
	8010h	Autotuning error: time constant over delay ratio < 2
	8011h	Autotuning error: the limit for Kp has been exceeded
	8012h	Autotuning error: the limit for Ti has been exceeded
	8013h	Autotuning error: the limit for Td has been exceeded

PID Tuning With Auto-Tuning (AT)

Overview of PID Tuning

The PID control function relies on the following three user-defined parameters: K_p , T_i and T_d . PID tuning aims at determining these process parameters accurately to provide optimum control of the process.

Scope of the Auto-Tuning

The AT function of the Twido PLC is especially suited for automatic tuning of thermal processes. As values of the PID parameters may vary greatly from one control process to another, the auto-tuning function provided by the Twido PLC can help you determine more accurate values than simply provided by best guesses, with less effort.

Auto-Tuning Requirements

When using the auto-tuning function, make sure the control process and the Twido PLC meet all of the following four requirements:

- The control process must be an open-loop, stable system.
- At the start of the auto-tuning run, the control process must be in steady state with a null process input (e.g.: an oven or a furnace shall be at ambient temperature.)
- During operation of the auto-tuning, make sure that no disturbances enter through the process for either computed parameters will be erroneous or the auto-tuning process will simply fail (e.g.: the door of the oven shall not be opened, not even momentarily.)
- Configure the Twido PLC to scan in **Periodic mode**. Once you have determined the correct sampling period (T_s) for the auto-tuning, the scan period must be configured so that the sampling period (T_s) is an exact multiple of the Twido PLC scan period.

Note: To ensure a correct run of the PID control and of the auto-tuning process, it is essential that the Twido PLC be configured to execute scans in Periodic mode (not Cyclic). In Periodic mode, each scan of the PLC starts at regular time intervals. This way, the sampling rate is constant throughout the measurement duration (unlike cyclic mode where a scan starts as soon as the previous one ends, which makes the sampling period unbalanced from scan to scan.)

AT Operating Modes

The auto-tuning can be used either independently (AT mode) or in conjunction with the PID control (AT + PID):

- **AT mode:** After convergence of the AT process and successful completion with the determination of the PID control parameters Kp, Ti and Td (or after detection of an error in the AT algorithm), the AT numerical output is set to 0 and the following message appears in the **List of PID States** drop-down list: "Auto-tuning complete."
- **AT + PID mode:** The AT is launched first. After successful completion of the AT, the PID control loop starts (based on the Kp, Ti and Td parameters computed by the AT)."

Note on AT+PID: If the AT algorithm encounters an error:

- no PID parameter is computed;
- the AT numerical output is set to output last applied to the process before start of the autotuning;
- an error message appears in the List of PID States drop-down list
- the PID control is cancelled.

Note: Bumpless transition

While in **AT+PID mode**, the transition from AT to PID is bumpless.

Methods for Determining the Sampling Period (Ts)

As will be explained in the two following sections (see *Appendix 1: PID Theory Fundamentals*, p. 563 and *Appendix 2: First-Order With Time Delay Model*, p. 565), the **sampling period (Ts)** is a key parameter of the PID control. The sampling period can be deduced from the AT **time constant (τ)**.

There are two methods for evaluating the correct sampling period (Ts) by using the auto-tuning: They are described in the following sections.

- The process response curve method
- The trial-and-error method

Both methods are described in the two following subsections.

Introducing the Process Response Curve Method

This method consists in setting a step change at the control process input and recording the process output curve with time.

The process response curve method makes the following assumption:

- The control process can be adequately described as a first-order with time delay model by the following transfer function:

$$\frac{S}{U} = \frac{k}{1 + \tau p} \cdot e^{-\theta p}$$

(For more details, see Appendix 2: First-Order With Time Delay Model)

Using the Process Response Curve Method

To determine the sampling period (Ts) using the process response curve method, follow these steps:

Step	Action
1	It is assumed that you have already configured the various settings in the General, Input, PID, AT and Output tabs of the PID.
2	Select the PID > Output tab from the Application Browser.
3	Select Authorize or Address bit from the Manual mode dropdown list to allow manual output and set the Output field to a high level (in the [5000-10000] range).
4	Select PLC > Transfer PC => PLC... from menu bar to download the application program to the Twido PLC.
5	Within the PID configuration window, switch to Trace mode.
6	Run the PID and check the response curve rise.
7	When the response curve has reached a steady state, stop the PID measurement. Note: Keep the PID Trace window active.
8	Use the following graphical method to determine time constant (τ) of the control process: 1. Compute the process variable output at 63% rise ($S_{[63\%]}$) by using the following formula: $S_{[63\%]} = S_{[initial]} + (S_{[ending]} - S_{[initial]}) \times 63\%$ 2. Find out graphically the time abscissa ($t_{[63\%]}$) that corresponds to $S(63\%)$. 3. Find out graphically the initial time ($t_{[initial]}$) that corresponds the start of the process response rise. 4. Compute the time constant (τ) of the control process by using the following relationship: $\tau = t_{[63\%]} - t_{[initial]}$
9	Compute the sampling period (Ts) based the value of (τ) that you have just determined in the previous step, using the following rule: $Ts = \tau/75$ Note: The base unit for the sampling period is 10ms. Therefore, you should round up/down the value of Ts to the nearest 10ms.
10	Select Program > Scan mode edit and proceed as follows: 1. Set the Scan mode of the Twido PLC to Periodic. 2. Set the Scan Period so that the sampling period (Ts) is an exact multiple of the scan period, using the following rule: $\text{Scan Period} = Ts / n$, where "n" is a positive integer. Note: You must choose "n" so that the resulting Scan Period is a positive integer in the range [2 - 150 ms].

**Example of
Process
Response Curve**

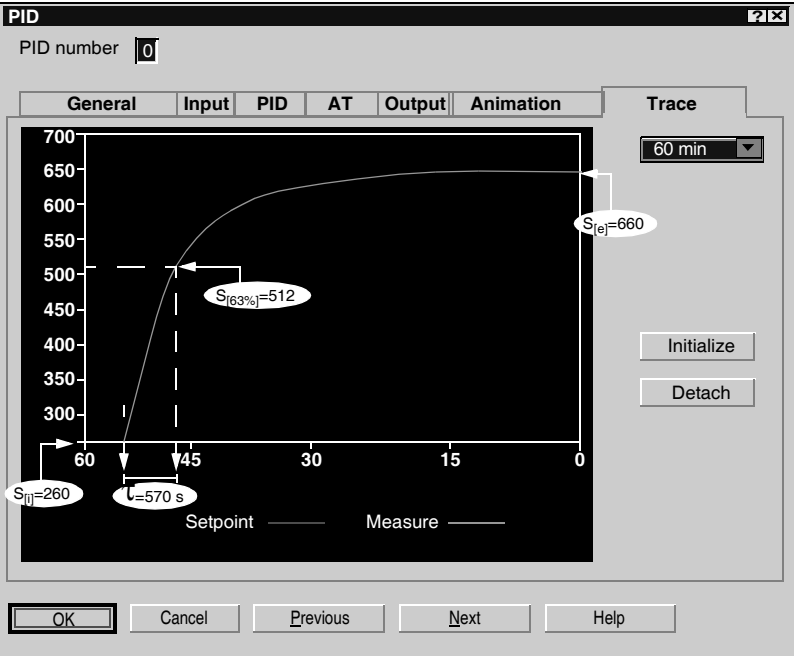
This example shows you how to measure the time constant (τ) of a simple thermal process by using the process response curve method described in the previous subsection.

The experimental setup for the time constant measurement is as follows:

- The control process consists in a forced air oven equipped with a heating lamp.
- Temperature measurements are gathered by the Twido PLC via a Pt100 probe, and temperature data are recorded in °C.
- The Twido PLC drives a heating lamp via the PWM discrete output of the PID.

The experiment is carried out as follows:

Step	Action
1	The PID Output tab is selected from the PID configuration screen.
2	Manual mode is selected from the Output tab.
3	The manual mode Output is set to 10000.
4	The PID run is launched from the PID Trace tab .
5	The PID run is stopped when the oven's temperature has reached a steady state.

Step	Action
6	The following information is obtained directly from the graphical analysis of the response curve, as shown in the figure below:  where • $S_{[i]}$ = initial value of process variable = 260 • $S_{[e]}$ = ending value of process variable = 660 • $S_{[63\%]}$ = process variable at 63% rise = $S_{[i]} + (S_{[i]} - S_{[e]}) \times 63\%$ $= 260 + (660 - 260) \times 63\%$ $= 512$ • τ = time constant = time elapsed from the start of the rise till $S_{[63\%]}$ is reached = 9 min 30 s = 570 s
7	The sampling period (T_s) is determined using the following relationship: $T_s = \tau / 75$ $= 570 / 75 = 7.6 \text{ s (7600 ms)}$
8	In the Program > Scan mode edit dialog box, the Scan Period must be set so that the sampling period (T_s) is an exact multiple of the scan period, as in the following example: Scan Period = $T_s / 76 = 7600 / 76 = 100 \text{ ms}$ (which satisfies the condition: $2 \text{ ms} \leq \text{Scan Period} \leq 150 \text{ ms}$.)

Trial-and-Error Method

The trial-and-error method consists in providing successive guesses of the sampling period to the auto-tuning function until the auto-tuning algorithm converges successfully towards Kp, Ti and Td that are deemed satisfactory by the user.

Note: Unlike the process response curve method, the trial-and-error method is not based on any approximation law of the process response. However, it has the advantage of converging towards a value of the sampling period that is in the same order of magnitude as the actual value.

To perform a trial-and-error estimation of the auto-tuning parameters, follow these steps:

Step	Action
1	Select the AT tab from the PID configuration window.
2	Set the Output limitation of AT to 10000 .
3	Select the PID tab from the PID configuration window.
4	Provide the first or n^{th} guess in the Sampling Period field. Note: If you do not have any first indication of the possible range for the sampling period, set this value to the minimum possible: 1 (1 unit of 10 ms).
5	Select PLC > Transfer PC => PLC... from menu bar to download the application program to the Twido PLC.
6	Launch Auto-Tuning .
7	Select the Animation tab from the PID configuration screen.
8	Wait till the auto-tuning process ends.
9	Two cases may occur: • Auto-tuning completes successfully: You may continue to Step 9. • Auto-tuning fails: This means the current guess for the sampling period (Ts) is not correct. Try a new Ts guess and repeat steps 3 through 8, as many times as required until the auto-tuning process eventually converges. Follow these guidelines to provide a new Ts guess: • AT ends with the error message "The computed time constant is negative!": This means the sampling period Ts is too large. You should decrease the value of Ts to provide as new guess. • AT ends with the error message "Sampling error!": This means the sampling period Ts is too small. You should increase the value of Ts to provide as new guess.
10	You may now view the PID control parameters (Kp, Ti and Td) in Animation tab, and adjust them in the PID tab of the PID configuration screen, as needed. Note: If the PID regulation provided by this set of control parameters does not provide results that are totally satisfactory, you may still refine the trial-and-error evaluation of the sampling period until you obtain the right set of Kp, Ti and Td control parameters.

Adjusting PID Parameters

To refine the process regulation provided by the PID parameters (K_p , T_i , T_d) obtained from auto-tuning, you also have the ability to adjust those parameter values manually, directly from the PID tab of the PID configuration screen or via the corresponding memory words (%MW).

Limitations on Using the Auto-tuning and the PID Control

The **auto-tuning** is best suited for processes whose time constant (τ) and delay-time (θ) meet the following requirement: $(\tau + \theta) < 2700$ s (i.e.: 45 min)
The **PID control** is best suited for the regulation of processes that satisfy the following condition: $2 < (\tau/\theta) < 20$, where (τ) is the time constant of the process and (θ) is the delay-time.

Note: Depending on the ratio (τ/θ):

- $(\tau/\theta) < 2$: The PID regulation has reached its limitations; more advanced regulation techniques are needed in this case.
 - $(\tau/\theta) > 20$: In this case, a simple on/off (or two-step) controller can be used in place of the PID controller.
-

Troubleshooting Errors of the Auto-tuning Function

The following table records the auto-tuning error messages and describes possible causes as well as troubleshooting actions:

Error Message	Possible Cause	Explanation / Possible Solution
Autotuning error: the process variable (PV) limit has been reached	The process variable is reaching the maximum value allowed.	This is a system safety. As the AT is an open-loop process, the Process Variable (PV) Limit works as an upper limit.
Autotuning error : due to either oversampling or output setpoint too low	Any of two possible causes: ● Sampling period is too small. ● AT Output is set too low.	Increase either the sampling period or the AT Output Setpoint value.
Autotuning error: the time constant is negative	The sampling period may be too large.	For more details, please check out <i>PID Tuning With Auto-Tuning (AT)</i> , p. 549.
Autotuning error: error calculating Kp	The AT algorithm has failed (no convergence).	Check the PID and AT parameters and make adjustments that can improve convergence. Check also that there is no disturbance that could affect the process variable.
Autotuning error: time constant over delay ratio > 20	$\tau/\theta > 20$	PID regulation is no longer guaranteed. For more details, please check out <i>PID Tuning With Auto-Tuning (AT)</i> , p. 549.
Autotuning error: time constant over delay ratio < 2	$\tau/\theta < 2$	PID regulation is no longer guaranteed. For more details, please check out <i>PID Tuning With Auto-Tuning (AT)</i> , p. 549.
Autotuning error: the limit for Kp has been exceeded	Computed value of static gain (Kp) is greater than 10000.	Measurement sensitivity of some application variables may be too low. The application's measurement range must be rescaled within the [0-10000] interval.
Autotuning error: the limit for Ti has been exceeded	Computed value of integral time constant (Ti) is greater than 20000.	Computational limit is reached.
Autotuning error: the limit for Td has been exceeded	Computed value of derivative time constant (Td) is greater than 10000.	Computational limit is reached.

PID parameter adjustment method

Introduction

Numerous methods to adjust the PID parameters exist, we suggest Ziegler and Nichols which have two variants:

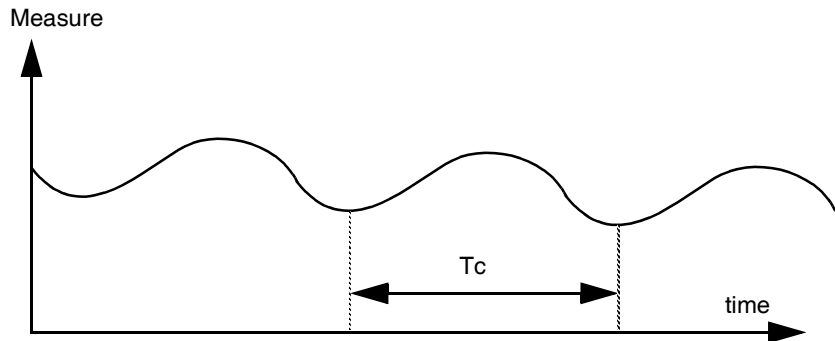
- closed loop adjustment,
- open loop adjustment.

Before implementing one of these methods, you must set the PID action direction:

- if an increase in the OUT output causes an increase in the PV measurement, make the PID inverted ($K_P > 0$),
- on the other hand, if this causes a PV reduction, make the PID direct ($K_P < 0$).

Closed loop adjustment

This principal consists of using a proportional command ($T_i = 0$, $T_d = 0$) to start the process by increasing production until it starts to oscillate again after having applied a level to the PID corrector setpoint. All that is required is to raise the critical production level (K_{pc}) which has caused the non damped oscillation and the oscillation period (T_c) to reduce the values giving an optimal regulation of the regulator.



According to the kind of (PID or PI) regulator, the adjustment of the coefficients is executed with the following values:

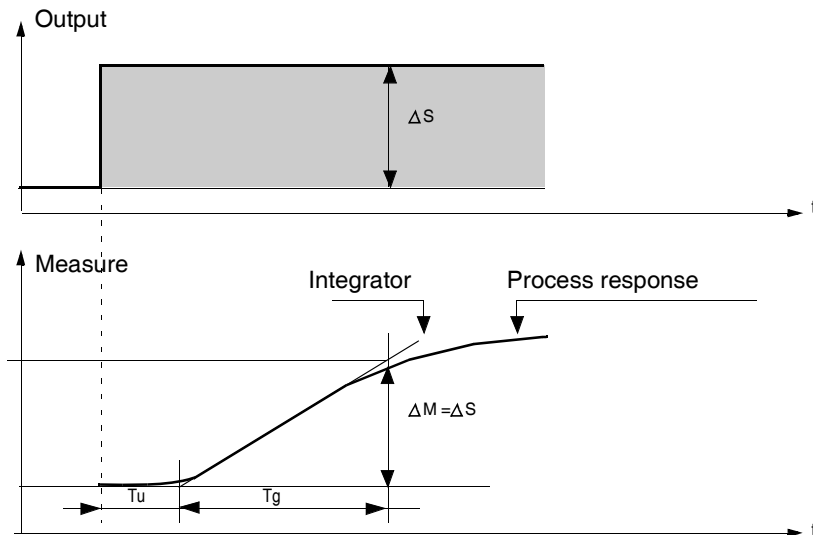
-	K_p	T_i	T_d
PID	$K_{pc}/1,7$	$T_c/2$	$T_c/8$
PI	$K_{pc}/2,22$	$0,83 \times T_c$	-

where K_p = proportional production, T_i = integration time and T_d = diversion time.

Note: This adjustment method provides a very dynamic command which can express itself through unwanted overshoots during the change of setpoint pulses. In this case, lower the production value until you get the required behavior.

Open loop adjustment

As the regulator is in manual mode, you apply a level to the output and make the procedure response start the same as an integrator with pure delay time.



The intersection point on the right hand side which is representative of the integrator with the time axes, determines the time T_u . Next, T_g time is defined as the time necessary for the controlled variable (measurement) to have the same variation size (% of the scale) as the regulator output. According to the kind of (PID or PI) regulator, the adjustment of the coefficients is executed with the following values:

-	K_p	T_i	T_d
PID	$-1,2 T_g/T_u$	$2 \times T_u$	$0,5 \times T_u$
PI	$-0,9 T_g/T_u$	$3,3 \times T_u$	-

where K_p = proportional production, T_i = integration time and T_d = diversion time.

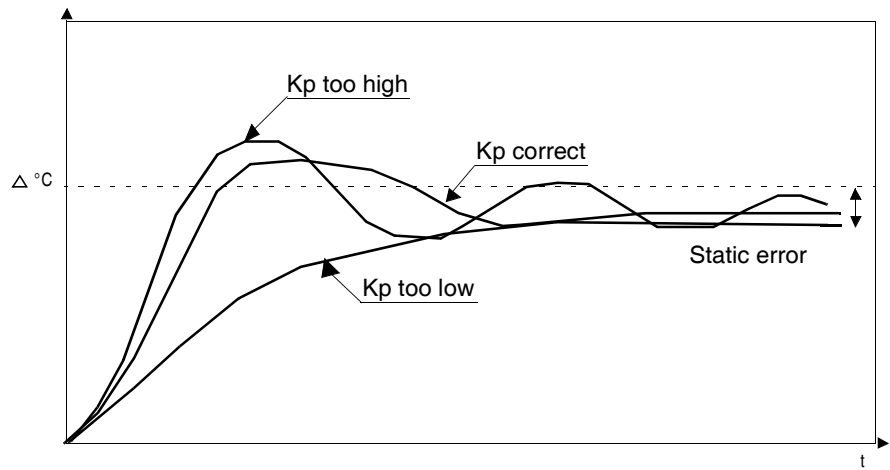
Note: Attention to the units. If the adjustment is carried out in PL7, multiply the value obtained for K_p by 100.

This adjustment method also provides a very dynamic command, which can express itself through unwanted overshoots during the change of setpoints' pulses. In this case, lower the production value until you get the required behavior. The method is interesting because it does not require any assumptions about the nature and the order of the procedure. You can apply it just as well to the stable procedures as to real integrating procedures. It is really interesting in the case of slow procedures (glass industry,...) because the user only requires the beginning of the response to regulate the coefficients K_p , T_i and T_d .

Role and influence of PID parameters

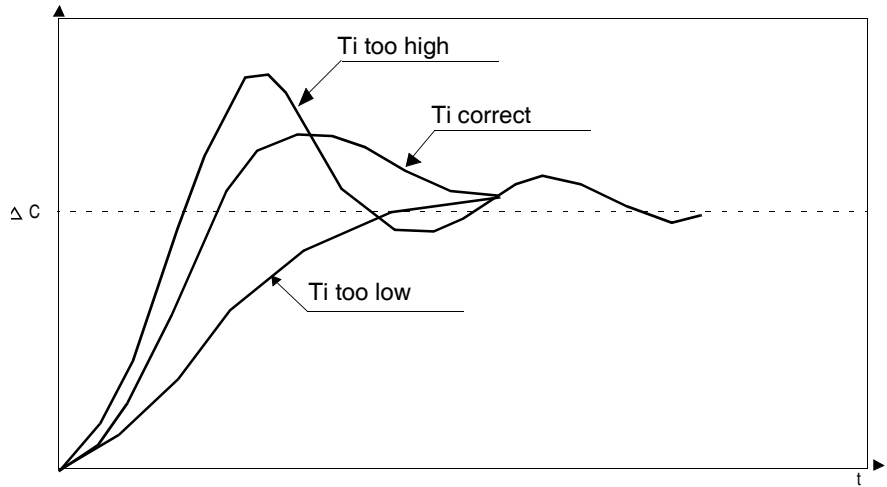
Influence of proportional action

Proportional action is used to influence the process response speed. The higher the gain, the faster the response, and the lower the static error (in direct proportion), though the more stability deteriorates. A suitable compromise between speed and stability must be found. The influence of integral action on process response to a scale division is as follows:



**Influence of
integral action**

Integral action is used to cancel out static error (deviation between the process value and the setpoint). The higher the level of integral action (low T_i), the faster the response and the more stability deteriorates. It is also necessary to find a suitable compromise between speed and stability. The influence of integral action on process response to a scale division is as follows:

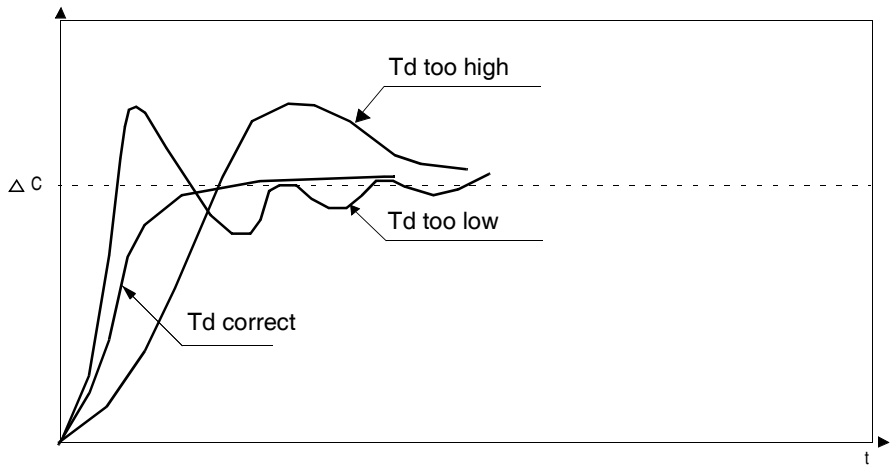


Note: A low T_i means a high level of integral action.

where K_p = proportional gain, T_i = integration time and T_d = derivative time.

Influence of derivative action

Derivative action is anticipatory. In practice, it adds a term which takes account of the speed of variation in the deviation, which makes it possible to anticipate changes by accelerating process response times when the deviation increases and by slowing them down when the deviation decreases. The higher the level of derivative action (high T_d), the faster the response. A suitable compromise between speed and stability must be found. The influence of derivative action on process response to a scale division is as follows:



Limits of the PID control loop

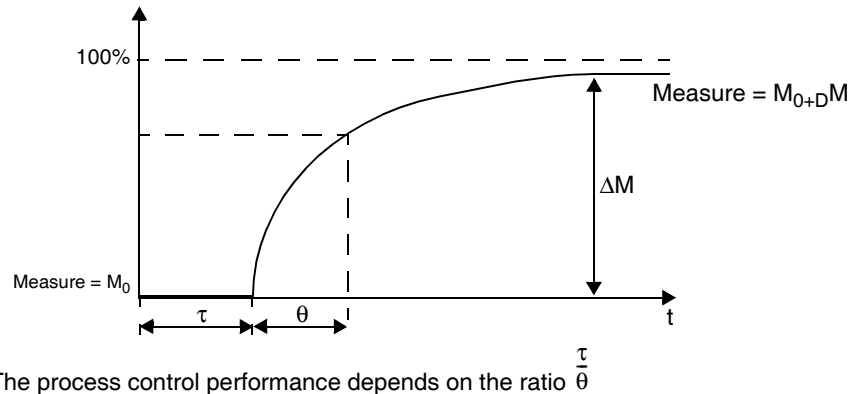
If the process is assimilated to a pure delay first order with a transfer function:

$$(H(p)) = K \frac{e^{(-\tau)p}}{(1 + \theta p)}$$

where:

τ = model delay,

θ = model time constant,



The process control performance depends on the ratio $\frac{\tau}{\theta}$

The suitable PID process control is attained in the following domain: $2 - \frac{\tau}{\theta} - 20$

For $\frac{\tau}{\theta} < 2$, in other words for fast control loops (low θ) or for processes with a large delay (high t) the PID process control is no longer suitable. In such cases more complex algorithms should be used.

For $\frac{\tau}{\theta} > 20$, a process control using a threshold plus hysteresis is sufficient.

Appendix 1: PID Theory Fundamentals

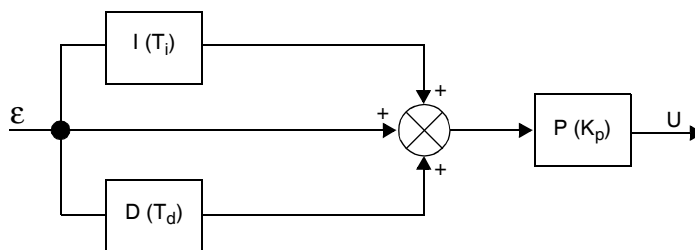
Introduction

The PID control function onboard all Twido controllers provides an efficient control to simple industrial processes that consist of one system stimulus (referred to as Setpoint in this document) and one measurable property of the system (referred to as Measure or Process Variable).

The PID Controller Model

The Twido PID controller implements a mixed (serial - parallel) PID correction (see PID Model Diagram below) via an analog measurement and setpoint in the [0-10000] format and provides an analog command to the controlled process in the same format.

The mixed form of the PID controller model is described in the following diagram:



where

where:

- I = the **integral** action (acting independently and parallel to the derivative action),
- D = the **derivative** action (acting independently and parallel to the integral action),
- P = the **proportional** action (acting serially on the combined output of the integral and derivative actions,
- U = the PID controller output (later fed as input into the controlled process.)

The PID Control Law

The PID controller is comprised of the mixed combination (serial - parallel) of the controller gain (K_p), and the integral (T_i) and derivative (T_d) time constants. Thus, the PID control law that is used by the Twido controller is of the following form (Eq. 1):

$$u(i) = K_p \cdot \left\{ \varepsilon(i) + \frac{T_s}{T_i} \sum_{j=1}^i \varepsilon(j) + \frac{T_d}{T_s} [\varepsilon(i) - \varepsilon(i-1)] \right\}$$

where

- K_p = the controller proportional gain,
- T_i = the integral time constant,
- T_d = the derivative time constant,
- T_s = the sampling period,
- $\varepsilon(i)$ = the deviation ($\varepsilon(i)$ = setpoint - process variable.)

Note: Two different computational algorithms are used, depending on the value of the integral time constant (T_i):

- $T_i \neq 0$: In this case, an incremental algorithm is used.
- $T_i = 0$: This is the case for non-integrating processes. In this case, a positional algorithm is used, along with a +5000 offset that is applied to the PID output variable.

For a detailed description of K_p , T_i and T_d please refer to *PID tab of PID function*, p. 530.

As can be inferred from (equ. 1) and (equ. 1'), the key parameter for the PID regulation is the **sampling period (T_s)**. The sampling period depends closely on the **time constant (τ)**, a parameter intrinsic to the process the PID aims to control. (See Appendix 2: First-Order With Time Delay Model, p. 565.)

Appendix 2: First-Order With Time Delay Model

Introduction

This section presents the first-order with time delay model used to describe a variety of simple but nonetheless important industrial processes, including thermal processes.

First-Order With Time Delay Model

It is widely assumed that simple (one-stimulus) thermal processes can be adequately approximated by a first-order with time delay model. The transfer function of such first-order, open-loop process has the following form in the Laplace domain (*equ.2*):

$$\frac{S}{U} = \frac{k}{1 + \tau p} \cdot e^{-\theta p}$$

where

- k = the static gain,
 - τ = the time constant,
 - θ = the delay-time,
 - U = the process input (this is the output of the PID controller),
 - S = the process output.
-

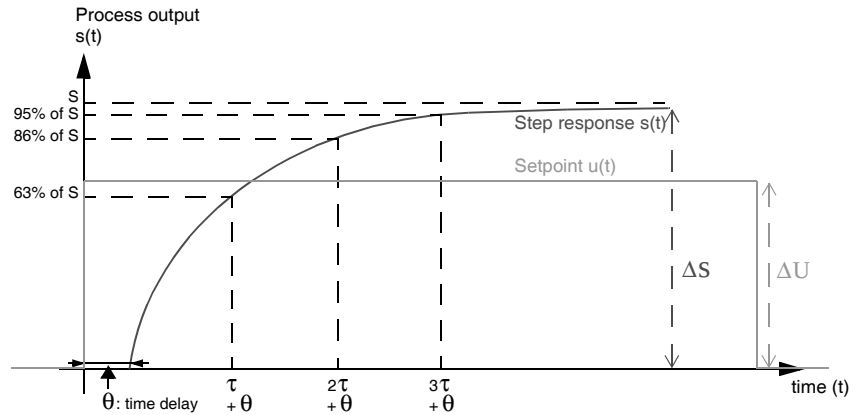
The Process

Time Constant τ

The key parameter of the process response law (*equ.2*) is the **time constant** τ . It is a parameter intrinsic to the process to control.

The time constant (τ) of a first-order system is defined as the time (in sec) it takes the system output variable to reach 63% of the final output from the time the system started reacting to the step stimulus $u(t)$.

The following figure shows a typical first-order process response to a step stimulus:



where

- k = the static gain computed as the ratio $\Delta S / \Delta U$,
- τ = the time at 63% rise = the time constant,
- 2τ = the time at 86% rise,
- 3τ = the time at 95% rise.

Note: When auto-tuning is implemented, the sampling period (T_s) must be chosen in the following range: $[\tau/125 < T_s < \tau/25]$. Ideally, you should use $[T_s = \tau/75]$. (See *PID Tuning With Auto-Tuning (AT)*, p. 549.)

17.5 Floating point instructions

At a Glance

Aim of this Section

This section describes advanced floating point (See *Floating point and double word objects*, p. 32) instructions in TwidoSoft language.
The Comparison and Assignment instructions are described in the *Numerical Processing*, p. 409

What's in this Section?

This section contains the following topics:

Topic	Page
Arithmetic instructions on floating point	568
Trigonometric Instructions	572
Conversion instructions	574
Integer Conversion Instructions <-> Floating	575

Arithmetic instructions on floating point

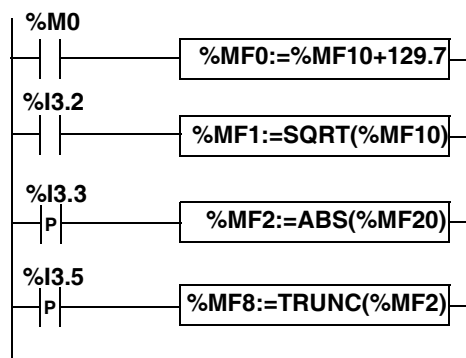
General

These instructions are used to perform an arithmetic operation between two operands or on one operand.

+	addition of two operands	SQRT	square root of an operand
-	subtraction of two operands	ABS	absolute value of an operand
*	multiplication of two operands	TRUNC	whole part of a floating point value
/	division of two operands	EXP	natural exponential
LOG	base 10 logarithm	EXPT	power of an integer by a real
LN	natural logarithm		

Structure

Ladder Language



Instruction List Language

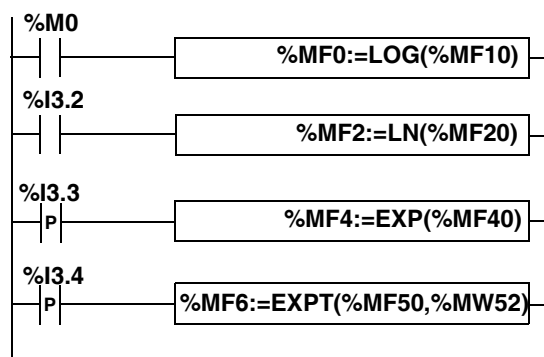
```
LD %M0
[%MF0:=%MF10+129.7]
```

```
LD %I3.2
[%MF1:=SQRT(%MF10)]
```

```
LDR %I3.3
[%MF2:=ABS(%MF20)]
```

```
LDR %I3.5
[%MF8:=TRUNC(%MF2)]
```

Ladder Language



Instruction List Language

LD %M0

[%MF0:=LOG(%MF10)]

LD %I3.2

[%MF2:=LN(%MF20)]

LDR %I3.3

[%MF4:=EXP(%MF40)]

LDR %I3.4

[%MF6:=EXPT(%MF50,%MW52)]

Syntax

Operators and syntax of arithmetic instructions on floating point

Operators	Syntax
+, -, *, /	Op1:=Op2 Operator Op3
SQRT, ABS, TRUNC, LOG, EXP, LN	Op1:=Operator(Op2)
EXPT	Op1:=Operator (Op2,Op3)

Note: When you perform an addition or subtraction between 2 floating point numbers, the two operands must comply with the condition: $Op1 > Op2 \times 2^{-24}$, where $Op1 > Op2$. If this condition is not respected, the result is equal to operand 1 (Op1). This phenomenon is of little consequence in the case of an isolated operation, as the resulting error is very low (2^{-24}), but it can have unforeseen consequences where the calculation is repeated. E.g. in the case where the instruction **%MF2:= %MF2 + %MF0** is repeated indefinitely. If the initial conditions are **%MF0 = 1.0** and **%MF2 = 0**, the value **%MF2** becomes blocked at 16777216. We therefore recommend you take great care when programming repeated calculations. If, however, you wish to program this type of calculation, it is up to the client application to manage truncation errors.

Operands of arithmetic instructions on floating point:

Operators	Operand 1 (Op1)	Operand 2 (Op2)	Operand 3 (Op3)
+, -, *, /	%MFi	%MFi, %KFi, immediate value	%MFi, %KFi, immediate value
SQRT, ABS, LOG, EXP, LN	%MFi	%MFi, %KFi	[-]
TRUNC	%MFi	%MFi, %KFi	[-]
EXPT	%MFi	%MFi, %KFi	%MWi, %KW, immediate value

Rules of use

- Operations on floating point and integer values can not be directly mixed. Conversion operations (See *Integer Conversion Instructions <-> Floating*, p. 575) convert into one or other of these formats.)
- The system bit %S18 is managed in the same way as integer operations (See *Arithmetic Instructions on Integers*, p. 418), the word %SW17 (See *System Words (%SW)*, p. 604) indicates the cause of the fault.
- When the operand of the function is an invalid number (e.g.: logarithm of a negative number), it produces an indeterminate or infinite result and changes bit %S18 to 1, the word %SW17 indicates the cause of the error.

Trigonometric Instructions

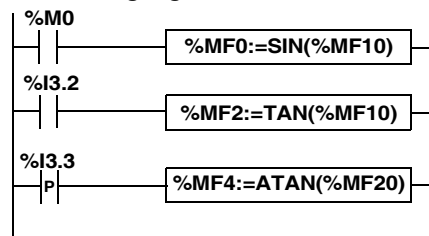
General

These instructions enable the user to perform trigonometric operations.

SIN	sine of an angle expressed in radian,	ASIN	arc sine (result within $-\frac{\pi}{2}$ and $\frac{\pi}{2}$)
COS	cosine of an angle expressed in radian,	ACOS	arc cosine (result within 0 and π)
TAN	tangent of an angle expressed in radian,	ATAN	arc tangent (result within $-\frac{\pi}{2}$ and $\frac{\pi}{2}$)

Structure

Ladder language



Instruction List Language

```
LD %M0
[%MF0 := SIN(%MF10)]
```

```
LD %I3.2
[%MF2 := TAN(%MF10)]
```

```
LDR %I3.3
[%MF4 := ATAN(%MF20)]
```

Structured text language

```
IF %M0 THEN
  %MF0 := SIN(%MF10);
END_IF;
IF %I3.2 THEN
  %MF2 := TAN(%MF10);
END_IF;
IF %I3.3 THEN
  %MF4 := ATAN(%MF20);
END_IF;
```


Syntax

Operators, operands and syntax of instructions for trigonometric operations

Operators	Syntax	Operand 1 (Op1)	Operand 2 (Op2)
SIN, COS, TAN, ASIN, ACOS, ATAN	Op1:=Operator(Op2)	%MFi	%MFi, %KFi

Rules of use

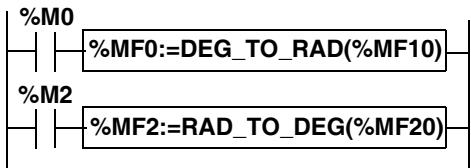
- when the operand of the function is an invalid number (e.g.: arc cosine of a number greater than 1), it produces an indeterminate or infinite result and changes bit %S18 to 1, the word %SW17 (See *System Words (%SW)*, p. 604) indicates the cause of the error.
- the functions SIN/COS/TAN allow as a parameter an angle between -4096π and 4096π but their precision decreases progressively for the angles outside the period -2π and $+2\pi$ because of the imprecision brought by the modulo 2π carried out on the parameter before any operation.

Conversion instructions

General These instructions are used to carry out conversion operations.

DEG_TO_RAD	conversion of degrees into radian, the result is the value of the angle between 0 and 2π
RAD_TO_DEG	conversion of an angle expressed in radian, the result is the value of the angle between 0 and 360 degrees

Structure **Ladder language**



Instruction List Language

```
LD %M0
[%MF0 := DEG_TO_RAD (%MF10)]
```

```
LD %M2
[%MF2 := RAD_TO_DEG (%MF20)]
```

Structured Text language

```
IF %M0 THEN
  %MF0 := DEG_TO_RAD (%MF10) ;
END_IF ;
IF %M2 THEN
  %MF2 := RAD_TO_DEG (%MF20) ;
END_IF ;
```

Syntax Operators, operands and syntax of conversion instructions

Operators	Syntax	Operand 1 (Op1)	Operand 2 (Op2)
DEG_TO_RAD RAD_TO_DEG	Op1:=Operator(Op2)	%MFi	%MFi, %KFi

Rules of use The angle to be converted must be between -737280.0 and +737280.0 (for DEG_TO_RAD conversions) or between -4096π and 4096π (for RAD_TO_DEG conversions). For values outside these ranges, the displayed result will be + 1.#NAN, the %S18 and %SW17:X0 bits being set at 1.

Integer Conversion Instructions <-> Floating

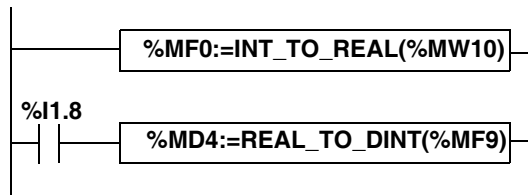
General

Four conversion instructions are offered.
Integer conversion instructions list<-> floating:

INT_TO_REAL	conversion of an integer word --> floating
DINT_TO_REAL	conversion of a double word (integer) --> floating
REAL_TO_INT	conversion of a floating --> integer word (the result is the nearest algebraic value)
REAL_TO_DINT	conversion of a floating --> double integer word (the result is the nearest algebraic value)

Structure

Ladder language



Instruction List Language

```
LD TRUE
[%MF0:=INT_TO_REAL(%MW10)]
```

```
LD I1.8
[%MD4:=REAL_TO_DINT(%MF9)]
```

Structured Text language

```
%MF0:=INT_TO_REAL(%MW10);
IF %I1.8 THEN
  %MD4:=REAL_TO_DINT(%MF9);
END_IF;
```

Syntax

Operators and syntax (conversion of an integer word --> floating):

Operators	Syntax
INT_TO_REAL	Op1=INT_TO_REAL(Op2)

Operands (conversion of an integer word --> floating):

Operand 1 (Op1)	Operand 2 (Op2)
%MFi	%MWi,%KWi

Example: integer word conversion --> floating: 147 --> 1.47e+02

Operators and syntax (double conversion of integer word --> floating):

Operators	Syntax
DINT_TO_REAL	Op1=DINT_TO_REAL(Op2)

Operands (double conversion of integer word --> floating):

Operand 1 (Op1)	Operand 2 (Op2)
%MFi	%MDi,%KDi

Example: integer double word conversion --> floating: 68905000 --> 6.8905e+07

Operators and syntax (floating conversion --> integer word or integer double word):

Operators	Syntax
REAL_TO_INT	Op1=Operator(Op2)
REAL_TO_DINT	

Operators (floating conversion --> integer word or integer double word):

Type	Operand 1 (Op1)	Operand 2 (Op2)
Words	%MWi	%MFi, %KFi
Double words	%MDi	%MFi, %KFi

Example:

floating conversion --> integer word: 5978.6 --> 5979

floating conversion --> integer double word: -1235978.6 --> -1235979

Note: If during a real to integer (or real to integer double word) conversion the floating value is outside the limits of the word (or double word), bit %S18 is set to 1.

Precision of Rounding

Standard IEEE 754 defines 4 rounding modes for floating operations.

The mode employed by the instructions above is the "rounded to the nearest" mode: "if the nearest representable values are at an equal distance from the theoretical result, the value given will be the value whose low significance bit is equal to 0". In certain cases, the result of the rounding can thus take a default value or an excess value.

For example:

Rounding of the value 10.5 -> 10

Rounding of the value 11.5 -> 12

17.6 Instructions on Object Tables

At a Glance

Aim of this Section

This section describes instructions specific to tables:

- of double words,
- of floating point objects.

Assignment instructions for tables are described in the chapter on "basic instructions" (See *Assignment of Word, Double Word and Floating Point Tables*, p. 414).

What's in this Section?

This section contains the following topics:

Topic	Page
Table summing functions	579
Table comparison functions	580
Table search functions	582
Table search functions for maxi and mini values	584
Number of occurrences of a value in a table	585
Table rotate shift function	586
Table sort function	588
Floating point table interpolation function	589
Mean function of the values of a floating point table	594

Table summing functions

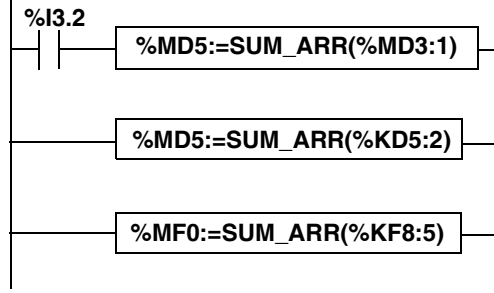
General

The SUM_ARR function adds together all the elements of an object table:

- if the table is made up of double words, the result is given in the form of a double word
- if the table is made up of floating words, the result is given in the form of a floating word

Structure

Ladder language



Instruction List Language

```
LD %I3.2
[ %MD5 := SUM_ARR ( %MD3 : 1 ) ]
%MD5 := SUM_ARR ( %KD5 : 2 )
%MF0 := SUM_ARR ( %KF8 : 5 )
```

Syntax

Syntax of table summing instruction:

```
Res:=SUM_ARR(Tab)
```

Parameters of table summing instruction

Type	Result (res)	Table (Tab)
Double word tables	%MDi	%MDi:L, %KDi:L
Floating word tables	%MFi	%MFi:L, %KFi:L

Note: When the result is not within the valid double word format range according to the table operand, the system bit %S18 is set to 1.

Example

```
%MD5 := SUM ( %MD30 : 4 )
where %MD30=10, %MD31=20, %MD32=30, %MD33=40
%MD5=10+20+30+40=100
```

Table comparison functions

General

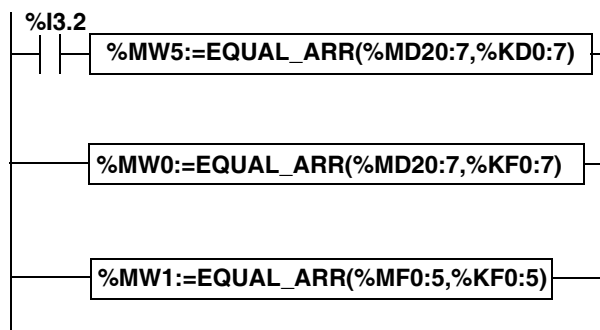
The EQUAL_ARR function carries out a comparison of two tables, element by element.

If a difference is shown, the rank of the first dissimilar elements is returned in the form of a word, otherwise the returned value is equal to -1.

The comparison is carried out on the whole table.

Structure

Ladder language



Instruction List Language

LD %I3.2

[%MW5:=EQUAL_ARR(%MD20:7,KD0:7)]

Structured Text language

%MW0:=EQUAL_ARR(%MD20:7,%KF0:7)

%MW1:=EQUAL_ARR(%MF0:5,%KF0:5)

Syntax

Syntax of table comparison instruction:

Res:=EQUAL_ARR(Tab1,Tab2)

Parameters of table comparison instructions:

Type	Result (Res)	Tables (Tab1 and Tab2)
Double word tables	%MWi	%MDi:L,%KDi:L
Floating word tables	%MWi	%MFi:L,%KFi:L

Note:

- it is mandatory that the tables are of the same length and same type.

Example

%MW5 :=EQUAL_ARR (%MD30:4,%KD0:4)

Comparison of 2 tables:

Rank	Word Table	Constant word tables	Difference
0	%MD30=10	%KD0=10	=
1	%MD31=20	%KD1=20	=
2	%MD32=30	%KD2=60	Different
3	%MD33=40	%KD3=40	=

The value of the word %MW5 is 2 (different first rank)

Table search functions

General

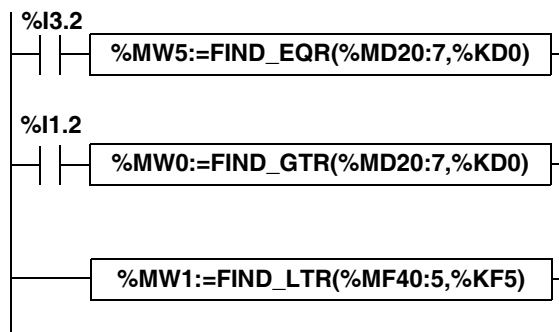
There are 3 search functions:

- **FIND_EQR**: searches for the position in a double or floating word table of the first element which is equal to a given value
- **FIND_GTR**: searches for the position in a double or floating word table of the first element which is greater than a given value
- **FIND_LTR**: searches for the position in a double or floating word table of the first element which is less than a given value

The result of these instructions is equal to the rank of the first element which is found or at -1 if the search is unsuccessful.

Structure

Ladder language



Instruction List Language

```

LD  %I3.2
[%MW5:=FIND_EQR(%MD20:7,%KD0)]
LD  %I1.2
[%MW0:=FIND_GTR(%MD20:7,%KD0)]
%MW1:=FIND_LTR(%MF40:5,%KF5)
  
```

Syntax

Syntax of table search instructions:

Function	Syntax
FIND_EQR	Res:=Function(Tab,Val)
FIND_GTR	
FIND_LTR	

Parameters of floating word and double word table search instructions:

Type	Result (Res)	Table (Tab)	Value (val)
Floating word tables	%MWi	%MFi:L,%KFi:L	%MFi,%KFi
Double word tables	%MWi	%MDi:L,%KDi:L	%MDi,%KDi

Example

%MW5 := FIND_EQR (%MD30:4, %KD0)

Search for the position of the first double word =%KD0=30 in the table:

Rank	Word Table	Result
0	%MD30=10	-
1	%MD31=20	-
2	%MD32=30	Value (val), rank
3	%MD33=40	-

Table search functions for maxi and mini values

General

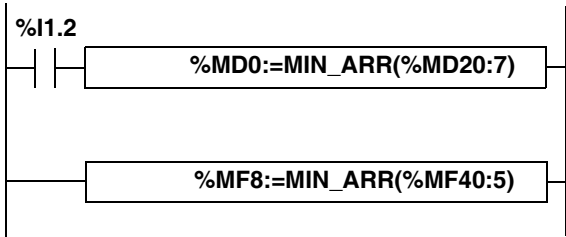
There are 2 search functions:

- **MAX_ARR**: search for the maximum value in a double word and floating word table
- **MIN_ARR**: search for the minimum value in a double word and floating word table

The result of these instructions is equal to the maximum value (or minimum) found in the table.

Structure

Ladder language



Instruction List Language

```
LD %I1.2
[ %MD0 := MIN_ARR ( %MD20 : 7 ) ]
%MF8 := MIN_ARR ( %MF40 : 5 )
```

Syntax

Syntax of table search instructions for max and min values:

Function	Syntax
MAX_ARR	Res:=Function(Tab)
MIN_ARR	

Parameters of table search instructions for max and min values:

Type	Result (Res)	Table (Tab)
Double word tables	%MDi	%MDi:L,%KDi:L
Floating word tables	%MFi	%MFi:L,%KFi:L

Number of occurrences of a value in a table

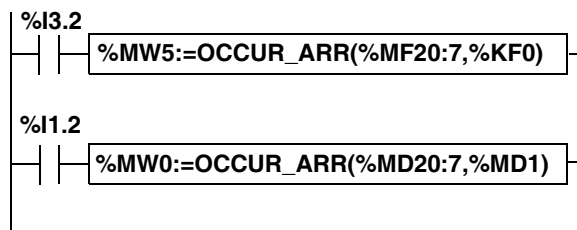
General

This search function:

- **OCCUR_ARR**: searches in a double word or floating word table for a number of elements equal to a given value

Structure

Ladder language



Instruction List Language

```

LD  %I3.2
[ %MW5 := OCCUR_ARR ( %MF20 : 7 , %KF0 ) ]
LD  %I1.2
[ %MW0 := OCCUR_ARR ( %MD20 : 7 , %MD1 ) ]
  
```

Syntax

Syntax of table search instructions for max and min values:

Function	Syntax
OCCUR_ARR	Res:=Function(Tab,Val)

Parameters of table search instructions for max and min values:

Type	Result (Res)	Table (Tab)	Value (Val)
Double word tables	%MWi	%MDi:L,%KDi:L	%MDi,%KDi
Floating word tables	%MFi	%MFi:L,%KFi:L	%MFi,%KFi

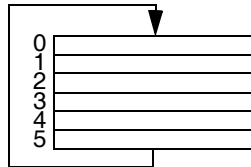
Table rotate shift function

General

There are 2 shift functions:

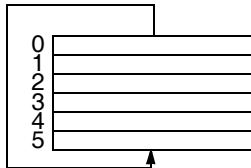
- **ROL_ARR**: performs a rotate shift of n positions from top to bottom of the elements in a floating word table

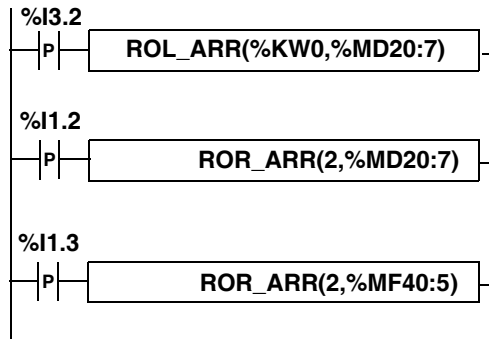
Illustration of the ROL_ARR functions



- **ROR_ARR**: performs a rotate shift of n positions from bottom to top of the elements in a floating word table

Illustration of the ROR_ARR functions



Structure**Ladder language****Instruction List Language**

```

LDR %I3.2
[ROL_ARR(%KW0,%MD20:7)]
LDR %I1.2
[ROR_ARR(2,%MD20:7)]
LDR %I1.3
[ROR_ARR(2,%MF40:5)]
  
```

Syntax

Syntax of rotate shift instructions in floating word or double word tables **ROL_ARR** and **ROR_ARR**

Function	Syntax
ROL_ARR	Function(n,Tab)
ROR_ARR	

Parameters of rotate shift instructions for floating word tables: **ROL_ARR** and **ROR_ARR**:

Type	Number of positions (n)	Table (Tab)
Floating word tables	%MWi, immediate value	%MFi:L
Double word tables	%MWi, immediate value	%MDi:L

Note: if the value of n is negative or null, no shift is performed.

Table sort function

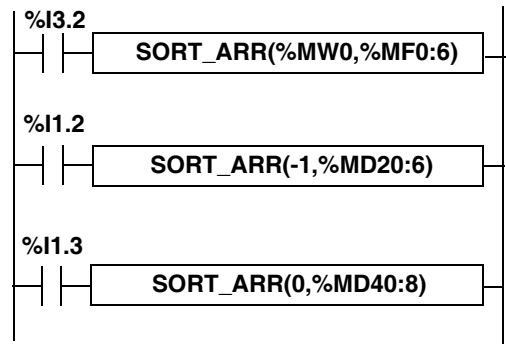
General

The sort function available is as follows:

- **SORT_ARR**: performs sorts in ascending or descending order of the elements of a double word or floating word table and stores the result in the same table.

Structure

Ladder language



Instruction List Language

```
Ld %I3.2
[ SORT_ARR ( %MW20 , %MF0 : 6 ) ]
Ld %I1.2
[ SORT_ARR ( -1 , %MD20 : 6 ) ]
Ld %I1.3
[ SORT_ARR ( 0 , %MF40 : 8 ) ]
```

Syntax

Syntax of table sort functions:

Function	Syntax
SORT_ARR	Function(direction,Tab)

- the "direction" parameter gives the order of the sort: direction > 0 the sort is done in ascending order; direction < 0, the sort is done in descending order, direction = 0 no sort is performed.
- the result (sorted table) is returned in the Tab parameter (table to sort).

Parameters of table sort functions:

Type	Sort direction	Table (Tab)
Double word tables	%MWi, immediate value	%MDi:L
Floating word tables	%MWi, immediate value	%MFi:L

Floating point table interpolation function

Overview

The **LKUP** function is used to interpolate a set of X versus Y floating point data for a given X value.

Interpolation Rule

The LKUP function makes use the linear interpolation rule, as defined in the following equation:

$$(equation\ 1:) \quad Y = Y_i + \left[\frac{(Y_{i+1} - Y_i)}{(X_{i+1} - X_i)} \cdot (X - X_i) \right]$$

for $X_i \leq X \leq X_{i+1}$, where $i = 1 \dots (m-1)$;

assuming X_i values are ranked in ascending order: $X_1 \leq X_2 \leq \dots X \dots \leq X_{m-1} \leq X_m$.

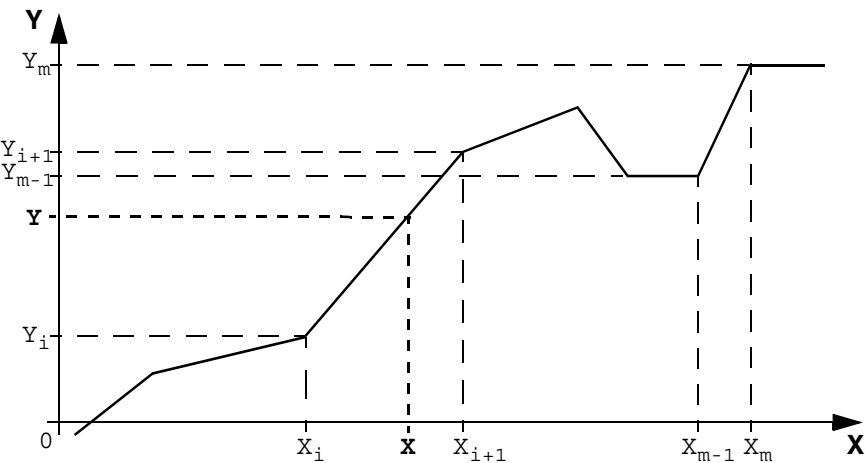
Note: If any of two consecutive X_i values are equal ($X_i = X_{i+1} = X$), equation (1) yields an invalid exception. In this case, to cope with this exception the following algorithm is used in place of equation (1):

$$(equation\ 2:) \quad Y = \left[\frac{(Y_{i+1} - Y_i)}{2} \right]$$

for $X_i = X_{i+1} = X$, where $i = 1 \dots (m-1)$.

**Graphical
Representation
of the Linear
Interpolation
Rule**

The following graph illustrates the linear interpolation rule described above:



**Syntax of the
LKUP Function**

The LKUP function uses three operands, two of which are function attributes, as described in the following table:

Syntax	Operand 1 (Op1) Output variable	Operand 2 (Op2) User-defined (X) value	Operands 3 (Op3) User-defined (X _i , Y _i) variable array
[Op1: = LKUP(Op2,Op3)]	%MWi	%MF0	Integer value, %MWi or %KW _i

Definition of Op1 **Op1** is the memory word that contains the output variable of the interpolation function.

Depending on the value of Op1, the user can know whether the interpolation was successful or failed, and what caused for the failure, as outlined in the following table:

Op1 (%Mwi)	Description
0	Successful interpolation
1	Interpolation error: Bad array, $X_m < X_{m-1}$
2	Interpolation error: Op2 out of range, $X < X_1$
4	Interpolation error: Op2 out of range, $X > X_m$
8	Invalid size of data array: • Op3 is set as odd number, or • $Op3 < 6$.

Note: Op1 **does not** contain the computed interpolation value (Y). For a given (X) value, the result of the interpolation (Y) is contained in %MF2 of the Op3 array (See *Definition of Op3* below).

Definition of Op2 **Op2** is the floating point variable (%MF0 of the Op3 floating point array) that contains the user-defined (X) value for which to compute the interpolated (Y) value:

- Valid range for Op2 is as follows: $X_1 \leq Op2 \leq X_m$.

Definition of Op3 Op3 sets the size ($\text{Op3} / 2$) of the floating-point array where the (X_i, Y_i) data pairs are stored.

X_i and Y_i data are stored in floating point objects with even indexes, starting at %MF4 (note that %MF0 and %MF2 floating point objects are reserved for the user set-point X and the interpolated value Y, respectively).

Given an array of (m) data pairs (X_i, Y_i) , the upper index (u) of the floating point array (%MFu) is set by using the following relationships:

- (equation 3:) $\text{Op3} = 2 \cdot m$;
- (equation 4:) $u = 2 \cdot (\text{Op3} - 1)$.

The floating point array Op3 (%MFi) has a structure similar to that of the following example (where $\text{Op3}=8$):

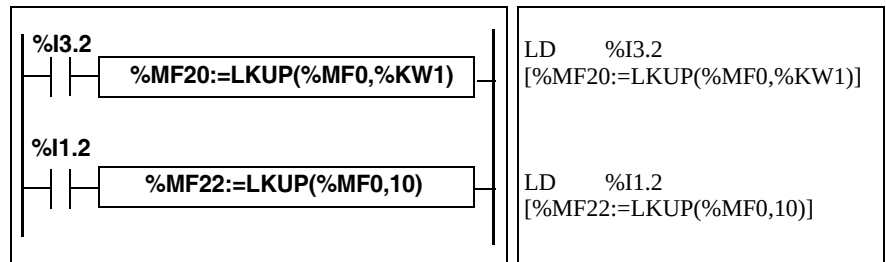
(X)		(X ₁)		(X ₂)		(X ₃)	
%MF0		%MF4		%MF8		%MF12	
	%MF2		%MF6		%MF10		%MF14
	(Y)		(Y ₁)		(Y ₂)		(Y ₃)
							(Op3=8)

Note: As a result of the above floating-point array's structure, Op3 must meet both of the following requirements, or otherwise this will trigger an error of the LKUP function:

- Op3 is an even number, and
- $\text{Op3} \geq 6$ (for there must be at least 2 data points to allow linear interpolation).

Structure

Interpolation operations are performed as follows:



Example

The following is an example use of a LKUP interpolation function:

```
[%MW20:=LKUP(%MF0,10)]
```

In this example:

- %MW20 is Op1 (the output variable).
- %MF0 is the user-defined (X) value which corresponding (Y) value must be computed by linear interpolation.
- %MF2 stores the computed value (Y) resulting from the linear interpolation.
- 10 is Op3 (as given by *equation 3* above). It sets the size of the floating point array. The highest ranking item %MFu, where u=18 is given by *equation 4*, above.

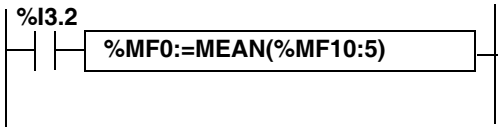
There are 4 pairs of data points stored in Op3 array [%MF4..%MF18]:

- %MF4 contains X_1 , %MF6 contains Y_1 .
 - %MF8 contains X_2 , %MF10 contains Y_2 .
 - %MF12 contains X_3 , %MF14 contains Y_3 .
 - %MF16 contains X_4 , %MF18 contains Y_4 .
-

Mean function of the values of a floating point table

General The **MEAN** function is used to calculate the mean average from a given number of values in a floating point table.

Structure **Ladder Language**



Instruction List Language

```
LD %I3.2
[ %MF0 := MEAN ( %MF10 : 5 ) ]
```

Syntax Syntax of the floating point table mean calculation function:

Function	Syntax
MEAN	Result=Function(Op1)

Parameters of the calculation function for a given number L of values from a floating point table:

Operand (Op1)	Result (Res)
%MFi:L, %KFi:L	%MFi

System Bits and System Words

18

At a Glance

Subject of this Chapter

This chapter provides an overview of the system bits and system words that can be used to create control programs for Twido controllers.

What's in this Chapter?

This chapter contains the following topics:

Topic	Page
System Bits (%S)	596
System Words (%SW)	604

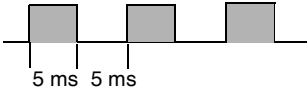
System Bits (%S)

Introduction

The following section provides detailed information about the function of system bits and how they are controlled.

Detailed Description

The following table provides an overview of the system bits and how they are controlled:

System Bit	Function	Description	Init state	Control
%S0	Cold Start	Normally set to 0, it is set to 1 by: • A power return with loss of data (battery fault), • The user program or Animation Table Editor, • Operations Display. This bit is set to 1 during the first complete scan. It is reset to 0 by the system before the next scan.	0	S or U->S
%S1	Warm Start	Normally set to 0, it is set to 1 by: • A power return with data backup, • The user program or Animation Table Editor, • Operations Display. It is reset to 0 by the system at the end of the complete scan.	0	S or U->S
%S4 %S5 %S6 %S7	Time base: 10 ms Time base: 100 ms Time base: 1 s Time base: 1 min	The rate of status changes is measured by an internal clock. They are not synchronized with the controller scan. Example: %S4 	-	S
%S8	Wiring test	Initially set to 1, this bit is used to test the wiring when the controller is in "non-configured" state. To modify the value of this bit, use the operations display keys to make the required output status changes: • Set to 1, output reset, • Set to 0, wiring test authorized.	1	U
%S9	Reset outputs	Normally set to 0. It can be set to 1 by the program or by the terminal (in the Animation Table Editor): • At state 1, outputs are forced to 0 when the controller is in RUN mode, • At state 0, outputs are updated normally.	0	U

System Bit	Function	Description	Init state	Control
%S10	I/O fault	Normally set to 1. This bit can be set to 0 by the system when an I/O fault is detected.	1	S
%S11	Watchdog overflow	Normally set to 0. This bit can be set to 1 by the system when the program execution time (scan time) exceeds the maximum scan time (software watchdog). Watchdog overflow causes the controller to change to HALT.	0	S
%S12	PLC in RUN mode	This bit reflects the running state of the controller. The systems sets the bit to 1 when the controller is running. Or to 0 for stop, init, or any other state.	0	S
%S13	First cycle in RUN	Normally at 0, this bit is set to 1 by the system during the first scan after the controller has been changed to RUN.	1	S
%S17	Capacity exceeded	Normally set to 0, it is set to 1 by the system: During a rotate or shift operation. The system switches the bit output to 1. It must be tested by the user program, after each operation where there is a risk of an overflow, then reset to 0 by the user if an overflow occurs.	0	S->U
%S18	Arithmetic overflow or error	Normally set to 0. It is set to 1 in the case of an overflow when a 16 bit operation is performed, that is: - A result greater than + 32 767 or less than - 32 768, in single length, - A result greater than + 2 147 483 647 or less than - 2 147 483 648, in double length, - A result greater than + 3.402824E+38 or less than - 3.402824E+38, in floating point, - Division by 0, - The square root of a negative number, - BTI or ITB conversion not significant: BCD value out of limits. It must be tested by the user program, after each operation where there is a risk of an overflow, then reset to 0 by the user if an overflow occurs.	0	S->U
%S19	Scan period overrun (periodic scan)	Normally at 0, this bit is set to 1 by the system in the event of a scan period overrun (scan time greater than the period defined by the user at configuration or programmed in %SW0). This bit is reset to 0 by the user.	0	S->U

System Bit	Function	Description	Init state	Control
%S20	Index overflow	Normally at 0, it is set to 1 when the address of the indexed object becomes less than 0 or more than the maximum size of an object. It must be tested by the user program, after each operation where there is a risk of overflow, then reset to 0 if an overflow occurs.	0	S->U
%S21	GRAFCET initialization	Normally set to 0, it is set to 1 by: • A cold restart, %S0=1, • The user program, in the preprocessing program part only, using a Set Instruction (S %S21) or a set coil -(S)- %S21, • The terminal. At state 1, it causes GRAFCET initialization. Active steps are deactivated and initial steps are activated. It is reset to 0 by the system after GRAFCET initialization.	0	U->S
%S22	GRAFCET reset	Normally set to 0, it can only be set to 1 by the program in pre-processing. At state 1 it causes the active steps of the entire GRAFCET to be deactivated. It is reset to 0 by the system at the start of the execution of the sequential processing.	0	U->S
%S23	Preset and freeze GRAFCET	Normally set to 0, it can only be set to 1 by the program in the pre-processing program module. Set to 1, it validates the pre-positioning of GRAFCET. Maintaining this bit at 1 freezes the GRAFCET (freezes the chart). It is reset to 0 by the system at the start of the execution of the sequential processing to ensure that the GRAFCET chart moves on from the frozen situation.	0	U->S
%S24	Operations Display	Normally at 0, this bit can be set to 1 by the user. • At state 0, the Operator Display is operating normally, • At state 1, the Operator Display is frozen, stays on current display, blinking disabled, and input key processing stopped.	0	U->S

System Bit	Function	Description	Init state	Control
%S25	Choosing a display mode on the operator display	You can choose between two display modes on the 2-line operator display: data mode and normal mode. ● If %S25=0, then normal mode is enabled. On the first line, you can write an object name (a system word, a memory word, a system bit, etc.). On the second line, you can read its value. ● If %S25=1, then data mode is enabled. On the first line, you can display %SW68 value. On the second line, you can display %SW69 value. When %S25=1, the operator keyboard is disabled. Note: Firmware version must be V3.0 or higher.	0	U
%S26	Choosing a signed or unsigned value on the operator display	You can choose between two value types: signed or unsigned. ● If %S26=0, then signed value (-32768 to 32767) display is enabled. +/- signs appear at each start of line. ● If %S26=1, then unsigned value (0 to 65535) display is enabled. %S26 can only be used if %S25=1. Note: Firmware version must be V3.0 or higher.	0	U
%S31	Event mask	Normally at 1. ● Set to 0, events cannot be executed and are queued. ● Set to 1, events can be executed, This bit can be set to its initial state 1 by the user and the system (on cold re-start).	1	U->S
%S38	Permission for events to be placed in the events queue	Normally at 1. ● Set to 0, events cannot be placed in the events queue. ● Set to 1, events are placed in the events queue as soon as they are detected, This bit can be set to its initial state 1 by the user and the system (on cold re-start).	1	U->S
%S39	Saturation of the events queue	Normally at 0. ● Set to 0, all events are reported, ● Set to 1, at least one event is lost. This bit can be set to 0 by the user and the system (on cold re-start).	0	U->S

System Bit	Function	Description	Init state	Control
%S50	Updating the date and time using words %SW49 to %SW53	Normally on 0, this bit can be set to 1 or 0 by the program or the Operator Display. - Set to 0, the date and time can be read, - Set to 1, the date and time can be updated. The controller's internal RTC is updated on a falling edge of %S50.	0	U->S
%S51	Time-of-day clock status	Normally on 0, this bit can be set to 1 or 0 by the program or the Operator Display. - Set to 0, the date and time are consistent, - Set to 1, the date and time must be initialized by the user. When this bit is set to 1, the time of day clock data is not valid. The date and time may never have been configured, the battery may be low, or the controller correction constant may be invalid (never configured, difference between the corrected clock value and the saved value, or value out of range). State 1 transitioning to state 0 forces a write of the correction constant to the RTC.	0	U->S
%S52	RTC = error	This bit managed by the system indicates that the RTC correction has not been entered, and the date and time are false. - Set to 0, the date and time are consistent, - At state 1, the date and time must be initialized.	0	S
%S59	Updating the date and time using word %SW59	Normally on 0, this bit can be set to 1 or 0 by the program or the Operator Display. - Set to 0, the system word %SW59 is not managed, - Set to 1, the date and time are incremented or decremented according to the rising edges on the control bits set in %SW59.	0	U
%S66	BAT LED display enable/disable (only on controllers that support an external battery: TWDLCA•40DRF controllers.)	This system bit can be set by the user. It allows the user to turn on/off the BAT LED: - Set to 0, BAT LED is enabled (it is reset to 0 by the system at power-up), - Set to 1, BAT LED is disabled (LED remains off even if there is a low external battery power or there is no external battery in the compartment).	0	S or U->S
%S69	User STAT LED display	Set to 0, STAT LED is off. Set to 1, STAT LED is on.	0	U

System Bit	Function	Description	Init state	Control
%S75	External battery status (only on controllers that support an external battery: TWDLCA•40DRF controllers.)	This system bit is set by the system. It indicates the external battery status and is readable by the user: - Set to 0, external battery is operating normally, - Set to 1, external battery power is low, or external battery is absent from compartment.	0	S
%S95	Restore memory words	This bit can be set when memory words were previously saved to the internal EEPROM. Upon completion the system sets this bit back to 0 and the number of memory words restored is set in %SW97	0	U
%S96	Backup program OK	This bit can be read at any time (either by the program or while adjusting), in particular after a cold start or a warm restart. - Set to 0, the backup program is invalid. - Set to 1, the backup program is valid.	0	S
%S97	Save %MW OK	This bit can be read at any time (either by the program or while adjusting), in particular after a cold start or a warm restart. - Set to 0, save %MW is not OK. - Set to 1, save %MW is OK.	0	S
%S100	TwidoSoft communications cable connection	Shows whether the TwidoSoft communication cable is connected. - Set to 1, TwidoSoft communications cable is either not attached or TwidoSoft is connected. - Set to 0, TwidoSoft Remote Link cable is connected.	-	S
%S101	Changing a port address (Modbus protocol)	Used to change a port address using system words %SW101 (port 1) and %SW102 (port 2). To do this, %S101 must be set to 1. - Set to 0, the address cannot be changed. The value of %SW101 and %SW102 matches the current port address, - Set to 1, the address can be changed by changing the values of %SW101 (port 1) and %SW102 (port 2). Having modified the values of the system words, %S101 must be set back to 0.	0	U

System Bit	Function	Description	Init state	Control
%S103 %S104	Using the ASCII protocol	Enables the use of the ASCII protocol on Comm 1 (%S103) or Comm 2 (%S104). The ASCII protocol is configured using system words %SW103 and %SW105 for Comm 1, and %SW104 and %SW106 for Comm 2. - Set to 0, the protocol used is the one configured in Twido Soft, - Set to 1, the ASCII protocol is used on Comm 1 (%S103) or Comm 2 (%S104). In this case, the system words %SW103 and %SW105 must be previously configured for Comm 1, and %SW104 and %SW106 for Comm 2.	0	U
%S110	Remote link exchanges	This bit is reset to 0 by the program or by the terminal. - Set to 1 for a master, all remote link exchanges (remote I/O only) are completed. - Set to 1 for a slave, exchange with master is completed.	0	S->U
%S111	Single remote link exchange	- Set to 0 for a master, a single remote link exchange is completed. - Set to 1 for a master, a single remote link exchange is active.	0	S
%S112	Remote link connection	- Set to 0 for a master, the remote link is activated. - Set to 1 for a master, the remote link is deactivated.	0	U
%S113	Remote link configuration/operation	- Set to 0 for a master or slave, the remote link configuration/operation is OK. - Set to 1 for a master, the remote link configuration/operation has an error. - Set to 1 for a slave, the remote link configuration/operation has an error.	0	S->U
%S118	Remote I/O error	Normally set to 1. This bit can be set to 0 when an I/O fault is detected on the remote link.	1	S
%S119	Local I/O error	Normally set to 1. This bit can be set to 0 when an I/O fault is detected on the remote link. %SW118 determines the nature of the fault. Resets to 1 when the fault disappears.	1	S

Table
Abbreviations
Described

Abbreviation table:

Abbreviation	Description
S	Controlled by the system
U	Controlled by the user
U->S	Set to 1 by the user, reset to 0 by the system
S->U	Set to 1 by the system, reset to 0 by the user

System Words (%SW)

Introduction

The following section provides detailed information about the function of the system words and how they are controlled.

Detailed Description

The following table provides detailed information about the function of the system words and how they are controlled:

System Words	Function	Description	Control
%SW0	Controller scan period (periodic task)	Modifies controller scan period defined at configuration through the user program in the Animation Table Editor.	U
%SW1	Save the value of a Periodic event	Modifies the cycle time [5-255 ms] of a Periodic event, without losing the Period value saved in the Periodic event box of the Scan Mode window. Allows you to recover the Period value saved in the Periodic event box: • in case of a cold start, or • if the value you write in %SW1 is outside [5-255] range. %SW1 value can be modified at each end of a cycle, in the program or in the Animation table, without having to stop the program. Cycle times can be correctly observed while the program is running.	U
%SW6	Controller Status	Controller Status: 0 = NO CONFIG 2 = STOP 3 = RUN 4 = HALT	S

System Words	Function	Description	Control
%SW7	Controller state	● Bit [0]: Backup/restore in progress: ● Set to 1 if backup/restore in progress, ● Set to 0 if backup/restore complete or disabled. ● Bit [1]: Controller's configuration OK: ● Set to 1 if configuration ok. ● Bit [3..2] EEPROM status bits: ● 00 = No cartridge ● 01 = 32 Kb EEPROM cartridge ● 10 = 64 Kb EEPROM cartridge ● 11 = Reserved for future use ● Bit [4]: Application in RAM different than EEPROM: ● Set to 1 if RAM application different to EEPROM. ● Bit [5]: RAM application different to cartridge: ● Set to 1 if RAM application different to cartridge. ● Bit [6] not used (status 0) ● Bit [7]: Controller reserved: ● Set to 1 if reserved. ● Bit [8]: Application in Write mode: ● Set to 1 if application is protected. ● Bit [9] not used (status 0) ● Bit [10]: Second serial port installed: ● Set to 1 if installed. ● Bit [11]: Second serial port type: (0 = EIA RS-232, 1 = EIA RS-485): ● Set to 0 = EIA RS-232 ● Set to 1 = EIA RS-485 ● Bit [12]: application valid in internal memory: ● Set to 1 if application valid. ● Bit [13] Valid application in cartridge: ● Set to 1 if application valid. ● Bit [14] Valid application in RAM: ● Set to 1 if application valid. ● Bit [15]: ready for execution: ● Set to 1 if ready for execution.	S
%SW11	Software watchdog value	Contains the maximum value of the watchdog. The value (10 to 500 ms) is defined by the configuration.	U
%SW14	Commercial version, Vxx.yy	For example, if %SW14=0232: ● 8 MSB=02 in hexadecimal, then xx=2 in decimal ● 8 LSB=32 in hexadecimal, then yy=50 in decimal As a result, Commercial version is V2.50. Note: Firmware version must be 2.5 or higher.	S

System Words	Function	Description	Control
%SW15	Firmware patch, Pzz	For example, if %SW15=0005: 8 MSB is not used 8 LSB=05 in hexadecimal, then zz=5 in decimal As a result, Firmware patch is P05. Note: Firmware version must be 2.5 or higher.	S
%SW16	Firmware version, Vxx.yy	For example, if %SW16=0232: 8 MSB=02 in hexadecimal, then xx=2 in decimal 8 LSB=32 in hexadecimal, then yy=50 in decimal As a result, Firmware version is V2.50. Note: Firmware version must be 2.5 or higher.	S
%SW17	Default status for floating operation	When a fault is detected in a floating arithmetic operation, bit %S18 is set to 1 and the default status of %SW17 is updated according to the following coding: - Bit [0]: Invalid operation, result is not a number (1.#NAN or -1.#NAN), - Bit 1: Reserved, - Bit 2: Divided by 0, result is infinite (-1.#INF or 1.#INF), - Bit 3: Result greater in absolute value than +3.402824e+38, result is infinite (-1.#INF or 1.#INF).	S and U
%SW18- %SW19	100 ms absolute timer counter	The counter works using two words: %SW18 represents the least significant word, %SW19 represents the most significant word.	S and U
%SW20 to %SW27	Provides status for CANopen slave modules with node address 1 to 16.	For more details, please refer to <i>CANopen Slave Reserved Specific System Words</i> , p. 271.	S
%SW30	Last scan time	Shows execution time of the last controller scan cycle (in ms). Note: This time corresponds to the time elapsed between the start (acquisition of inputs) and the end (update of outputs) of a scan cycle.	S

System Words	Function	Description	Control
%SW31	Max scan time	Shows execution time of the longest controller scan cycle since the last cold start (in ms). Notes: • This time corresponds to the time elapsed between the start (acquisition of inputs) and the end (update of outputs) of a scan cycle. • To allow proper detection of a pulse signal when the latching input option is selected, the pulse width (T_{ON}) and the cyclic period (T_{pulse}) must meet the following two requirements: • $T_{ON} \geq 1 \text{ ms}$ • The input signal cyclic period must follow the Nyquist-Shannon sampling rule stating that the cyclic period (T_{pulse}) of the input signal must be at least twice the maximum program scan time (%SW31): $T_{pulse} \geq 2 \times \%SW31$. Note: If this condition is not fulfilled, some pulses may be missed.	S
%SW32	Min. scan time	Shows execution time of shortest controller scan cycle since the last cold start (in ms). Note: This time corresponds to the time elapsed between the start (acquisition of inputs) and the end (update of outputs) of a scan cycle.	S
%SW48	Number of events	Shows how many events have been executed since the last cold start. (Counts all events except periodic events.) Note: Set to 0 (after application loading and cold start), increments on each event execution.	S

System Words	Function	Description	Control
%SW49 %SW50 %SW51 %SW52 %SW53	Real-Time Clock (RTC)	RTC Functions: words containing current date and time values (in BCD):	S and U
		%SW49 xN Day of the week (N=1 for Monday)	
		%SW50 00SS Seconds	
		%SW51 HHMM Hour and minute	
		%SW52 MMDD Month and day	
		%SW53 CCYY Century and year	
		These words are controlled by the system when bit %S50 is at 0. These words can be written by the user program or by the terminal when bit %S50 is set to 1. On a falling edge of %S50 the controller's internal RTC is updated from the values written in these words.	
%SW54 %SW55 %SW56 %SW57	Date and time of the last stop	System words containing the date and time of the last power failure or controller stop (in BCD):	S
		%SW54 SS Seconds	
		%SW55 HHMM Hour and minute	
		%SW56 MMDD Month and day	
		%SW57 CCYY Century and year	
%SW58	Code of last stop	Displays code giving cause of last stop:	S
		1 = Run/Stop input edge	
		2 = Stop at software fault (controller scan overshoot)	
		3 = Stop command	
		4 = Power outage	
		5 = Stop at hardware fault	

System Word	Function	Description	Control
%SW59	Adjust current date	Adjusts the current date. Contains two sets of 8 bits to adjust current date. The operation is always performed on rising edge of the bit. This word is enabled by bit %S59.	U
		Increment	
		Decrement	
		Parameter	
		bit 0	
		bit 8	
		Day of week	
		bit 1	
		bit 9	
		Seconds	
		bit 2	
		bit 10	
		Minutes	
		bit 3	
		bit 11	
		Hours	
		bit 4	
		bit 12	
		Days	
		bit 5	
		bit 13	
		Month	
		bit 6	
		bit 14	
		Years	
		bit 7	
		bit 15	
		Centuries	
%SW60	RTC correction	RTC correction value	U
%SW63	EXCH1 block error code	EXCH1 error code: 0 - operation was successful 1 – number of bytes to be transmitted is too great (> 250) 2 - transmission table too small 3 - word table too small 4 - receive table overflowed 5 - time-out elapsed 6 - transmission 7 - bad command within table 8 - selected port not configured/available 9 - reception error 10 - can not use %KW if receiving 11 - transmission offset larger than transmission table 12 - reception offset larger than reception table 13 - controller stopped EXCH processing	S
%SW64	EXCH2 block error code	EXCH2 error code: See %SW63.	S

System Word	Function	Description	Control
%SW65	EXCH3 block error code	EXCH3 error code is implemented on Ethernet-capable TWDLCAE40DRF Twido controllers only 1-4, 6-13: See %SW63. (Note that error code 5 is invalid and replaced by the Ethernet-specific error codes 109 and 122 described below.) The following are Ethernet-specific error codes: 101 - no such IP address 102 - the TCP connection is broken 103 - no socket available (all connection channels are busy) 104 - network is down 105 - network cannot be reached 106 - network dropped connection on reset 107 - connection aborted by peer device 108 - connection reset by peer device 109 - connection time-out elapsed 110 - rejection on connection attempt 111 - host is down 120 - unknown index (remote device is not indexed in configuration table) 121 - fatal (MAC, Chip, Duplicated IP) 122 - receiving timed-out elapsed after data was sent 123 - Ethernet initialization in progress	S
%SW67	Function and type of controller	Contains the following information: ● Controller type bits [0 -11] ● 8B0 = TWDLC•A10DRF ● 8B1 = TWDLC•A16DRF ● 8B2 = TWDLMDA20DUK/DTK ● 8B3 = TWDLC•A24DRF ● 8B4 = TWDLMDA40DUK/DTK ● 8B6 = TWDLMDA20DRT ● 8B8 = TWDLCAA40DRF ● 8B9 = TWDLCAE40DRF ● Bit 12,13,14,15 not used = 0	S

System Words	Function	Description	Control
%SW68 and %SW69	Elements to be displayed simultaneously on the 2-line operator display	If %S25=1, then data display mode is enabled. The operator keyboard is disabled. %SW68 and %SW69 can be displayed on the 2-line operator display: ● %SW68 value on the first line, ● %SW69 value on the second line. Note: Firmware version must be V3.0 or higher.	U

System Words	Function	Description	Control
%SW73 and %SW74	AS-Interface System State	● Bit [0]: Set to 1 if configuration OK. ● Bit [1]: Set to 1 if data exchange enabled. ● Bit [2]: Set to 1 if module in Offline mode. ● Bit [3]: Set to 1 if ASI_CMD instruction terminated. ● Bit [4]: Set to 1 error in ASI_CMD instruction in progress.	S and U
%SW76 to %SW79	Down counters 1-4	These 4 words serve as 1 ms timers. They are decremented individually by the system every ms if they have a positive value. This gives 4 down counters down counting in ms which is equal to an operating range of 1 ms to 32767 ms. Setting bit 15 to 1 can stop decrementation.	S and U
%SW80	Base I/O Status	Bit [0] Channels in normal operation (for all its channels) Bit [1] Module under initialization (or of initializing information of all channels) Bit [2] Hardware failure (external power supply failure, common to all channels) Bit [3] Module configuration fault Bit [4] Converting data input channel 0 in progress Bit [5] Converting data input channel 1 in progress Bit [6] Input thermocouple channel 0 not configured Bit [7] Input thermocouple channel 1 not configured Bit [8] Not used Bit [9] Unused Bit [10] Analog input data channel 0 over range Bit [11] Analog input data channel 1 over range Bit [12] Incorrect wiring (analog input data channel 0 below current range, current loop open) Bit [13] Incorrect wiring (analog input data channel 1 below current range, current loop open) Bit [14] Unused Bit [15] Output channel not available	S
%SW81	● Expansion I/O Module 1 Status: Same definitions as %SW80 ● CANopen Master Module Status at Expansion Address 1: ● Bit [0] Configuration state (1 = configuration OK; 0 = configuration error) - Bit [1] Operational state (1 = PDO exchange ON; 0 = PDO exchange OFF) - Bit [2] Init state (1 = init state ON; 0 = init state OFF) - Bit [3] CAN_CMD instruction complete (1 = complete; 0 = in progress) - Bit [4] CAN_CMD instruction error (1 = error; 0 = OK) - Bit [5] Initialization error (1 = error; 0 = OK) - Bit [6] Loss of message, power supply error (1 = error; 0 = OK)		S
%SW82	Expansion I/O Module 2 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 2: Same definitions as %SW81		S

System Words	Function	Description	Control
%SW83		Expansion I/O Module 3 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 3: Same definitions as %SW81	S
%SW84		Expansion I/O Module 4 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 4: Same definitions as %SW81	S
%SW85		Expansion I/O Module 5 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 5: Same definitions as %SW81	S
%SW86		Expansion I/O Module 6 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 6: Same definitions as %SW81	S
%SW87		Expansion I/O Module 7 Status: Same definitions as %SW80 CANopen Master Module Status at Expansion Address 7: Same definitions as %SW81	S
%SW94	Application's signature	In case of an application change, in terms of configuration or programming data, the signature (sum of all checksums) changes consequently. If %SW94=91F3 in hexadecimal, the application's signature is 91F3 in hexadecimal. Note: Firmware version must be V2.5 or higher.	S
%SW96	Command and/or diagnostics for save/restore function of application program and %MW.	- Bit [0]: Indicates that the %MW memory words must be saved to EEPROM: - Set to 1 if a backup is required, - Set to 0 if the backup in progress is not complete. - Bit [1]: This bit is set by the firmware to indicate when the save is complete: - Set to 1 if the backup is complete, - Set to 0 if a new backup request is asked for. - Bit [2]: Backup error, refer to bits 8, 9, 10 and 14 for further information: - Set to 1 if an error appeared, - Set to 0 if a new backup request is asked for. - Bit [6]: Set to 1 if the controller contains an empty application. - Bit [8]: Indicates that the number of %MWs specified in %SW97 is greater than the number of %MWs configured in the application: - Set to 1 if an error is detected, - Bit [9]: Indicates that the number of %MWs specified in %SW97 is greater than the maximum number of %MWs that can be defined by any application in TwidoSoft. - Set to 1 if an error is detected, - Bit [10]: Difference between internal RAM and internal EEPROM (1 = yes). - Set to 1 if there is a difference. - Bit [14]: Indicates if an EEPROM write fault has occurred: - Set to 1 if an error is detected,	S and U

System Words	Function	Description	Control
%SW97	Command or diagnostics for save/restore function	When saving memory words, this value represents the physical number %MW to be saved to internal EEPROM. When restoring memory words, this value is updated with the number of memory words restored to RAM. For the save operation, when this number is set to 0, memory words will not be stored. The user must define the user logic program. Otherwise, this program is set to 0 in the controller application, except in the following case: On cold start, this word is set to -1 if the internal Flash EEPROM has no saved memory word %MW file. In the case of a cold start where the internal Flash EEPROM contains a memory word %MW list, the value of the number of saved memory words in the file must be set in this system word %SW97.	S and U

System Words	Function	Description	Control
%SW101 %SW102	Value of the port's Modbus address	When bit %S101 is set to 1, you can change the Modbus address of port 1 or port 2. The address of port 1 is %SW101, and that of port 2 is %SW102.	S

System Words	Function	Description	Control																																
%SW103 %SW104	Configuration for use of the ASCII protocol	When bit %S103 (Comm 1) or %S104 (Comm 2) is set to 1, the ASCII protocol is used. System word %SW103 (Comm 1) or %SW104 (Comm 2) must be set according to the elements below: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="8">End of the character string</td><td>Data bit</td><td>Stop bit</td><td colspan="2">Parity</td><td>RTS / CTS</td><td colspan="3">Baud rate</td></tr></table> ● Baud rate: ● 0: 1200 bauds, ● 1: 2400 bauds, ● 2: 4800 bauds, ● 3: 9600 bauds, ● 4: 19200 bauds, ● 5: 38400 bauds. ● RTS/CTS: ● 0: disabled, ● 1: enabled. ● Parity: ● 00: none, ● 10: odd, ● 11: even. ● Stop bit: ● 0: 1 stop bit, ● 1: 2 stop bits. ● Data bits: ● 0: 7 data bits, ● 1: 8 data bits.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	End of the character string								Data bit	Stop bit	Parity		RTS / CTS	Baud rate			S
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
End of the character string								Data bit	Stop bit	Parity		RTS / CTS	Baud rate																						
%SW105 %SW106	Configuration for use of the ASCII protocol	When bit %S103 (Comm 1) or %S104 (Comm 2) is set to 1, the ASCII protocol is used. System word %SW105 (Comm 1) or %SW106 (Comm 2) must be set according to the elements below: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="8">Timeout frame in ms</td><td colspan="8">Timeout response in multiple of 100 ms</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Timeout frame in ms								Timeout response in multiple of 100 ms								S
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
Timeout frame in ms								Timeout response in multiple of 100 ms																											
%SW111	Remote link status	Indication: Bit 0 corresponds to remote controller 1, bit 1 to remote controller 2, etc. Bit [0] to [6]: ● Set to 0 = remote controller 1-7 absent ● Set to 1 = remote controller 1-7 present Bit [8] to bit [14]: ● Set to 0 = remote I/O detected on remote controller 1-7 ● Set to 1 = extension controller detected on remote controller 1-7	S																																

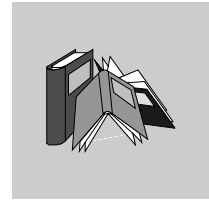
System Words	Function	Description	Control
%SW112	Remote Link configuration/ operation error code	00: successful operations 01: timeout detected (slave) 02: checksum error detected (slave) 03: configuration mismatch (slave) This is set to 1 by the system and must be reset by the user.	S
%SW113	Remote link configuration	Indication: Bit 0 corresponds to remote controller 1, bit 1 to remote controller 2, etc. Bit [0] to [6]: - Set to 0 = remote controller 1-7 not configured - Set to 1 = remote controller 1-7 configured Bit [8] to bit [14]: - Set to 0 = remote I/O configured as remote controller 1-7 - Set to 1 = peer controller configured as remote controller 1-7	S
%SW114	Enable schedule blocks	Enables or disables operation of schedule blocks by the user program or operator display. Bit 0: 1 = enables schedule block #0 ... Bit 15: 1 = enables schedule block #15 Initially all schedule blocks are enabled. If schedule blocks are configured the default value is FFFF If no schedule blocks are configured the default value is 0.	S and U
%SW118	Base controller status word	Shows faults detected on master controller. Bit 9: 0 = External fault or comm. Fault Bit 12: 0 = RTC not installed Bit 13: 0 = Configuration fault (I/O extension configured but absent or faulty). All the other bits of this word are set to 1 and are reserved. For a controller which has no fault, the value of this word is FFFFh.	S
%SW120	Expansion I/O module health	One bit per module. Address 0 = Bit 0 1 = Unhealthy 0 = OK	S

**Table
Abbreviations
Described**

Abbreviation table:

Abbreviation	Description
S	Controlled by the system
U	Controlled by the user

Glossary



!

% Prefix that identifies internal memory addresses in the controller that are used to store the value of program variables, constants, I/O, and so on.

A

Addresses	Internal registers in the controller used to store values for program variables, constants, I/O, and so on. Addresses are identified with a percentage symbol (%) prefix. For example, %I0.1 specifies an address within the controller RAM memory containing the value for input channel 1.
Analog potentiometer	An applied voltage that can be adjusted and converted into a digital value for use by an application.
Analyze program	A command that compiles a program and checks for program errors: syntax and structure errors, symbols without corresponding addresses, resources used by the program that are not available, and if the program does not fit in available controller memory. Errors are displayed in the Program Errors Viewer.
Animation table	Table created within a language editor or an operating screen. When a PC is connected to the controller, provides a view of controller variables and allows values to be forced when debugging. Can be saved as a separate file with an extension of .tat.
Animation Tables Editor	A specialized window in the TwidoSoft application for viewing and creating Animation Tables.

Application	A TwidoSoft application consists of a program, configuration data, symbols, and documentation.
Application browser	A specialized window in the TwidoSoft that displays a graphical tree-like view of an application. Provides for convenient configuration and viewing of an application.
Application file	Twido applications are stored as file type .twd.
ASCII	(American Standard Code for Information Interchange) Communication protocol for representing alphanumeric characters, notably letters, figures and certain graphic and control characters.
Auto line validate	When inserting or modifying List instructions, this optional setting allows for program lines to be validated as each is entered for errors and unresolved symbols. Each element must be corrected before you can exit the line. Selected using the Preferences dialog box.
Auto load	A feature that is always enabled and provides for the automatic transfer of an application from a backup cartridge to the controller RAM in case of a lost or corrupted application. At power up, the controller compares the application that is presently in the controller RAM to the application in the optional backup memory cartridge (if installed). If there is a difference, then the copy in the backup cartridge is copied to the controller and the internal EEPROM. If the backup cartridge is not installed, then the application in the internal EEPROM is copied to the controller.

B

Backup	A command that copies the application in controller RAM into both the controller internal EEPROM and the optional backup memory cartridge (if installed).
BootP	A UDP/IP-based protocol (Bootstrap Protocol) which allows a booting host to configure itself dynamically and without user supervision. BootP provides a means to notify a host of its assigned IP address.

C

CAN	Controller Area Network: field bus originally developed for automobile applications which is now used in many sectors, from industrial to tertiary.
CiA	CAN in Automation: international organization of users and manufacturers of CAN products.

Client	A computer process requesting service from other computer processes.
COB	Communication OBject: transport unit on CAN bus. A COB is identified by a unique identifier, which is coded on 11 bits, [0, 2047]. A COB contains a maximum of 8 data bytes. The priority of a COB transmission is shown by its identifier - the weaker the identifier, the more priority the associated COB has.
Coil	A ladder diagram element representing an output from the controller.
Cold start or restart	A start up by the controller with all data initialized to default values, and the program started from the beginning with all variables cleared. All software and hardware settings are initialized. A cold restart can be caused by loading a new application into controller RAM. Any controller without battery backup always powers up in Cold Start.
Comment lines	In List programs, comments can be entered on separate lines from instructions. Comments lines do not have line numbers, and must be inserted within parenthesis and asterisks such as: (*COMMENTS GO HERE*).
Comments	Comments are texts you enter to document the purpose of a program. For Ladder programs, enter up to three lines of text in the Rung Header to describe the purpose of the rung. Each line can consist of 1 to 64 characters. For List programs, enter text on n unnumbered program line. Comments must be inserted within parenthesis and asterisks such as: (*COMMENTS GO HERE*).
Compact controller	Type of Twido controller that provides a simple, all-in-one configuration with limited expansion. Modular is the other type of Twido controller.
Configuration editor	Specialized TwidoSoft window used to manage hardware and software configuration.
Constants	A configured value that cannot be modified by the program being executed.
Contact	A ladder diagram element representing an input to the controller.
Counter	A function block used to count events (up or down counting).
Cross references	Generation of a list of operands, symbols, line/rung numbers, and operators used in an application to simplify creating and managing applications.
Cross References Viewer	A specialized window in the TwidoSoft application for viewing cross references.

D

Data variable	See Variable.
Date/Clock functions	Allow control of events by month, day of month, and time of day. See Schedule Blocks.
Default gateway	The IP address of the network or host to which all packets addressed to an unknown network or host are sent. The default gateway is typically a router or other device.
Drum controller	A function block that operates similar to an electromechanical drum controller with step changes associated with external events.

E

EDS	Electronic Data Sheet: description file for each CAN device (provided by the manufacturers).
EEPROM	Electrically Erasable Programmable Read-Only Memory. Twido has an internal EEPROM and an optional external EEPROM memory cartridge.
Erase	This command deletes the application in the controller, and has two options: • To delete the contents of the controller RAM, the controller internal EEPROM, and the installed optional backup cartridge. • To delete the contents of the installed optional backup cartridge only.
Executive loader	A 32-Bit Windows application used for downloading a new Firmware Executive program to a Twido controller.
Expansion bus	Expansion I/O Modules connect to the base controller using this bus.
Expansion I/O modules	Optional Expansion I/O Modules are available to add I/O points to a Twido controller. (Not all controller models allow expansion).

F

Fast counters	A function block that provides for faster up/down counting than available with the Counters function block. A Fast Counter can count up to a rate of 5 KHz.
FIFO	First In, First Out. A function block used for queue operations.
Firmware executive	The Firmware Executive is the operating system that executes your applications and manages controller operation.
Forcing	Intentionally setting controller inputs and outputs to 0 or 1 values even if the actual values are different. Used for debugging while animating a program.
Frame	A group of bits which form a discrete block of information. Frames contain network control information or data. The size and composition of a frame is determined by the network technology being used.
Framing types	Two common framing types are Ethernet II and IEEE 802.3.
Function block	A program unit of inputs and variables organized to calculate values for outputs based on a defined function such as a timer or a counter.

G

Gateway	A device which connects networks with dissimilar network architectures and which operates at the Application Layer. This term may refer to a router.
Grafcet	Grafcet is used to represent the functioning of a sequential operation in a structured and graphic form. This is an analytical method that divides any sequential control system into a series of steps, with which actions, transitions, and conditions are associated.

H

Host	A node on a network.
Hub	A device which connects a series of flexible and centralized modules to create a network.

I

Init state	The operating state of TwidoSoft that is displayed on the Status Bar when TwidoSoft is started or does not have an open application.
Initialize	A command that sets all data values to initial states. The controller must be in Stop or Error mode.
Instance	A unique object in a program that belongs to a specific type of function block. For example, in the timer format %T <i>Mi</i> , <i>i</i> is a number representing the instance.
Instruction List language	A program written in instruction list language (IL) is composed of a series of instructions executed sequentially by the controller. Each instruction is composed of a line number, an instruction code, and an operand.
Internet	The global interconnection of TCP/IP based computer communication networks.
IP	Internet Protocol. A common network layer protocol. IP is most often used with TCP.
IP Address	Internet Protocol Address. A 32-bit address assigned to hosts using TCP/IP.

L

Ladder editor	Specialized TwidoSoft window used to edit a Ladder program.
Ladder language	A program written in Ladder language is composed of graphical representation of instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller.
Ladder list rung	Displays parts of a List program that are not reversible to Ladder language.
Latching input	Incoming pulses are captured and recorded for later examination by the application.
LIFO	Last In, First Out. A function block used for stack operations.
List editor	Simple program editor used to create and edit a List program.

M

MAC Address	Media Access Control address. The hardware address of a device. A MAC address is assigned to an Ethernet TCP/IP module in the factory.
Master controller	A Twido controller configured to be the Master on a Remote Link network.
MBAP	Modbus Application Protocol
Memory cartridge	Optional Backup Memory Cartridges that can be used to backup and restore an application (program and configuration data). There are two sizes available: 32 and 64 Kb.
Memory usage indicator	A portion of the Status Bar in the TwidoSoft main window that displays a percentage of total controller memory used by an application. Provides a warning when memory is low.
Modbus	A master-slave communications protocol that allows one single master to request responses from slaves.
Modular controller	Type of Twido controller that offers flexible configuration with expansion capabilities. Compact is the other type of Twido controller.
Monitor state	The operating state of TwidoSoft that is displayed on the Status Bar when a PC is connected to a controller in a non-write mode.

N

Network	Interconnected devices sharing a common data path and protocol for communication.
Node	An addressable device on a communications network.

O

Offline operation	An operation mode of TwidoSoft when a PC is not connected to the controller and the application in PC memory is not the same as the application in controller memory. You create and develop an application in Offline operation.
Offline state	The operating state of TwidoSoft that is displayed on the Status Bar when a PC is not connected to a controller.
Online operation	An operation mode of TwidoSoft when a PC is connected to the controller and the application in PC memory is the same as the application in controller memory. Online operation can be used to debug an application.
Online state	The operating state of TwidoSoft that is displayed on the Status Bar when a PC is connected to the controller.
Operand	A number, address, or symbol representing a value that a program can manipulate in an instruction.
Operating states	Indicates the TwidoSoft state. Displayed in the status bar. There are four operating states: Initial, Offline, Online, and Monitor.
Operator	A symbol or code specifying the operation to be performed by an instruction.

P

Packet	The unit of data sent across a network.
PC	Personal Computer.
Peer controller	A Twido controller configured as a slave on a Remote Link network. An application can be executed in the Peer Controller memory and the program can access both local and expansion I/O data, but I/O data can not be passed to the Master Controller. The program running in the Peer Controller passes information to the Master Controller by using network words (%INW and %QNW).
PLC	Twido programmable controller. There are two types of controllers: Compact and Modular.

PLS	Pulse Generation. A function block that generates a square wave with a 50% on and 50% off duty cycle.
Preferences	A dialog box with selectable options for setting up the List and Ladder program editors.
Program errors viewer	Specialized TwidoSoft window used to view program errors and warnings.
Programmable controller	A Twido controller. There are two types of controllers: Compact and Modular.
Protection	Refers to two different types of application protection: password protection which provides access control, and controller application protection which prevents all reads and writes of the application program.
Protocol	Describes message formats and a set of rules used by two or more devices to communicate using those formats.
PWM	Pulse Width Modulation. A function block that generates a rectangular wave with a variable duty cycle that can be set by a program.

R

RAM	Random Access Memory. Twido applications are downloaded into internal volatile RAM to be executed.
Real-time clock	An option that will keep the time even when the controller is not powered for a limited amount of time.
Reflex output	In a counting mode, the very fast counter's current value (%VFC.V) is measured against its configured thresholds to determine the state of these dedicated outputs.
Registers	Special registers internal to the controller dedicated to LIFO/FIFO function blocks.
Remote controller	A Twido controller configured to communicate with a Master Controller on a Remote Link network.

Remote link	High-speed master/slave bus designed to communicate a small amount of data between a Master Controller and up to seven Remote Controllers (slaves). There are two types of Remote Controllers that can be configured to transfer data to a Master Controller: a Peer Controller that can transfer application data, or a Remote I/O Controller that can transfer I/O data. A Remote link network can consist of a mixture of both types.
Resource manager	A component of TwidoSoft that monitors the memory requirements of an application during programming and configuring by tracking references to software objects made by an application. An object is considered to be referenced by the application if it is used as an operand in a list instruction or ladder rung. Displays status information about the percentage of total memory used, and provides a warning if memory is getting low. See Memory Usage Indicator.
Reversible instructions	A method of programming that allows instructions to be viewed alternately as List instructions or Ladder rungs.
Router	A device that connects two or more sections of a network and allows information to flow between them. A router examines every packet it receives and decides whether to block the packet from the rest of the network or transmit it. The router will attempt to send the packet through the network by the most efficient path.
RTC	See Real-Time Clock.
RTU	Remote Terminal Unit. A protocol using eight bits that is used for communicating between a controller and a PC.
Run	A command that causes the controller to run an application program.
Rung	A rung is located between two potential bars in a grid and is composed of a group of graphical elements joined to each other by horizontal and vertical links. The maximum dimensions of a rung are seven rows and eleven columns.
Rung header	A panel that appears directly over a Ladder rung and can be used to document the purpose of the rung.

S

Scan	A controller scans a program and essentially performs three basic functions. First, it reads inputs and places these values in memory. Next, it executes the application program one instruction at a time and stores results in memory. Finally, it uses the results to update outputs.
-------------	--

Scan mode	Specifies how the controller scans a program. There are two types of scan modes: Normal (Cyclic), the controller scans continuously, or Periodic, the controller scans for a selected duration (range of 2 - 150 msec) before starting another scan.
Schedule blocks	A function block used to program Date and Time functions to control events. Requires Real-Time Clock option.
Server	A computer process that provides services to clients. This term may also refer to the computer process on which the service is based.
Step	A Grafcet step designates a state of sequential operation of automation.
Stop	A command that causes the controller to stop running an application program.
Subnet	A physical or logical network within an IP network, which shares a network address with other portions of the network.
Subnet mask	A bit mask used to identify or determine which bits in an IP address correspond to the network address and which bits correspond to the subnet portions of the address. The subnet mask is the network address plus the bits reserved for identifying the subnetwork.
Switch	A network device which connects two or more separate network segments and allows traffic to be passed between them. A switch determines whether a frame should be blocked or transmitted based on its destination address.
Symbol	A symbol is a string of a maximum of 32 alphanumeric characters, of which the first character is alphabetic. It allows you to personalize a controller object to facilitate the maintainability of the application.
Symbol table	A table of the symbols used in an application. Displayed in the Symbol Editor.

T

TCP	Transmission Control Protocol.
TCP/IP	A protocol suite consisting of the Transmission Control Protocol and the Internet Protocol; the suite of communications protocols on which the Internet is based.
Threshold outputs	Coils that are controlled directly by the very fast counter (%VFC) according to the settings established during configuration.

Timer	A function block used to select a time duration for controlling an event.
Twido	A line of Schneider Electric controllers consisting of two types of controllers (Compact and Modular), Expansion Modules to add I/O points, and options such as Real-Time Clock, communications, operator display, and backup memory cartridges.
TwidoSoft	A 32-Bit Windows, graphical development software for configuring and programming Twido controllers.

U

UDP	A communications protocol (User Datagram Protocol) that is the part of the TCP/IP suite used by applications to transfer datagrams. UDP is also the part of TCP/IP responsible for port addresses.
Unresolved symbol	A symbol without a variable address.

V

Variable	Memory unit that can be addressed and modified by a program.
Very fast counter:	A function block that provides for faster counting than available with Counters and Fast Counters function blocks. A Very Fast Counter can count up to a rate of 20 KHz.

W

Warm restart	A power-up by the controller after a power loss without changing the application. Controller returns to the state which existed before the power loss and completes the scan which was in progress. All of the application data is preserved. This feature is only available on modular controllers.
---------------------	--

Index



Symbols

-	568	%S20	598
%Ci	398	%S21	68, 598
%DR	454	%S22	68, 598
%FC	459	%S23	68, 598
%INW	42	%S24	598
%MSG	477	%S25	599
%MSG3 function block		%S26	599
Instruction	180	%S31	599
%PLS	451	%S38	599
%PWM	448	%S39	599
%QNW	42	%S4	596
%S	596	%S5	596
%S0	596	%S50	600
%S1	596	%S51	600
%S10	597	%S52	600
%S100	601	%S59	600
%S101	601	%S6	596
%S103	602	%S66	600
%S104	602	%S69	600
%S11	597	%S7	596
%S110	602	%S75	601
%S111	602	%S8	596
%S112	602	%S9	596
%S113	602	%S95	601
%S118	602	%S96	601
%S119	602	%S97	601
%S12	597	%SBR	404
%S13	597	%SCi	406
%S17	597	%SW	604
%S18	597	%SW0	604
%S19	597	%SW1	604
		%SW101	613

%SW102, 613
%SW103, 614
%SW104, 614
%SW105, 614
%SW106, 614
%SW11, 605
%SW111, 614
%SW112, 615
%SW113, 615
%SW114, 615
%SW118, 615
%SW120, 615
%SW14, 605
%SW15, 606
%SW16, 606
%SW17, 606
%SW18, 606
%SW19, 606
%SW20..%SW27, 271, 606
%SW30, 606
%SW31, 607
%SW32, 607
%SW48, 607
%SW49, 608
%SW50, 608
%SW51, 608
%SW52, 608
%SW53, 608
%SW54, 608
%SW55, 608
%SW56, 608
%SW57, 608
%SW58, 608
%SW59, 609
%SW6, 604
%SW60, 609
%SW63, 609
%SW64, 609
%SW65, 610
%SW67, 610
%SW68, 610
%SW69, 610
%SW7, 605
%SW73, 611
%SW74, 611
%SW76, 611

%SW77, 611
%SW78, 611
%SW79, 611
%SW80, 611
%SW81..%SW87, 270, 611
%SW94, 612
%SW96, 612
%SW97, 613
%TM, 395
%VFC, 462
*, 568
+, 568
/, 568

A

ABS, 568
Absolute value, 418
Accessing debugging
 PID, 540
Accessing the configuration
 PID, 524
Accumulator, 346
ACOS, 572
Action Zone, 326
Add, 418
Addressing analog I/O modules, 191
Addressing I/O, 40
Advanced function blocks
 Bit and word objects, 438
 Programming principles, 440
Analog Channel, 188
Analog Module
 operating, 190
Analog module
 Example, 199
Analog Modules
 Configuring I/O, 192
Analog modules
 addressing, 191
AND instructions, 378
Animation tab
 PID, 541
Arithmetic Instructions, 418

- ASCII
 - Communication, 84
 - Communications, 115
 - Configuring the port, 118
 - Hardware configuration, 116
 - Software configuration, 117
 - ASCII Link
 - Example, 123
 - ASIN, 572
 - AS-Interface Bus V2
 - configuration screen, 207
 - AS-Interface V2 bus
 - accepting the new configuration, 223
 - Changing a slave address, 218
 - Debug screen, 215
 - Explicit exchanges, 229
 - Faulty slave, 227
 - general functional description, 203
 - I/O addressing, 228
 - Implicit exchanges, 228
 - Operating mode, 233
 - Presentation, 202
 - Programming and diagnostics for the AS-Interface bus, 229
 - Slave diagnostics, 217
 - Slave insertion, 226
 - software configuration, 209
 - software set up principle, 206
 - transfer of a slave image, 221
 - Assignment instructions, 376
 - Numerical, 411
 - AT tab
 - PID, 532
 - ATAN, 572
- B**
- Backup and restore
 - 32K backup cartridge, 56
 - 64K extended memory cartridge, 58
 - memory structure, 52
 - without cartridges, 54
 - Basic function blocks, 386
 - Bit objects, 438
 - Addressing, 36
 - Overview, 27
 - Bit strings, 45
 - BLK, 338
 - Blocks
 - in Ladder diagrams, 328
 - Boolean accumulator, 346
 - Boolean instructions, 370
 - Assignment, 376
 - OR, 380
 - Understanding the format used in this manual, 372
 - BootP, 161
 - Boot-up, 242
 - Bus AS-Interface V2
 - automatic slave addressing, 225
 - Bus AS-Interface V2 bus
 - debugging the bus, 220
- C**
- Calculation, 418
 - CAN bus line, 239
 - CAN_CMD, 273
 - CAN-high, 239
 - CAN-low, 239
 - CANopen
 - Description, 239
 - The protocol, 239
 - CANopen bus
 - configuration methodology, 254
 - CANopen fieldbus
 - Explicit exchanges, 270
 - Implicit exchanges, 269
 - Programming and diagnostics for the CANopen fieldbus, 270
 - CANopen master
 - PDO addressing, 269
 - Checking scan time, 67
 - Clock functions
 - Overview, 481
 - Schedule blocks, 482
 - Setting date and time, 487
 - time and date stamping, 485
 - Closed loop adjustment, 557
 - Coils, 328
 - graphic elements, 331
 - Cold start, 73

- Communication by modem, 86
- Communication overview, 84
- Communications
 - ASCII, 115
 - Modbus, 126
 - Remote Link, 104
- Communications cable connection, 86
- Comparison block
 - graphic element, 332
- Comparison blocks, 329
- Comparison Instructions, 416
- Configuration
 - PID, 524
- Configuring
 - A port for ASCII, 118
 - Port for Modbus, 130
 - Transmission/Reception table for ASCII, 119
- Connections management, 176
- Contacts, 328
 - graphic element, 330
- Control parameters
 - ASCII, 119
- Control table
 - Modbus, 132
- Conversion instructions, 425
- COS, 572
- Counters, 398
 - Programming and configuring, 402

D

- Debugging
 - PID, 540
- Decrement, 418
- DEG_TO_RAD, 574
- Derivative action, 561
- DINT_TO_REAL, 575
- Direct labeling, 48
- Divide, 418
- Documenting your program, 340
- Double word objects, 44
 - Addressing, 39
 - Overview, 32
- Drum controller function block, 454

- Drum controllers
 - programming and configuring, 457

E

- Edge detection
 - falling, 371
 - Rising, 370
- END Instructions, 429
- END_BLK, 338
- EQUAL_ARR, 580
- error, 420
- Ethernet
 - Connections management, 176
 - Network connection, 159
 - TCP/IP setup, 165
- Event tasks
 - Different event sources, 79
 - Event management, 80
 - Overview, 78
- Example
 - Up/Down Counter, 403
- EXCH, 476
- EXCH instruction, 476
- EXCH3, 180
 - Error code, 183
- Exchange function block, 477
- Exclusive OR, instructions, 382
- EXP, 568
- EXPT, 568

F

- Fast counter function block, 459
- FIFO
 - introduction, 443
 - operation, 445
- FIND_, 582
- Floating objects
 - Addressing, 38
- Floating point objects
 - Overview, 32
- Function Blocks
 - PWM, 448

Function blocks
 Counters, 398
 Drum controller, 457
 drum controller, 454
 graphic element, 332
 in programming grid, 328
 Overview of basic function blocks, 386
 programming standard function blocks, 388
 registers, 443
 Schedule blocks, 482
 Shift Bit Register (%SBR), 404
 Step counter (%SCi), 406
 timers, 390, 395

G

Gateway address, 160
 General tab
 PID, 525
 Grafcet
 associated actions, 362
 Examples, 357
 Instructions, 356
 preprocessing, 359
 sequential processing, 360
 Grafcet methods, 68
 Graphic elements
 Ladder diagrams, 330

I

I/O
 Addressing, 40
 Increment, 418
 Index overflow, 49
 Initialization of objects, 75
 Input tab
 PID, 528

Instructions
 AND, 378
 Arithmetic, 418
 Comparison, 416
 Conversion, 425
 JMP, 432
 Load, 374
 logic, 422
 NOT, 384
 RET, 433
 SR, 433
 XOR, 382
 instructions
 END, 429
 NOP, 431
 INT_TO_REAL, 575
 Integral action, 560
 IP address, 160
 BootP, 161
 Default IP address, 161

J

JMP, 432
 Jump Instructions, 432

L

Labeling
 Indexed, 48
 Ladder diagrams
 blocks, 328
 graphic elements, 330
 introduction, 324
 OPEN and SHORT, 333
 programming principles, 326
 Ladder List Rung, 339
 Ladder program
 reversing to List, 337
 Ladder rungs, 325
 LAN ACT, 178
 LAN ST, 179
 LD, 374
 LDF, 371, 374
 LDN, 374
 LDR, 370, 374

- Life guarding, 248
- Life time, 248
- LIFO
 - introduction, 443
 - operation, 444
- Link elements
 - graphic elements, 330
- List instructions, 347
- List Language
 - overview, 344
- List Line Comments, 340
- LKUP, 589
- LN, 568
- LOG, 568
- logic instructions, 422

M

- MAC address, 161
- Marked IP, 169
- MAX_ARR, 584
- MEAN, 594
- Memory
 - 32K cartridge, 56
 - 64K cartridge, 58
 - Structure, 52
 - without cartridge, 54
- Memory bits, 27
- Memory words, 29
- MIN_ARR, 584
- Modbus
 - Communication, 84
 - Communications, 126
 - Configuring the port, 130
 - Hardware configuration, 127
 - master, 84
 - Slave, 84
 - Software configuration, 130
 - Standard requests, 144
 - TCP Client/Server, 152
 - TCP Modbus messaging, 180
- Modbus Link
 - Example 1, 138
 - Example 2, 141
- Modbus TCP/IP
 - Remote devices, 173

- Mode
 - Operational, 244
 - pre-operational, 244
- MPP, 352
- MPS, 352
- MRD, 352
- Multiply, 418

N

- Network
 - Addressing, 42
- Node guarding, 248
- Non-reversible programming, 440
- NOP, 431
- NOP Instruction, 431
- NOT instruction, 384
- Numerical instructions
 - Assignment, 411
 - shift, 423
- Numerical processing
 - Overview, 410

O

- Object tables, 45
- Object validation, 26
- Objects
 - Bit objects, 27
 - Double word, 32
 - Floating point, 32
 - Function blocks, 43
 - Structured, 45
 - words, 29
- OCCUR_ARR, 585
- OPEN, 333
- Open loop adjustment, 558
- Operands, 346
- Operate blocks, 329
 - graphic element, 332
- Operating modes, 68

Operator Display
 Controller ID and states, 309
 Overview, 306
 Real-Time correction, 319
 Serial port settings, 317
 System objects and variables, 311
 Time of day clock, 318
 OR Instruction, 380
 OUT_BLK, 338
 Output tab
 PID, 537
 Overflow
 Index, 49
 overflow, 420
 Overview
 PID, 517

P

Parameters, 391
 Parentheses
 modifiers, 351
 nesting, 351
 using in programs, 350
 Physical layer, 239
 CAN bus line, 239
 PID
 Animation tab, 541
 AT tab, 532
 Configuration, 524
 Debugging, 540
 General tab, 525
 Input tab, 528
 Output tab, 537
 Overview, 517
 PID tab, 530
 Trace tab, 543
 PID characteristics, 521
 PID tab
 PID, 530
 Pin outs
 Communications cable female
 connector, 88
 Communications cable male connector,
 88
 Polarization (external, 129

Potentiometer, 186
 Power cut, 69
 Power restoration, 69
 Programming
 documenting your program, 340
 Programming advice, 334
 Programming grid, 326
 Programming languages
 overview, 21
 Programming Principles, 440
 Proportional action, 559
 Protocol
 Modbus TCP/IP, 85
 Protocols, 84
 Pulse generation, 451
 Pulse width modulation, 448

Q

Queue, 443

R

RAD_TO_DEG, 574
 REAL_TO_DINT, 575
 REAL_TO_INT, 575
 Real-Time correction factor, 319
 Receiving messages, 476
 Registers
 FIFO, 445
 LIFO, 444
 programming and configuring, 446
 Remainder, 418
 Remote Link
 Communications, 104
 Example, 112
 Hardware configuration, 105
 Master controller configuration, 107
 Remote controller configuration, 107
 Remote controller scan synchronization,
 107
 Remote I/O data access, 109
 Software configuration, 106
 Remote link
 Communication, 84
 RET, 433

Reversibility
 guidelines, 338
 introduction, 337
Reversible programming, 440
ROL_ARR, 586
ROR_ARR, 586
RS-485 EIA line), 129
RTC correction, 481
Run/Stop bit, 70
Rung Header, 327
 comments, 341
Rungs
 unconditional, 339

S

Scan time, 67
Scanning
 Cyclic, 62
 Periodic, 64
Shift bit register, 404
Shift instructions, 423
SHORT, 333
SIN, 572
Single/double word conversion instructions,
 427
Software watchdog, 67
SORT_ARR, 588
SQRT, 568
Square root, 418
SR, 433
Stack, 443
Stack instructions, 352
Step counter, 406
Subnet mask, 160
Subroutine instructions, 433
Subtract, 418
SUM_ARR, 579
Symbolizing, 50
System bits, 596
System words, 604

T

TAN, 572
Task cycle, 67

TCP Client/Server, 152
TCP/IP
 Protocol, 85
TCP/IP setup, 165
Test Zone, 326
Time out (Ethernet), 171
Timers, 391
 introduction, 390
 programming and configuring, 395
 time base of 1 ms, 396
 TOF type, 392
 TON type, 393
 TP type, 394
TOF timer, 392
TON timer, 393
TP type timer, 394
Trace tab
 PID, 543
Transmitting messages, 476
TRUNC, 568
TwidoSoft
 Introduction, 20

U

Unconditional rungs, 339
Unit ID, 174

V

Very fast counters function block (%VFC),
 462

W

Warm restart, 71
Word Objects, 438
Word objects
 Addressing, 37
 Overview, 29

X

XOR, 382