

PLC Programming

First Edition
Course book of Series of
Advanced Mechatronics Systems

edited by Géza HUSI, Péter SZEMES, István BARTHA

Debrecen (HU)
2012.

First edition: **PLC Programming Course book**,
Géza HUSI, Péter SZEMES, István BARTHA, 2012.

Although great care has been taken to provide accurate and current information, neither the author(s) nor the publisher, nor anyone else associated with this publication, shall be liable for any loss, damage, or liability directly or indirectly caused or alleged to be caused by this book. The material contained herein is not intended to provide specific advice or recommendations for any specific situation.

Trademark notice: Product or corporate names may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe.

HU ISSN 2063-2657

HU ISBN 978-963-473-518-2

Copyright© 2012, Géza HUSI, Péter SZEMES, István BARTHA

Reviewers:

Technical-scientific: Prof.eng. Helga SILAGHI Ph.D, Ass.Prof.eng. Eugen Ioan Gergely PhD
University of Oradea

Language: Éva DUDÁS University of Debrecen, Faculty of Engineering , XXXXX YYYYY CEZE KFT.

Publisher: Dr. habil Edit Szűcs PhD, dean of Faculty of Engineering, University of Debrecen, Debrecen, Hungary

Publisher's Note

The publisher has gone to great lengths to ensure the quality of this reprint but points out that some imperfections in the original may be apparent.

Two countries, one goal, joint success!



Hungary-Romania
Cross-Border Co-operation
Programme 2007-2013

www.hungary-romania-cbc.eu
www.huro-cbc.eu

European Union

European Regional Development Fund



The content of this book does not necessarily represent the official position of the European Union.



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

Advanced Mechatronics Systems

*A Series of course book and laboratory handbook
Debrecen (HU)*

Editors

GÉZA HUSI, Ph.D.

Associate Professor

Head of Electrical Engineering and Mechatronics Department
University of Debrecen, Faculty of Engineering
Debrecen, Hungary

RADU-CATALIN TARCA Ph.D.,

Professor

Head of Mechatronics Department
University of Oradea, Managerial and Technological Faculty
Oradea, Romania

Series Introduction

Worldwide interest in Mechatronics and its associated activities continue to grow day by day. The multidisciplinary field of mechatronics brings together mechanical engineering, electrical and electronic engineering, control engineering, and computer science in a synergistic manner.

In the latest period, major developments were noticed in this field showing that the mechatronics has advanced rapidly and gained maturity, through the development of an increasing number of degree programs, extensive research activities, product and system developments, and an increasingly broad range of industrial applications.

Many textbooks have been developed in the field of Mechatronics and this series of books also lines up with the current trends.

The appearance of this series of books was made possible as a result of HURO MECHA 0901/179/2.3.1 project implementation, funded by European Regional Development Fund.

These books fully address both the theoretical and practical aspects of the multidisciplinary field of mechatronics and fit the needs in knowledge for students enrolled to MSc program in the field of mechatronics, implemented at both Oradea and Debrecen Universities.

The purpose of these courses in mechatronics is to provide a focused interdisciplinary experience for graduate students in the field of mechanics, electrical and computer sciences.

Knowledge will be provided in the fields of Advanced Mechatronics, Materials and Machine Parts for Mechatronics, Electrical Actuators, CAD for Mechatronics, Modeling and Simulation of Mechatronics Systems, PLC Programming, Mechatronics Control Systems, Robot and CNC Programming, Mechanical Design of a Mechatronic System, Management of Complex Production Systems, Software Reliability Engineering, Product Lifecycle Management, Finite Element Analysis, Diagnosis and Maintenance of Mechatronics Systems.

This series presents books that draw on expertise from both the academic world and the application domains, and will be useful not only as academically recommended course texts but also as handbooks for practitioners in many application domains.

GÉZA HUSI, RADU-CĂTĂLIN ȚARCĂ, editors

*Advanced Mechatronics Systems
A Series of course book and laboratory handbook
Debrecen (HU)*

Editors: GÉZA HUSI, RADU-CATALIN TARCA,

1. Radu Cătălin ȚARCĂ: ADVANCED MECHATRONICS - course book
2. Radu Cătălin ȚARCĂ: ADVANCED MECHATRONICS - laboratory handbook
3. Ioan Constantin ȚARCĂ: MATERIALS AND MACHINE PARTS FOR MECHATRONICS - course book
4. Ioan Constantin ȚARCĂ: MATERIALS AND MACHINE PARTS FOR MECHATRONICS - laboratory handbook
5. János TÓTH: ELECTRICAL ACTUATORS - course book
6. János TÓTH: ELECTRICAL ACTUATORS - laboratory handbook
7. Mircea Teodor POP: CAD FOR MECHATRONICS - course book
8. Mircea Teodor POP: CAD FOR MECHATRONICS - laboratory handbook
9. Florin Sandu BLAGA - MODELING AND SIMULATION OF MECHATRONICS SYSTEMS - course book
10. Florin Sandu BLAGA - MODELING AND SIMULATION OF MECHATRONICS SYSTEMS - laboratory handbook
11. Géza HUSI, Péter SZEMES, István BARTHA: PLC PROGRAMMING - course book
12. Géza HUSI, Péter SZEMES, István BARTHA: PLC PROGRAMMING - laboratory handbook
13. Géza HUSI: MECHATRONICS CONTROL SYSTEMS – course book
14. Géza HUSI: MECHATRONICS CONTROL SYSTEMS – laboratory handbook
15. Tiberiu VESSELENYI: ROBOT AND CNC PROGRAMMING– course book
16. Tiberiu VESSELENYI: ROBOT AND CNC PROGRAMMING– laboratory handbook
17. Edit SZÜCS: MANAGEMENT OF COMPLEX PRODUCTION SYSTEMS - course book
18. Zsolt TIBA, Géza HUSI: MECHANICAL DESIGN OF A MECHATRONICS SYSTEM – laboratory handbook
19. Florin VLĂDICESCU POPENȚIU: SOFTWARE RELIABILITY ENGINEERING- course book
20. Alexandru Viorel PELE: PRODUCT LIFECYCLE MANAGEMENT- course book
21. Alexandru Viorel PELE: PRODUCT LIFECYCLE MANAGEMENT- laboratory handbook
22. Flavius ARDELEAN: FINITE ELEMENT ANALYSIS – course book
23. Flavius ARDELEAN: FINITE ELEMENT ANALYSIS - laboratory handbook
24. Sorin Marcel PATER: DIAGNOSIS AND MAINTENANCE OF MECHATRONICS SYSTEMS – course book
25. Sorin Marcel PATER: DIAGNOSIS AND MAINTENANCE OF MECHATRONICS SYSTEMS- laboratory handbook

Additional Volumes under Preparation

PLC Programming

Course book
First Edition

Géza HUSI, Péter SZEMES, István BARTHA
*Electrical Engineering and Mechatronics Department
University of Debrecen, Faculty of Engineering
Debrecen, Hungary.*

This book is based on the "Twido programmable controllers Programming Guide" Version 1.0, 06/2006 published by Schneider Electric.

This book would not have been possible unless the help of Schneider Electric Hungária Villamossági Zrt., through the Schneider Electric Knowledge Center, operating as a joint cooperation effort between the company and the department. We would like to thank to Schneider Electric to they continuous support.

Debrecen
2012.

TABLE OF CONTENTS

Table of Contents.....	7
1 Introduction to TwidoSuite.....	15
At a Glance.....	15
1.1.1 Introduction to TwidoSuite.....	15
1.1.2 Introduction to Twido Languages.....	16
2 Twido Language Objects.....	19
At a Glance.....	19
2.1.1 Language Object Validation.....	19
2.1.2 Bit Objects.....	20
2.1.3 Word Objects.....	21
2.1.4 Floating point and double word objects.....	23
2.1.5 Addressing Bit Objects.....	26
2.1.6 Addressing Word Objects	27
2.1.7 Addressing floating objects	28
2.1.8 Addressing double word objects	29
2.1.9 Addressing Inputs/Outputs	30
2.1.10 Network Addressing	31
2.1.11 Function Block Objects	32
2.1.12 Structured Objects.....	33
2.1.13 Indexed objects.....	36
2.1.14 Symbolizing Objects.....	37
3 User Memory.....	38
At a Glance.....	38
3.1.1 User Memory Structure.....	38
3.1.2 Backup and Restore without Backup Cartridge or Extended Memory	40
3.1.3 Backup and Restore with a 32K Backup Cartridge	42
3.1.4 Using the 64K Extended Memory Cartridge	43
4 Event task management	46
At a Glance.....	46
4.1.1 Overview of event tasks	46
4.1.2 Description of different event sources.....	47
4.1.3 Event management.....	48

5 Communications.....	50
At a Glance.....	50
5.1.1 Presentation of the different types of communication.....	50
5.1.2 TwidoSuite to Controller communications.....	51
5.1.3 Communication between TwidoSuite and a Modem.....	54
5.1.4 Remote Link Communications.....	61
5.1.5 ASCII Communications.....	69
5.1.6 Modbus Communications.....	77
5.1.7 Standard Modbus Requests.....	89
5.1.8 Transparent Ready Implementation Class (Twido Serial A05, Ethernet A15)	92
6 Built-In Analog Functions.....	94
At a Glance.....	94
6.1.1 Analog potentiometer	94
6.1.2 Analog Channel.....	95
7 Managing Analog Modules.....	97
At a Glance.....	97
7.1.1 Analog Module Overview	97
7.1.2 Addressing Analog Inputs and Outputs	98
7.1.3 Configuring Analog Inputs and Outputs	99
7.1.4 Analog Module Status Information	103
7.1.5 Example of Using Analog Modules	104
8 Installing the AS-Interface V2 bus.....	106
At a Glance.....	106
8.1.1 Presentation of the AS-Interface V2 bus	106
8.1.2 General functional description	107
8.1.3 Software set up principles	109
8.1.4 Description of the configuration screen for the AS-Interface bus	110
8.1.5 Configuration of the AS-Interface bus	111
8.1.6 Description of the AS-Interface Window in Online Mode.....	116
8.1.7 Modification of Slave Address	118
8.1.8 Updating the AS-Interface bus configuration in online mode.....	119
8.1.9 Automatic addressing of an AS-Interface V2 slave.....	122
8.1.10 How to insert a slave device into an existing AS-Interface V2 configuration.....	123
8.1.11 Automatic replacement of a faulty AS-Interface V2 slave	123
8.1.12 Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus	124

8.1.13 Programming and diagnostics for the AS-Interface V2 bus.....	125
8.1.14 AS-Interface V2 bus interface module operating mode:.....	129
9 Installing and Configuring the CANopen Fieldbus	131
At a Glance.....	131
9.1 CANopen Fieldbus Overview	131
9.1.1 At a Glance.....	131
9.1.2 CANopen Knowledge Base.....	132
9.1.3 About CANopen	133
9.1.4 CANOpen Boot-Up.....	135
9.1.5 Process Data Object (PDO) Transmission	137
9.1.6 Access to Data by Explicit Exchanges (SDO)	138
9.1.7 "Node Guarding" and "Life Guarding"	139
9.1.8 Internal Bus Management	141
9.2 Implementing the CANopen Bus	141
Overview.....	141
9.2.1 Overview.....	142
9.2.2 Hardware Setup.....	142
9.2.3 Configuration Methodology	143
9.2.4 Declaration of CANopen Master.....	144
9.2.5 Network CANopen Slave Declaration	144
9.2.6 CANopen Objects Mapping	148
9.2.7 CANopen Objects Linking	151
9.2.8 CANopen Objects Symbolization	152
9.2.9 Addressing PDOs of the CANopen master.....	153
9.2.10 Programming and diagnostics for the CANopen fieldbus	154
10 Configuring the TwidoPort Ethernet Gateway	159
At a Glance.....	159
10.1 Normal Configuration and Connection of TwidoPort.....	159
10.1.1 At a Glance.....	159
10.1.2 Normal Configuration with TwidoSuite	159
10.1.3 BootP Configuration	163
10.2 TwidoPort's Telnet Configuration.....	164
At a Glance.....	164
10.2.1 Introducing Telnet Configuration	165
10.2.2 Telnet Main Menu	165

10.2.3 IP/Ethernet Settings.....	166
10.2.4 Serial Parameter Configuration	167
10.2.5 Configuring the Gateway	167
10.2.6 Security Configuration	168
10.2.7 Ethernet Statistics.....	169
10.2.8 Serial Statistics.....	170
10.2.9 Saving the Configuration	170
10.2.10 Restoring Default Settings	171
10.2.11 Upgrading the TwidoPort Firmware	171
10.2.12 Forget Your Password and/or IP Configuration?	173
10.3 Communication Features	173
At a Glance.....	174
10.3.1 Ethernet Features	174
10.3.2 Modbus/TCP Communications Protocol	174
10.3.3 Locally Supported Modbus Function Codes	175
11 Operator Display Operation	176
At a Glance.....	176
11.1.1 Operator Display.....	176
11.1.2 Controller Identification and State Information.....	178
11.1.3 System Objects and Variables	180
11.1.4 Serial Port Settings.....	185
11.1.5 Time of Day Clock	186
11.1.6 Real-Time Correction Factor.....	187
12. PLC Programming Languages.....	188
At a Glance.....	188
12.1 Ladder Language.....	188
At a Glance.....	188
12.1.1 Introduction to Ladder Diagrams	189
12.1.2 Programming Principles for Ladder Diagrams	190
12.1.3 Ladder Diagram Blocks	192
12.1.4 Ladder Language Graphic Elements	194
12.1.5 Special Ladder Instructions OPEN and SHORT.....	196
12.1.6 Programming Advice	197
12.1.7 Ladder/List Reversibility	199
12.1.8 Guidelines for Ladder/List Reversibility.....	200

12.1.9 Program Documentation	201
12.2 Instruction List Language	203
At a Glance.....	203
12.2.1 Overview of List Programs.....	204
12.2.2 Operation of List Instructions	205
12.2.3 List Language Instructions	206
12.2.4 Using Parentheses	208
12.2.4 Stack Instructions (MPS, MRD, MPP).....	209
12.3 Grafset	211
At a Glance.....	211
12.3.1 Description of Grafset Instructions	211
12.3.2 Description of Grafset Program Structure.....	213
12.3.3 Actions Associated with Grafset Steps	215
13. Basic Instructions.....	217
13.1 Boolean Processing.....	217
At a Glance.....	217
13.1.1 Boolean Instructions.....	217
13.1.2 Understanding the Format for dcribing boolean instructions	219
13.1.3 Load Instructions (LD, LDN, LDR, LDF)	220
13.1.4 Assigment Instructions (ST, STN, R, S)	221
13.1.5 Logical AND Instructions (AND, ANDN, ANDR, ANDF).....	222
13.1.6 Logical OR Instructions (OR, ORN, ORR, ORF).....	223
13.1.7 Exclusive OR Instructions (XOR, XORN, XORR, XORF).....	224
13.1.8 NOT Instruction (N).....	225
13.2 Basic Function blocks.....	226
13.2.1 At a Glance.....	226
13.2.2 Basic Function Blocks.....	227
13.2.3 Standard function blocks programming principles.....	228
13.2.4 Timer Function Block (%Tmi).....	230
13.2.5 TOF Type of Timer	231
13.2.6 TON Type of Timer.....	232
13.2.7 TP Type of Timer	232
13.2.8 Programming and Configuring Timers.....	233
13.2.9 Up/Down Counter Function Block (%Ci).....	235
13.2.10 Programming and Configuring Counters	237

13.2.11 Shift Bit Register Function Block (%SBRi)	238
13.2.12 Step Counter Function Block (%Sci).....	240
13.3 Numerical Processing	241
13.3.1 At a Glance.....	241
13.3.2 Introduction to Numerical Instructions.....	242
13.3.3 Assignment Instructions	242
13.3.4 Comparison Instructions.....	246
13.3.5 Arithmetic Instructions on Integers.....	247
13.3.6 Logic Instructions.....	249
13.3.7 Shift Instructions.....	250
13.3.8 Conversion Instructions.....	251
13.3.9 Single/double word conversion instructions.....	252
13.4 Program Instructions	253
13.4.1 At a Glance.....	253
13.4.2 END Instructions	253
13.4.3 NOP Instruction	254
13.4.4 Jump Instructions	255
13.4.5 Subroutine Instructions	256
14. Advanced Instructions	258
At a Glance.....	258
14.1 Advanced Function Blocks	258
14.1.1 At a Glance.....	258
14.1.2 Bit and Word Objects Associated with Advanced Function Blocks.....	259
14.1.3 Programming Principles for Advanced Function Blocks.....	260
14.1.4 LIFO/FIFO Register Function Block (%Ri)	262
14.1.5 LIFO Operation.....	262
14.1.6 FIFO,operation	263
14.1.7 Programming and Configuring Registers.....	263
14.1.8 Pulse Width Modulation Function Block (%PWM)	265
14.1.9 Pulse Generator Output Function Block (%PLS)	267
14.1.10 Drum Controller Function Block (%DR)	268
14.1.11 Drum Controller Function Block %DRI Operation	269
14.1.12 Programming and Configuring Drum Controllers.....	270
14.1.13 Fast Counter Function Block (%FC).....	271
14.1.14 Very Fast Counter Function Block (%VFC)	273

14.1.15 Transmitting/Receiving Messages - the Exchange Instruction (EXCH)	281
14.1.16 Exchange Control Function Block (%MSGx).....	282
14.2 Clock Functions	285
14.2.1 At a Glance.....	285
14.2.2 Clock Functions.....	285
14.2.3 Schedule Blocks	286
14.2.4 Time/Date Stamping.....	287
14.2.5 Setting the Date and Time	288
14.3 Twido PID Quick Start Guide.....	291
14.3.1 At a Glance.....	291
14.3.2 Purpose of Document.....	291
14.3.3 Step 1 - Configuration of Analog Channels Used for Control	292
14.3.4 Step 2 - Prerequisites for PID Configuration.....	294
14.3.5 Step3 – Configuring the PID.....	295
14.3.6 Step 4 - Initialization of Control Set-Up.....	298
14.3.7 Step 5 - Control Set-Up AT + PID.....	301
14.3.8 Step 6 - Debugging Adjustments	304
14.4 PID Function.....	305
14.4.1 At a Glance.....	305
14.4.2 Overview.....	306
14.4.3 Principal of the Regulation Loop	307
14.4.4 Development Methodology of a Regulation Application.....	308
14.4.5 Compatibilities and Performances	309
14.4.6 Detailed characteristics of the PID function	310
14.4.7 How to access the PID configuration.....	312
14.4.8 PID Screen Elements of PID function.....	313
14.4.9 General tab of PID function	316
14.4.10 Input tab of the PID	318
14.4.11 PID tab of PID function	320
14.4.12 AT tab of PID function.....	321
14.4.13 Output tab of the PID	324
14.4.14 How to access PID debugging.....	326
14.4.15 Animation tab of PID function	327
14.4.16 Trace screen of PID function.....	328
14.4.17 PID States and Errors Codes	329

14.4.18 PID Tuning With Auto-Tuning (AT)	331
14.4.19 PID parameter adjustment method	336
14.4.20 Role and influence of PID parameters.....	338
14.4.21 Appendix 1: PID Theory Fundamentals	340
14.4.22 Appendix 2: First-Order With Time Delay Model	341
14.5 Floating point instructions.....	343
14.5.1 At a Glance.....	343
14.5.2 Arithmetic instructions on floating point	343
14.5.3 Trigonometric Instructions	345
14.5.4 Conversion instructions	346
14.5.5 Integer Conversion Instructions <-> Floating	347
14.6 Instructions on Object Tables	349
14.6.1 At a Glance.....	350
14.6.2 Table summing functions	350
14.6.3 Table comparison functions	351
14.6.4 Table search functions.....	352
14.6.5 Table search functions for maxi and mini values	354
14.6.6 Number of occurrences of a value in a table.....	354
14.6.7 Table rotate shift function.....	355
14.6.8 Table sort function.....	356
14.6.9 Floating point table interpolation function	357
14.6.10 Mean function of the values of a floating point table.....	360
15 System Bits and System Words	362
At a Glance.....	362
15.1.1 System Bits (%S).....	362
15.1.2 System Words (%SW)	369

1 INTRODUCTION TO TWIDOSUITE

AT A GLANCE

SUBJECT OF THIS CHAPTER

This chapter provides a brief introduction to TwidoSuite, 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 TwidoSuite 16

Introduction to Twido Languages 17

1.1.1 INTRODUCTION TO TWIDOSUITE

1.1.1.1 INTRODUCTION

TwidoSuite is a full-featured, graphical development environment for creating, configuring, and maintaining automation applications for Telemecanique Twido programmable controllers. TwidoSuite 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.

1.1.1.2 TWIDOSUITE

TwidoSuite is a 32-bit Windows-based program for a personal computer (PC) running Microsoft Windows 2000/XP Professional operating systems.

The main software features of TwidoSuite:

- ▲ Project-oriented, intuitive user-interface
- ▲ Menu-free software design. All tasks and functions of a selected project step show at all times.
- ▲ Programming and configuration support
- ▲ Communication with controller
- ▲ Task-level, first-hand help providing relevant links to the online help

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



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

1.1.1.3 MINIMUM CONFIGURATION

The minimum configuration for using TwidoSuite is:

- ▲ PC-compatible computer with
- ▲ processor Pentium 466 MHz or higher recommended,
- ▲ 128 MB of RAM or higher recommended,
- ▲ 100 MB of hard disk space.
- ▲ Operating system : Windows 2000 or Windows XP:
- ▲ Avoid patch 834707-SP1 (corrected by patch 890175) and patch 896358 which cause display problems with Online Help.
- ▲ Service Pack 2 or higher recommended. Available for download from www.microsoft.com web site.

1.1.2 INTRODUCTION TO TWIDO LANGUAGES

1.1.2.1 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.

1.1.2.2 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.

1.1.2.3 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.

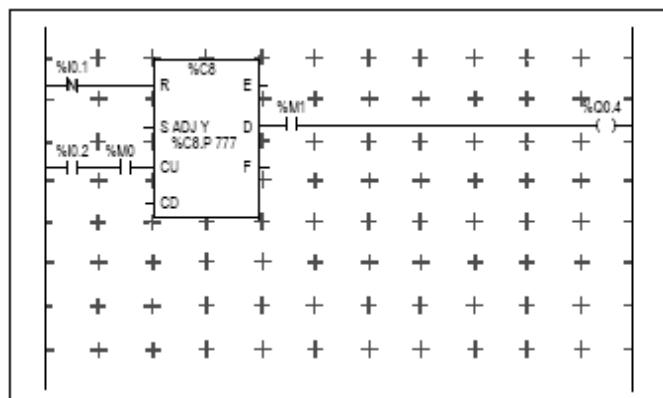
```

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

```

1.1.2.4 LADDER DIAGRAMS

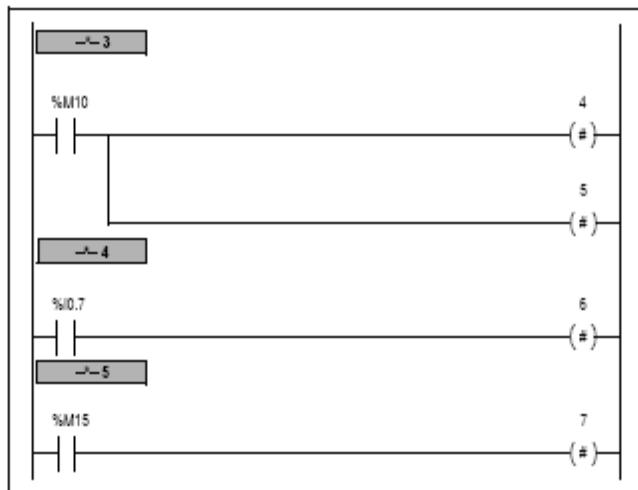
Ladder diagrams are similar to relay logic diagrams that represent relay control circuits. Graphic elements such as coils, contacts, and blocks represent instructions. The following is an example of a Ladder diagram.



1.1.2.5 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	.A.	3
1	LD	%M10
2	#	4
3	#	5
4	.A.	4
5	LD	%I0.7
6	#	6
7	.A.	5
8	LD	%M15
9	#	7
10	..	



2 TWIDO LANGUAGE OBJECTS

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 20

Bit Objects 21

Word Objects 22

Floating point and double word objects 24

Addressing Bit Objects 27

Addressing Word Objects 28

Addressing floating objects 29

Addressing double word objects 30

Addressing Inputs/Outputs 31

Network Addressing 32

Function Block Objects 33

Structured Objects 34

Indexed objects 37

Symbolizing Objects 38

2.1.1 LANGUAGE OBJECT VALIDATION

2.1.1.1 INTRODUCTION

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



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

2.1.1.2 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.

2.1.2 BIT OBJECTS

2.1.2.1 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

2.1.2.2 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, %Gi.P, and so on.	Note (4)	No (3)
Reversible function blocks	Function blocks programmed using reversible programming instructions BLK, OUT_BLK, and END_BLK.	E, O, F, Q, TH0, TH1	Note (4)	No

Type	Description	Address or value	Maximum number	Write access (1)
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.	%Xi	62 TWDLC*A10DRF, TWDLC*A16 DRF 96 TWDLC*A24DRF, TWDLC*A40DRF 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. 191.)

2.1.3 WORD OBJECTS

2.1.3.1 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 (%MW_i) (memory words)
- ▲ Constant words (%KW_i)
- ▲ I/O exchange words (%IW_i, %QW_i%)
- ▲ AS-Interface analog I/O words (IW_{Ai}, %QWA_i)
- ▲ System words (%SW_i)
- ▲ Function blocks (configuration and/or runtime data)

2.1.3.2 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
																Bit state
0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	1	Bit value
+ 16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	0	

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

2.1.3.3 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.	%MWi	3000	Yes
Constants	Store constants or alphanumeric messages. Their content can only be written or modified by using TwidoSuite during configuration.	%KWi	256	Yes, only by using TwidoSuite
System	These 16-bit words have several functions: <ul style="list-style-type: none">• 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, %DI.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	%IWj.i:Xk	Note (2)	No
	Output	%QWj.i: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	%INWj.i:Xk	Note (2)	No
	Network Output	%QNWj.i: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. 191.)

2.1.4 FLOATING POINT AND DOUBLE WORD OBJECTS

2.1.4.1 INTRODUCTION

TwidoSuite 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.

2.1.4.2 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:

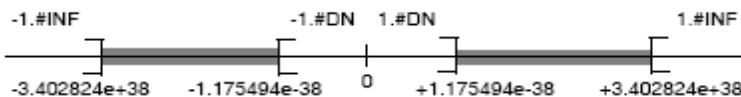
Bit 31	Bits {30...23}	Bits {22...0}
S	Exponent	Fractional part

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

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

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"

2.1.4.3 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 (where: EXPT(%MF,%MW) x^y = %MW^%MF)	EXPT(y, x)	x < 0	y.In(x) > 88
Base 10 logarithm	LOG(x)	x <= 0	x > 2.4E38
Natural logarithm	LN(x)	x <= 0	x > 1.65E38
Natural exponential	EXP(x)	x < 0	x > 88.0

2.1.4.4 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*24DRF	Yes	No
TWDLCA*16DRF	Yes	No
TWDLCA*10DRF	No	No

2.1.4.5 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 reusage purposes.

2.1.4.6 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 (double word) or decimal (floating point) numbers with identical format to 32 bit objects.	-	[-]	No	-
Internal floating point	Objects used to store values during operation in data memory.	%MF _i	1500	Yes (ODM/T)	%MF _i [index]
Internal double word		%MD _i	1500	Yes (ODM/T)	%MD _i [index]
Floating constant value	Used to store constants.	%KF _i	128	Yes, (T)	%KF _i [index]
Double constant		%KD _i	128	Yes, (T)	%KD _i [index]
Note:	1. ODM: Write access using the Operator Display Module (see <i>Operator Display Operation</i> , p. 271) 2. T: Write access using TwidoSuite				

2.1.4.7 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		...
	%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		...
	%KF _i / %KD _i	%KW _i
%KF _i +1 / %KD _i +1		%KW _i +1

Example:

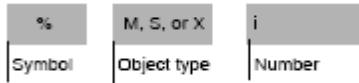
%MF0 corresponds to %MW0 and %MW1. %KF543 corresponds to %KW543 and %KW544.

2.1.5 ADDRESSING BIT OBJECTS



2.1.5.1 SYNTAX

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



2.1.5.2 DESCRIPTION

The following table describes the elements in the addressing format.

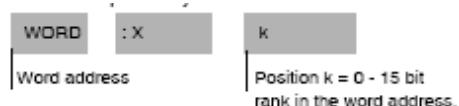
Group	Item	Description
Symbol	%	The percent symbol always precedes a software variable.
Type of object	M	Internal bits store intermediary values while a program is running.
	S	System bits provide status and control information for the controller.
	X	Step bits provide status of step activities.
Number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

2.1.5.3 BIT OBJECTS EXTRACTED FROM WORDS

TwidoSuite 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:



Examples:

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

2.1.6 ADDRESSING WORD OBJECTS

2.1.6.1 INTRODUCTION

Addressing word objects, except for input/output addressing (see Addressing Inputs/Outputs, p. 41) and function blocks (see Function Block Objects, p. 44), follows the format described below.

2.1.6.2 SYNTAX

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



2.1.6.3 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 TwidoSuite.
	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

2.1.7 ADDRESSING FLOATING OBJECTS

2.1.7.1 INTRODUCTION

Addressing floating objects, except for input/output addressing (see Addressing Inputs/Outputs, p. 41) and function blocks (see Function Block Objects, p. 44), follows the format described below.

2.1.7.2 SYNTAX

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



2.1.7.3 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 TwidoSuite.
Syntax	D	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

2.1.8 ADDRESSING DOUBLE WORD OBJECTS

2.1.8.1 INTRODUCTION

Addressing double word objects, except for input/output addressing (see Addressing Inputs/Outputs, p. 41) and function blocks (see Function Block Objects, p. 44), follows the format described below.

2.1.8.2 SYNTAX

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

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

2.1.8.3 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 TwidoSuite.
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

2.1.9 ADDRESSING INPUTS/OUTPUTS

2.1.9.1 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. 191.
- ▲ For TWDNCO1M, see Addressing PDOs of the CANopen master, p. 233.

2.1.9.2 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.

2.1.9.3 FORMAT

Use the following format to address inputs/outputs.

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

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

%	I, Q	w	x	y
Symbol	Object type	Format	Controller position	point

2.1.9.4 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.
	O	-	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.

2.1.9.5 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).
%O0.3.4	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.

2.1.10 NETWORK ADDRESSING

2.1.10.1 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. 73 for more details.

2.1.10.2 FORMAT

Use the following format for network addressing.

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

2.1.10.3 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.
	ON	-	Network output word. Data transfer from peer to master.
Format	W	-	A16-bit word.
Controller position	x	0 1 - 7	Master controller (Remote Link master). 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.

2.1.10.4 EXAMPLES

The table below shows some examples of networking addressing.

Network object	Description
%INW3.1	Network word number 1 of remote controller number 3.
%DNW0.3	Network word number 3 of the base controller.

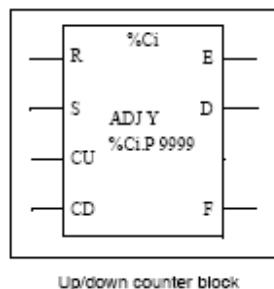
2.1.11 FUNCTION BLOCK OBJECTS

2.1.11.1 INTRODUCTION

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

2.1.11.2 EXAMPLE OF A FUNCTION BLOCK

The following illustration shows a counter function block.



2.1.11.3 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. 358).
- ▲ By specifying the block type (for example, LD %Ci.E).

Inputs can be accessed in the form of instructions.

2.1.11.4 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.

2.1.11.5 DOUBLE 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. 34 to find out if your Twido controller can accommodate double words.

2.1.11.6 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. 356.
- ▲ For Advanced Function Blocks, see Bit and Word Objects Associated with Advanced Function Blocks, p. 406.

2.1.12 STRUCTURED OBJECTS

2.1.12.1 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

2.1.12.2 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

%M8	%M9	%M10	%M11	%M12	%M13

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. 380).

2.1.12.3 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:i:L with i multiple of 8	0<L<17 and i+L≤ 128	Depending on i
Grafset Step bits	%Xi:i:L with i multiple of 8	0<L<17 and i+L≤ 95 (2)	Yes (by program)
Internal bits	%Mi:i:L with i multiple of 8	0<L<17 and i+L≤ 256 (3)	Yes

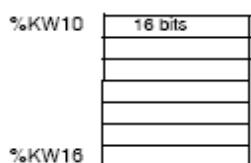
Key:

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

2.1.12.4 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



Word tables can be used with the Assignment instruction (see Assignment Instructions, p. 380).

2.1.12.5 AVAILABLE TYPES OF WORDS

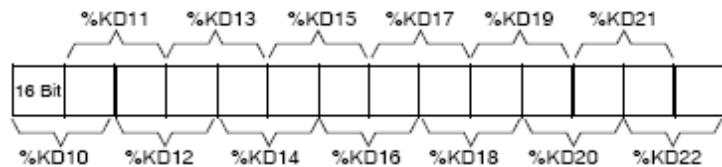
Available types of words for word tables:

Type	Address	Maximum size	Write access
Internal words	%MW <i>i</i> :L	0<L<256 and i+L< 3000	Yes
Constant words	%KW <i>i</i> :L	0<L<256 and i+L< 256	No
System Words	%SW <i>i</i> :L	0<L and i+L<128	Depending on i

2.1.12.6 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



Double word tables can be used with the Assignment instruction (see Assignment Instructions, p. 380).

2.1.12.7 AVAILABLE TYPES OF DOUBLE WORDS

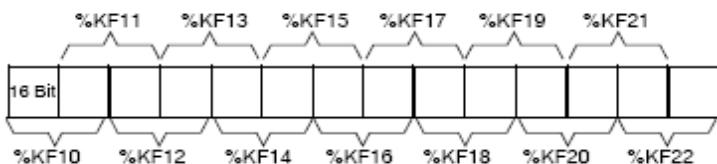
Available types of words for double word tables:

Type	Address	Maximum size	Write access
Internal words	%MD <i>i</i> :L	0<L<256 and i+L< 3000	Yes
Constant words	%KD <i>i</i> :L	0<L and i+L<256	No

2.1.12.8 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



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

2.1.12.9 TYPES OF FLOATING WORDS AVAILABLE

Available types of words for floating word tables:

Type	Address	Maximum size	Write access
Internal words	%MF <i>i</i> :L	0<L<256 and i+L< 3000	Yes
Constant words	%KF <i>i</i> :L	0<L and i+L<256	No

2.1.13 INDEXED OBJECTS

2.1.13.1 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

2.1.13.2 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.

2.1.13.3 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 %MW*i*. 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).

2.1.13.4 OBJECTS AVAILABLE FOR INDEXED ADDRESSING

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

Type	Address	Maximum size	Write access
Internal words	%MW <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <3000	Yes
Constant words	%KW <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <256	No
Internal double words	%MD <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <2999	Yes
Double constant words	%KD <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <255	No
Internal floating points	%MF <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <2999	Yes
Constant floating points	%KF <i>i</i> [%MW] <i>j</i>	0≤ <i>i</i> +%MW <i>j</i> <255	No

Indexed objects can be used with the assignment instructions (see Assignment Instructions, p. 380 for single and double words) and in comparison instructions (see Comparison Instructions, p. 384 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.

2.1.13.5 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.

2.1.14 SYMBOLIZING OBJECTS

2.1.14.1 INTRODUCTION

You can use Symbols to address TwidoSuite 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.

2.1.14.2 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.

2.1.14.3 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.

2.1.14.4 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.

3 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 39

Backup and Restore without Backup Cartridge or Extended Memory 41

Backup and Restore with a 32K Backup Cartridge 43

Using the 64K Extended Memory Cartridge 44

3.1.1 USER MEMORY STRUCTURE

3.1.1.1 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)

3.1.1.2 BIT MEMORY

The bit memory is located in the controller's built-in RAM. It contains the map of 128 bit objects.

3.1.1.3 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.

3.1.1.4 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.

- ▲ 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.

3.1.1.5 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 TwidoSuite.

3.1.1.6 MEMORY CONFIGURATIONS

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

3.1.2 BACKUP AND RESTORE WITHOUT BACKUP CARTRIDGE OR EXTENDED MEMORY

3.1.2.1 INTRODUCTION

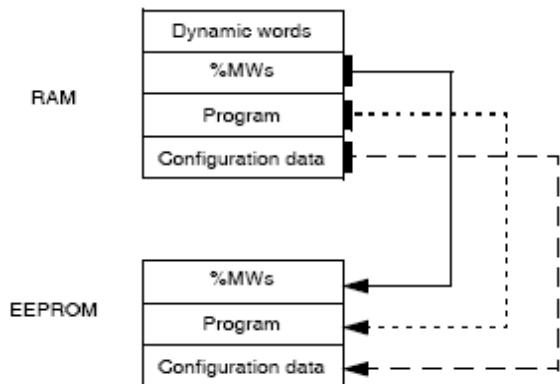
The following information details backup and restore memory functions in modular and compact controllers without a backup cartridge or extended memory plugged in.

3.1.2.2 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.

3.1.2.3 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:



3.1.2.4 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 TwidoSuite window bring down the menu under 'Controller', scroll down to 'Backup' and click on it.

3.1.2.5 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 TwidoSuite window bring down the menu under 'Controller', scroll down to 'Restore' and click on it.

3.1.2.6 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.

3.1.2.7 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

3.1.3 BACKUP AND RESTORE WITH A 32K BACKUP CARTRIDGE

3.1.3.1 INTRODUCTION

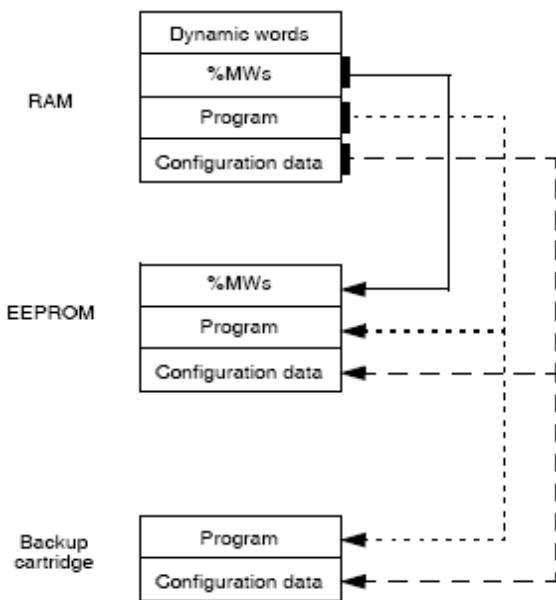
The following information details backup and restore memory functions in modular and compact controllers using a 32K backup cartridge.

3.1.3.2 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.

3.1.3.3 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:



3.1.3.4 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 TwidoSuite 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.

3.1.3.5 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.

3.1.3.6 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 %SW98:X0 to 1.

3.1.3.7 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

3.1.4 USING THE 64K EXTENDED MEMORY CARTRIDGE

3.1.4.1 INTRODUCTION

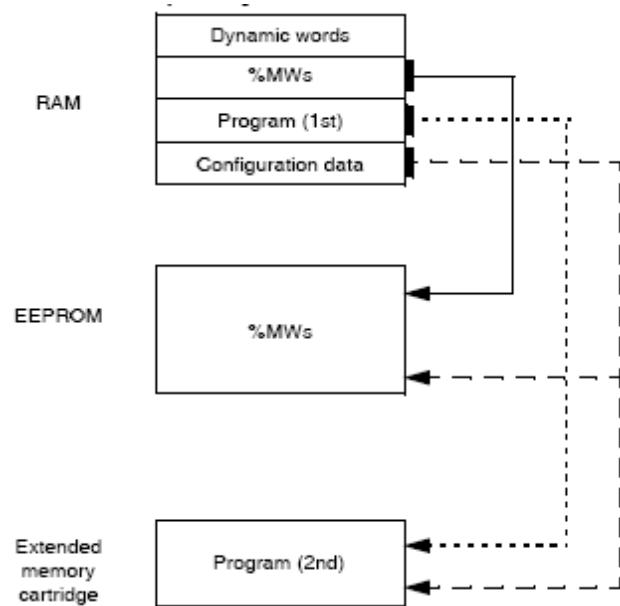
The following information details using the memory functions in modular controllers using a 64K extended memory cartridge.

3.1.4.2 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.

3.1.4.3 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 64Kextended memory cartridge from RAM:



3.1.4.4 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 TwidoSuite window enter 'TWDXCPMF64'.
2	Power down the controller.
3	Plug in the 64K extended memory cartridge.
4	Powerup the controller.

3.1.4.5 SAVE YOUR PROGRAM.

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

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

3.1.4.6 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.

3.1.4.7 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

4 EVENT TASK MANAGEMENT

AT A GLANCE

SUBJECT OF THIS CHAPTER

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

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

WHAT'S IN THIS CHAPTER?

This chapter contains the following topics:

Topic Page

Overview of event tasks 47

Description of different event sources 48

Event management 49

4.1.1 OVERVIEW OF EVENT TASKS

4.1.1.1 INTRODUCTION

The previous chapter presented periodic and cyclic 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.

4.1.1.2 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. 67),
- ▲ a section which is an 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.

4.1.2 DESCRIPTION OF DIFFERENT EVENT SOURCES

4.1.2.1 OVERVIEW OF DIFFERENT EVENT SOURCES

An event source needs to be managed by the software to make 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 TwidoSuite. 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.

4.1.2.2 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 "TwidoSuite Operation Guide" on-line help.

4.1.2.3 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 "TwidoSuite Operation Guide" on-line help.

4.1.2.4 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 "TwidoSuite Operation Guide" on-line help.

4.1.3 EVENT MANAGEMENT

4.1.3.1 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.

4.1.3.2 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.

4.1.3.3 EVENT CHECK

System bits and words are used to check the events (See System Bits and System Words, p. 563):

- ▲ %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.

5 COMMUNICATIONS

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 51

TwidoSuite to Controller communications 52

Communication between TwidoSuite and a Modem 55

Remote Link Communications 62

ASCII Communications 70

Modbus Communications 78

Standard Modbus Requests 92

Transparent Ready Implementation Class (Twido Serial A05, Ethernet A15) 93

5.1.1 PRESENTATION OF THE DIFFERENT TYPES OF COMMUNICATION

5.1.1.1 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 TwidoSuite, 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.

5.1.1.2 REMOTE LINK



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

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.

5.1.1.3 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.

5.1.1.4 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.

5.1.1.5 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.

5.1.2 TWIDOSUITE TO CONTROLLER COMMUNICATIONS

5.1.2.1 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 TwidoSuite programming software.

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

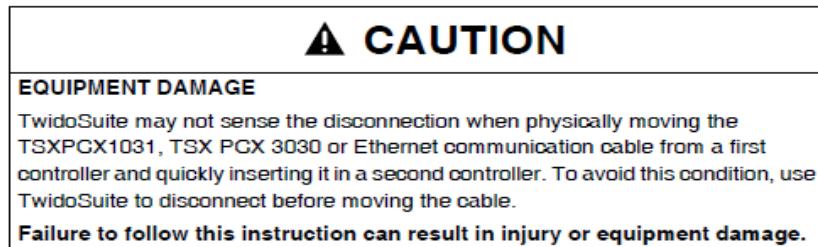
There are several ways to connect the PC to the Twido controller RS-485 Port 1:

- ▲ 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 TwidoSuite programming software.

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

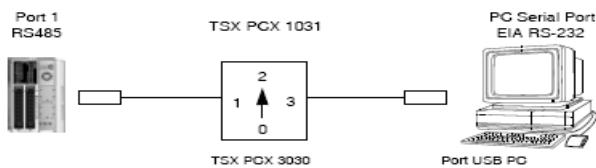
- ▲ By direct cable connection via a UTP Cat5 RJ45 Ethernet crossover cable (not recommended),
- ▲ By connection to the Ethernet network via a SFTP Cat5 RJ45 Ethernet cable available from the Schneider Electric catalog (cable reference: 490NTW000••).



5.1.2.2 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 TwidoSuite to Twido controller is location 2.

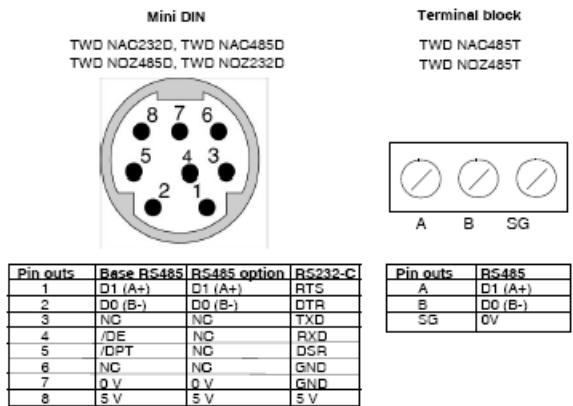
This connection is illustrated in the diagram below.



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 TwidoSuite connection. The signal is pulled up internally, informing the firmware executive that this is a TwidoSuite connection.

5.1.2.3 PIN OUTS OF MALE AND FEMALE CONNECTORS

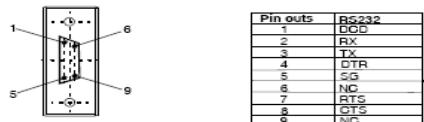
The following figure shows the pin outs of a male 8-pin miniDIN connector and of a terminal:



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

Note: There is no relation between the Twido RS485 A and B terminals and the D(A) and D(B) terminals on other equipment such as ATV drives, XBT, Premium, etc.

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

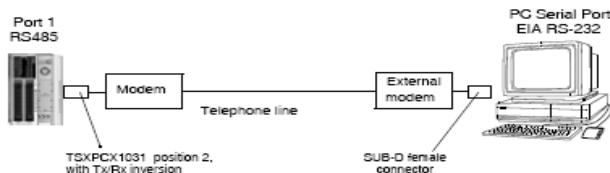


5.1.2.4 TELEPHONE LINE CONNECTION

A modem (See Communication between TwidoSuite and a Modem, p. 82) 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.



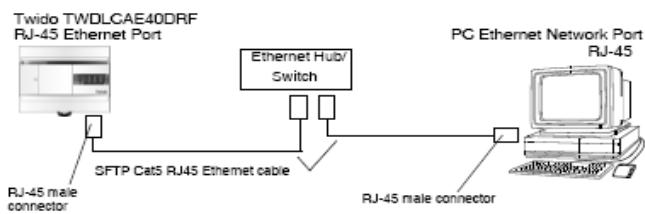
Note: Only one modem can be connected to port 1 of the controller.

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

5.1.2.5 ETHERNET NETWORK CONNECTION

Note: Although direct cable connection (using a Ethernet crossover cable) is supported between the Twido TWDLCAE40DRF and the PC running the TwidoSuite 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 TwidoSuite 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	(+)
2	TxD	(-)
1	TxD	(+)

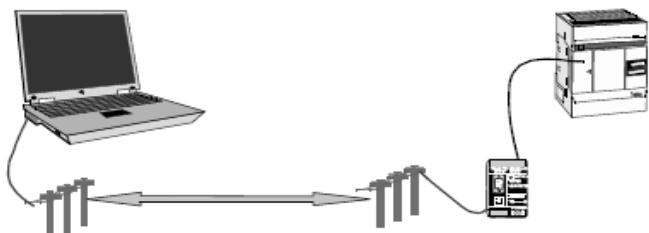
Note:

- ▲ 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.

5.1.3 COMMUNICATION BETWEEN TWIDOSUITE AND A MODEM

5.1.3.1 GENERAL

A PC executing TwidoSuite 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.



5.1.3.2 INSTALLING THE MODEM

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

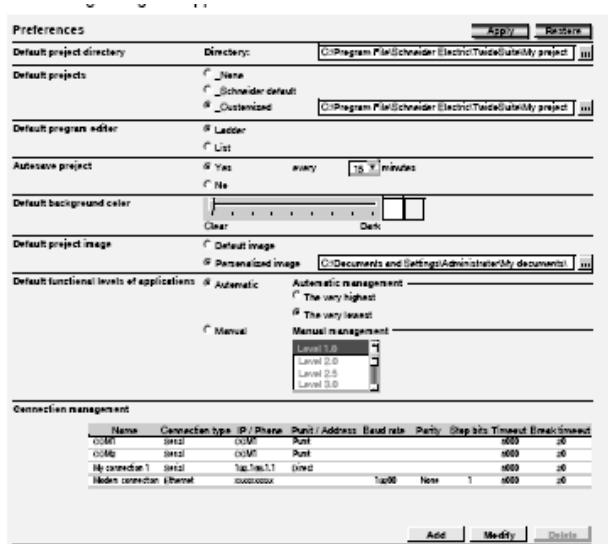
5.1.3.3 ESTABLISHING CONNECTION

The default communication connection between TwidoSuite and the Twido controller is made by a serial communication port, using the TSX PCX 1031 cable and a crossed adapter (see Appendix 1, p. 91).

If a modem is used to connect the PC, this must be indicated in the TwidoSuite software.

To select a connection using TwidoSuite, click Preferences. Result:

The following dialog box appears:



This screen allows you to create, modify or delete a connection.

To use an existing connection, select it from the connection table in Program → Debug → Connect task.

If you have to add, modify or delete a connection, use the "Connection Management" table that displays the list of connections and their properties.

In this case, 2 serial ports are displayed (Com1 and Com2), as well as a modem connection showing a TOSHIBA V.90 model configured to compose the number: xxxxxxxxxxx (modem number).

Note: Compose the number in adjacent number format.

You can change the name of each connection for application maintenance purposes (COM1 or COM2 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.

5.1.3.4 CONFIGURING THE MODEM

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

To add a modem to an open application, follow the procedure described in (See Positioning a Modem, TwidoSuite Programming Software, Online Help)

Once the modem is configured on port 1, the properties must be defined. In the Describe step, double-click the Modem thumbnail

* Result: The Modem Feature (See Modem Configuration, TwidoSuite Programming Software, Online Help) dialog box appears. The Modem properties dialog box lets you either select a known modem, create a new modem, or modify a modem configuration.

Illustration of the Modem Feature dialog box:

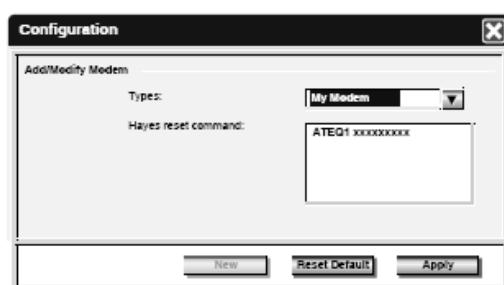


The selected configuration corresponds to the one read in the controller: the Hayes initialization command, then read, is displayed in Hayes standard format.

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.

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

Illustration of the Add/Modify Modem dialog box:



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

In the figure above, "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 TwidoSuite via modems.

5.1.3.5 CONNECTION SEQUENCE

Once TwidoSuite and the Twido controller are ready, establish connection as follows:

Step	Action
1	Power-up the Twido controller and modem.
2	Start your computer and run TwidoSuite.
3	Select Preferences, and select a modem connection from the "Connection Management" table, (for example, "My modem" or the name you have given to your modem connection – see "creation of a connection").
4	Connect TwidoSuite

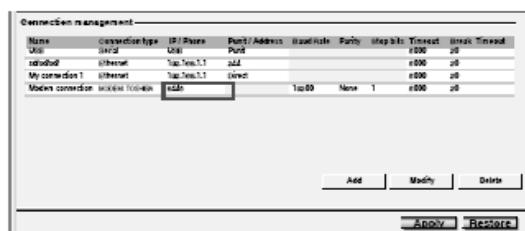
Note: If you want to use your modem connection all the time, click "file", "preferences", and select "my modem" (or the name you have given it). TwidoSuite will now memorize this preference.

5.1.3.6 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 "AT&F" command to establish whether the modem is connected. If the controller receives an answer, the initialization string is sent to the modem.

5.1.3.7 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



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,xxxxxxxxxx or 9,xxxxxxxxxx

Connection management							
Name	Description type	IP / Phone	Port / Address	Associate	Party	Stop bits	Timeout
Unit	Serial		Port			n800	20
individual	Ethernet	192.168.1.1	244			n800	20
My connection 1	Ethernet	192.168.1.1	Direct			n800	20
Modem connection	Modem	192.168.1.1	8 serial port	1		n800	20

[Add](#) [Modify](#) [Delete](#)

[Apply](#) [Restore](#)

For international calls, the syntax is: +1xxxxxxxxxx, for example. And if you are using a switchboard: 0,+1xxxxxxxxxx

Connection management							
Name	Description type	IP / Phone	Port / Address	Associate	Party	Stop bits	Timeout
Unit	Serial		Port			n800	20
individual	Ethernet	00:0c:29:1e:11:11	244			n800	20
My connection 1	Ethernet	00:0c:29:1e:11:11	Direct			n800	20
Modem connection	Modem	192.168.1.1	8 serial port	1		n800	20

[Add](#) [Modify](#) [Delete](#)

[Apply](#) [Restore](#)

5.1.3.8 FREQUENTLY ASKED QUESTIONS

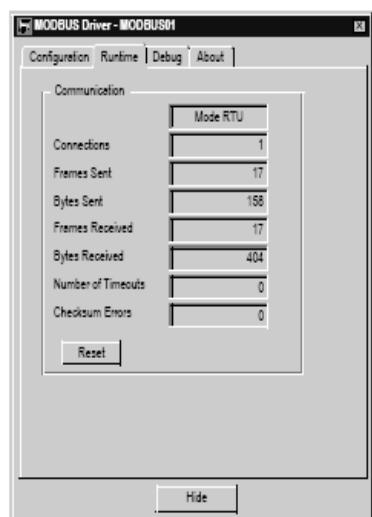
When your communication has been established for a few minutes, you can experience some communication errors. In this case, you must adjust the communication parameters.

TwidoSuite 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" table, accessible using TwidoSuite by clicking

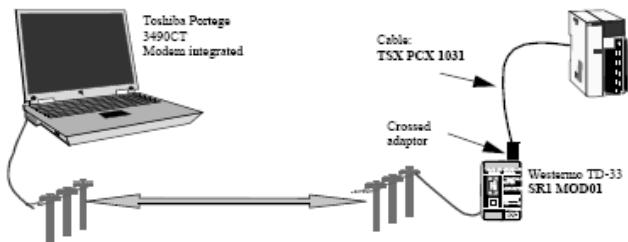
Preferences. Click on the "timeout" field, then click the modification button and enter a new, higher value. The default value is "5000", in milliseconds.

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



5.1.3.9 EXAMPLES

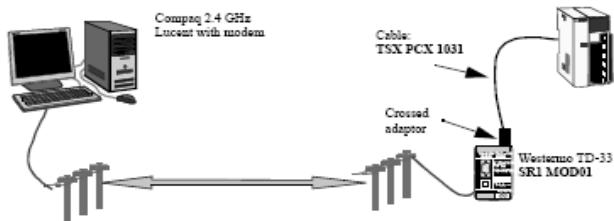
- ▲ Example 1: TwidoSuite connected to a TWD LMDA 20DRT (Windows 98 SE).
- ▲ PC: Toshiba Portege 3490CT running Windows 98,
- ▲ Modem (internal on PC): Toshiba internal V.90 modem,
- ▲ Twido Controller: TWD LMDA 20DRT version 2.0,
- ▲ Modem (connected to Twido): Type Westermo TD-33 / V.90, reference SR1 MOD01, available from the new Twido catalog (September 03) (see Appendix 2, p. 92), (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. 91). 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 (TwidoSuite "preferences" then "Connection management") were established with their default value, with a timeout of 5000 and break timeout of 20.

- ▲ Example 2: TwidoSuite 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. 92), (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. 91). You can also use the TSX PCX 1130 cable (RS485/232 conversion and Rx/Tx crossing).

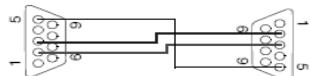


The first test involves using two analog telephone lines internal to the company, and not using the entire number – just the extension (hence only 4 digits for the internal Toshiba V.90 modem telephone number).

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

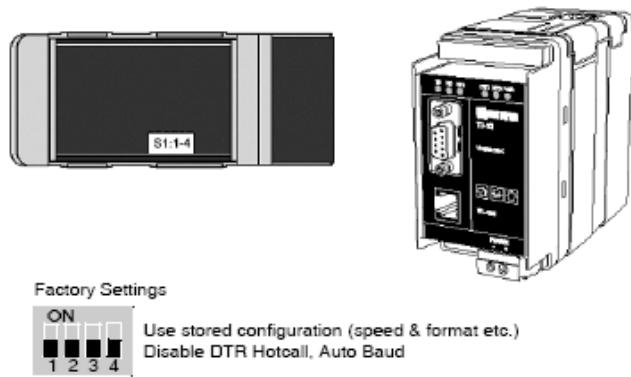
5.1.3.10 APPENDIX 1

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



5.1.3.11 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:



Note:

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

5.1.3.12 APPENDIX 3

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



Note:

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

5.1.3.13 APPENDIX 4

Reference numbers of the products used in this document:

- ▲ Twido product: TWD LMDA 20DRT,
- ▲ TwidoSuite 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:

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

5.1.4 REMOTE LINK COMMUNICATIONS

5.1.4.1 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.

5.1.4.2 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
TWDLC•A10/16/24DRF, TWDLCA•40DRF, 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. <i>Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.</i>
TWDNOZ485T	2	Communication module equipped with a 3-wire EIA RS-485 port with a terminal. <i>Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.</i>
TWDNAC485D	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a miniDIN connector. <i>Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.</i>
TWDNAC485T	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a terminal. <i>Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.</i>
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. <i>Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have a Communication expansion module.</i>

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.

5.1.4.3 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 TwidoSuite.

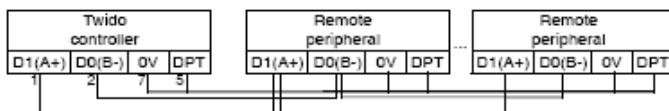
Note: The DPT to 0V connection is only necessary if you are connected to a base controller on Port 1.



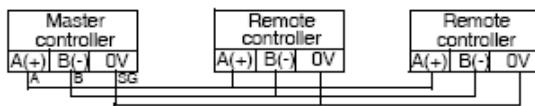
Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

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

Mini-DIN connection



Terminal block connection



5.1.4.4 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.

5.1.4.5 MASTER CONTROLLER CONFIGURATION

The master controller is configured using TwidoSuite 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 TwidoSuite corresponds to address 0.

To configure a controller as a Master Controller, use TwidoSuite 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.

5.1.4.6 REMOTE CONTROLLER CONFIGURATION

A remote controller is configured using TwidoSuite, 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 %ONW A maximum of 4 input words and 4 output words can be transmitted to and from each peer.

5.1.4.7 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 TwidoSuite 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 (TwidoSuite cable NOT connected)
	1	master/slave: DPT active (TwidoSuite 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

5.1.4.8 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.

5.1.4.9 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.

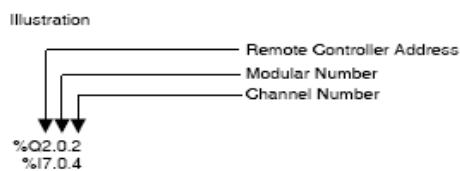
5.1.4.10 MASTER CONTROLLER STOP

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.

5.1.4.11 REMOTE I/O DATA ACCESS

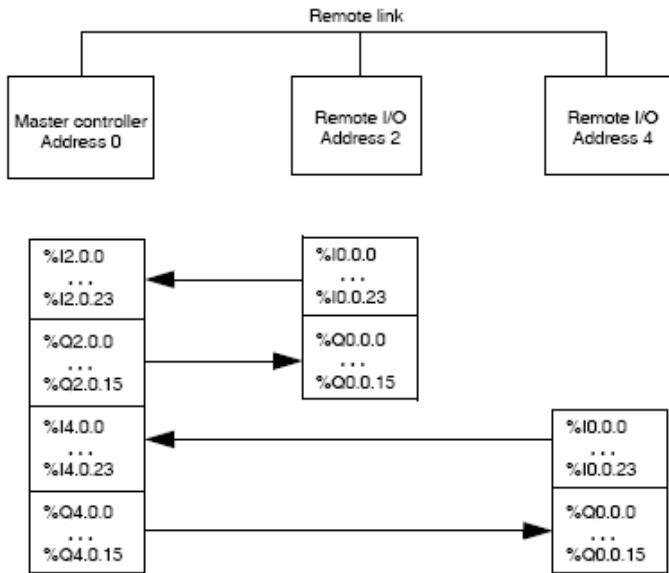
The remote controller configured to be a remote I/O does not have or execute its own application program. The remote controller's base discrete inputs and outputs are a simple extension of the master controller's. The application must only use the full three digit addressing mechanism provided.

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



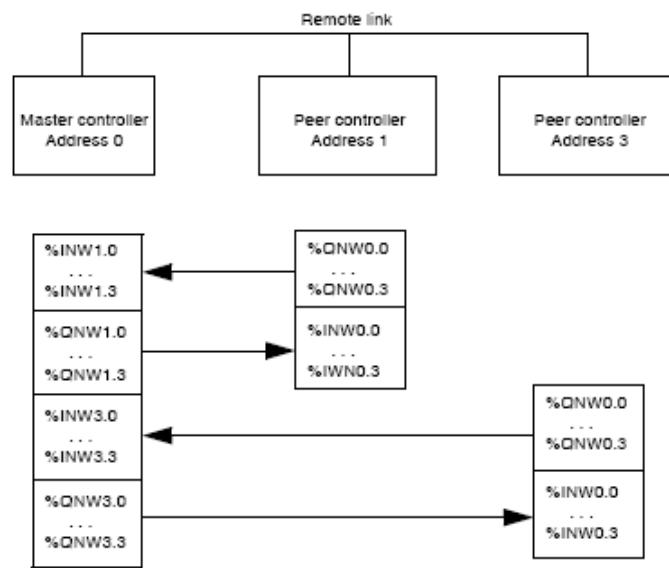
To communicate with remote I/O, the master controller uses the standard input and output notation of $\%I$ and $\%Q$. To access the third output bit of the remote I/O configured at address 2, instruction $\%Q2.0.2$ is used. Similarly, to read the fifth input bit of the remote I/O configured at location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

Note: The master is restricted to accessing only the discrete 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**5.1.4.12 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 ":" 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.

5.1.4.13 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	

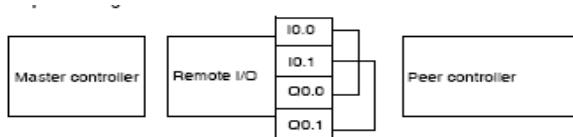
5.1.4.14 REMOTE LINK EXAMPLE

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:

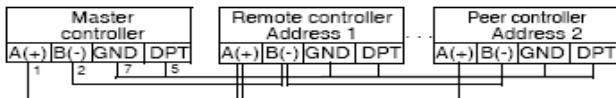


The hardware configuration is three base controllers of any type. Port 1 is used for two communication modes. One mode is to configure and transfer the application program with TwidoSuite. The second mode is for the Remote Link network. If available, an optional Port 2 on any of the controllers can be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

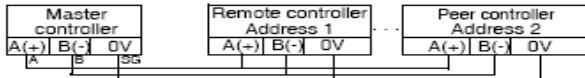
Note: In this example, the two first inputs on the Remote I/O are hard wired to the first two outputs.

Step 2: Wire the controllers

Mini-DIN connection

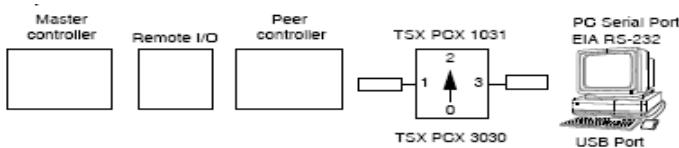


Terminal block connection



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.

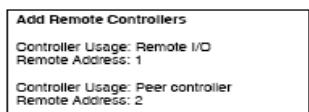
Step 4: Configure the Software:

Each of the three controllers uses TwidoSuite 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)".



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

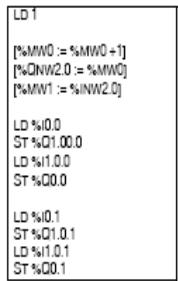


For the controller configured as peer, verify that the controller communication setup is set to "Remote Link" and the address is set to "2".



Step 5: Write the applications:

For the Master controller, write the following application program:



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:



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.

5.1.5 ASCII COMMUNICATIONS

5.1.5.1 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.

5.1.5.2 HARDWARE CONFIGURATION

An ASCII link (see system bits %S103 and %S104 (See System Bits (%S), p. 564)) 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
TWDLC•A10/16/24DRF, TWDLC•40DRF, 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.

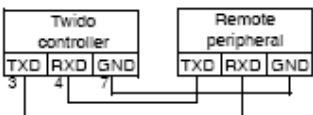
5.1.5.3 NOMINAL CABLING

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 TwidoSuite software.

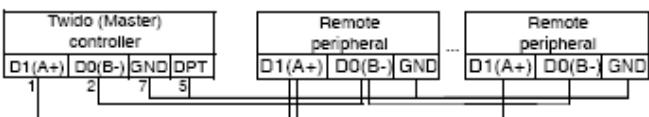
Cable connections to each device are illustrated below.

Mini-DIN connection

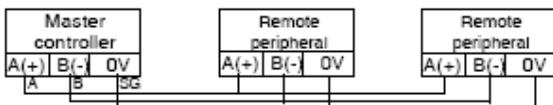
RS-232 EIA cable



RS-485 EIA cable



Terminal block connection



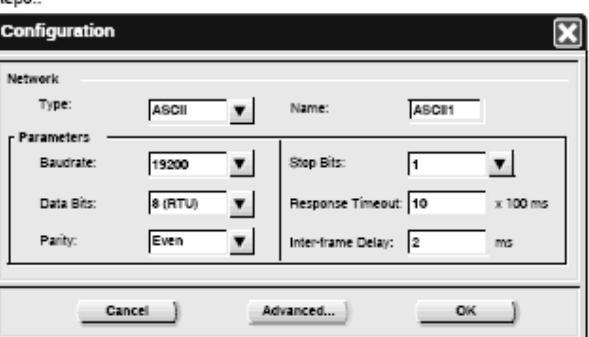
5.1.5.4 SOFTWARE CONFIGURATION

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 TwidoSuite.
2	Create in your application a transmission/reception table that will be used by the EXCHx instruction.

5.1.5.5 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	Declare the ASCII network in the Describe step of TwidoSuite (see (See How to Create a Network (Modbus Example), TwidoSuite Programming Software, Online Help) and (See How to Create a Network (Modbus Example), TwidoSuite Programming Software, Online Help) for ASCII).
3	Select Port 1 (or Port 2 if installed) to configure in the Describe window (see (See Configuring an Object, TwidoSuite Programming Software, Online Help)).
4	To configure the ASCII element, use any of the two methods: <ul style="list-style-type: none"> Click the Configure icon from the toolbar then select the ASCII element in the describe graphic, Double click the ASCII element in the describe graphic.
5	To bring up the Feature dialog box (See Configuring an Object, TwidoSuite Programming Software, Online Help) associated to the ASCII link hardware parameters, use any of the two methods: <ul style="list-style-type: none"> Click the Configure icon from the toolbar then select the ASCII link in the describe graphic, Double click the ASCII link in the describe graphic.
6	Configure the Feature dialog box that appears, as explained in the subsequent steps.. 
7	Set the communication parameters.
8	Click Advanced button to set the advanced parameters.

5.1.5.6 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
	Transmitted Byte n+1	
Reception table	Received Byte 1	Received Byte 2
	...	Received Byte p
	Received Byte p+1	

5.1.5.7 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

5.1.5.8 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.

5.1.5.9 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).

5.1.5.10 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 %KWi: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.

5.1.5.11 %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.

5.1.5.12 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 TwidoSuite 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

5.1.5.13 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.

5.1.5.14 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.

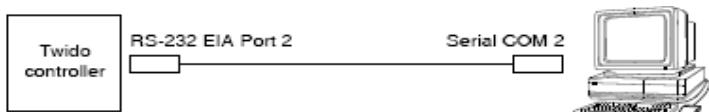
5.1.5.15 ASCII LINK EXAMPLE

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.

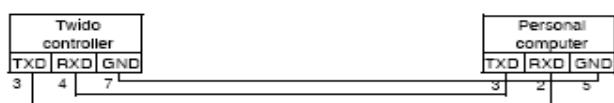
Step 1: Configure the Hardware:



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		Terminal Emulator on a PC
Serial Port 2		
Protocol	ASCII	Port: COM2
Address		Baud Rate: 19200
Baud Rate	19200	Data: 8 Bit
Data Bits	8	Parity: None
Parity	None	Stop: 1 Bit
Stop Bit	1	Flow control: None
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		

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

Use TwidoSuite 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#0A00]
LD 1
AND %MSG2.D
[EXCH2 %MW10:8]
LD %MSG2.E
ST %Q0.0
END

```

Use TwidoSuite 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	0A00	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.

5.1.6 MODBUS COMMUNICATIONS

5.1.6.1 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.

5.1.6.2 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. 564).

The table below lists the devices that can be used:

Remote	Port	Specifications
TWDLOA10/16/24DRF, TWDLOA40DRF, 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.
TWDXCP0DM	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.

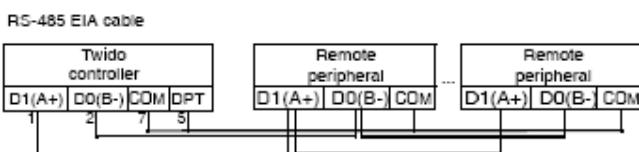
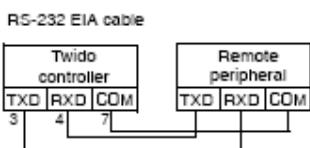
5.1.6.3 NOMINAL CABLING

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

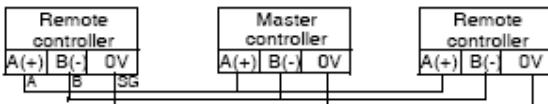
Note: If port 1 is used on the Twido controller, the DPT signal on pin 5 must be tied to the circuit common (COM) on pin 7. This signifies to the Twido controller that the communications through port 1 is Modbus and is not the protocol used to communicate with the TwidoSuite software.

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

Mini-DIN connection



Terminal block connection



5.1.6.4 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.

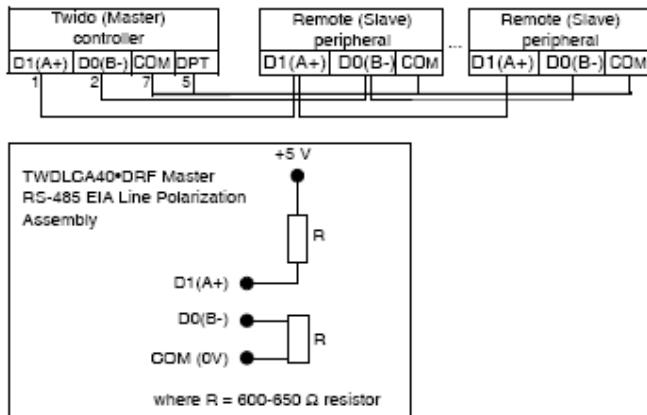
The external line polarization assembly on the TWDLCA•40DRF mini-DIN RS-485 EIA line shall consist in:

- ▲ 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).

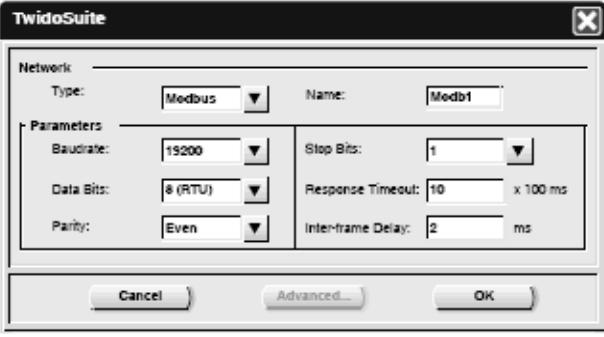
5.1.6.5 SOFTWARE CONFIGURATION

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 TwidoSuite.
2	Create in your application a transmission/reception table that will be used by the EXOHx instruction.

5.1.6.6 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	Declare the Modbus network in the Describe step of TwidoSuite (see (See How to Create a Network (Modbus Example), TwidoSuite Programming Software, Online Help).
3	Select Port 1 (or Port 2 if installed) to configure in the Describe window (see (See Configuring an Object, TwidoSuite Programming Software, Online Help).
4	To configure the Modbus element, use any of the two methods: <ul style="list-style-type: none">• Click the Configure icon from the toolbar then select the Modbus element in the describe graphic,• Double click the Modbus element in the describe graphic.
5	To bring up the Feature dialog box (See Configuring an Object, TwidoSuite Programming Software, Online Help) associated to the Modbus link hardware parameters, use any of the two methods: <ul style="list-style-type: none">• Click the Configure icon from the toolbar then select the Modbus link in the describe graphic,• Double click the Modbus link in the describe graphic.
6	Configure the Feature dialog box that appears, as explained in the subsequent steps.: 
7	Select Modbus in the Protocol :Type box.
8	Set the associated communication parameters.

5.1.6.7 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.

5.1.6.8 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 which was actually received would be entered into the fourth location in the table.

5.1.6.9 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.

5.1.6.10 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, TwidoSuite 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

5.1.6.11 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).

5.1.6.12 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.

5.1.6.13 %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.

5.1.6.14 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 TwidoSuite 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

5.1.6.15 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.

5.1.6.16 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.

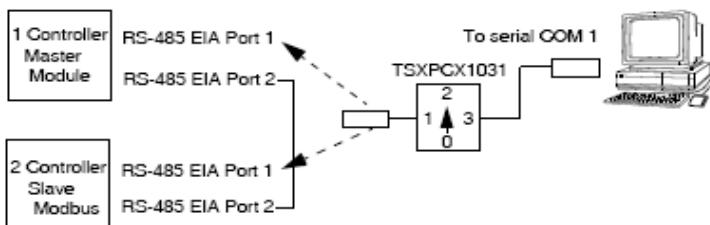
5.1.6.17 MODBUS LINK EXAMPLE 1

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 7) must be tied to circuit common (pin 7). This conditioning of DPT determines if TwidoSuite 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 TWDNOD485-
Hardware => Controller Comm. Setting	
Serial Port 2	Serial Port 2
Protocol	Modbus
Address	1
Baud Rate	19200
Data Bits	8 (RTU)
Parity	None
Stop Bit	1
Response Timeout (x100ms)	100
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:

<pre> 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 </pre>	<pre> LD 1 [%MW0 := 16#6566] [%MW1 := 16#6768] [%MW2 := 16#6970] [%MW3 := 16#7172] END </pre>
--	---

Using TwidoSuite, 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:

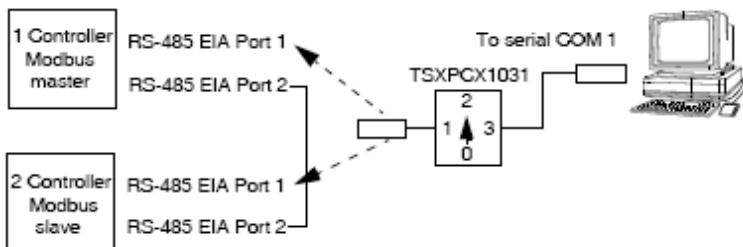
Address	Current	Retained	Format
1 %MW5	0203	0000	Hexadecimal
2 %MW6	0008	0000	Hexadecimal
3 %MW7	6566	0000	Hexadecimal
4 %MW8	6768	0000	Hexadecimal
5 %MW9	6970	0000	Hexadecimal
6 %MW10	7172	0000	Hexadecimal

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.

5.1.6.18 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		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

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#6788]
LD 1
AND %MSG2.D
[EXCH2 %MW0:11]
LD %MSG2.E
ST %Q0.0
END

```

```

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

```

Using TwidoSuite, 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.

5.1.7 STANDARD MODBUS REQUESTS

5.1.7.1 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

5.1.7.2 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

5.1.7.3 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.

5.1.7.4 MODBUS MASTER: WRITE BIT

This table represents Request 05.



	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)	06 (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.

5.1.7.5 MODBUS MASTER: WRITE WORD

This table represents Request 06.

	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).

5.1.7.6 MODBUS MASTER: WRITE OF N BITS

This table represents Request 15.

	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	
Reception table (after response)		Slave@(1..247)	15 (Response code)
		Address of the 1 st bit written	
		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.

5.1.7.7 MODBUS MASTER: WRITE OF N WORDS

This table represents Request 16.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	8 + (2 ¹ 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 ¹ 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.

5.1.8 TRANSPARENT READY IMPLEMENTATION CLASS (TWIDO SERIAL A05, ETHERNET A15)

5.1.8.1 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>

5.1.8.2 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)

6 BUILT-IN ANALOG FUNCTIONS

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 95

Analog Channel 96

6.1.1 ANALOG POTENTIOMETER

6.1.1.1 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

6.1.1.2 PROGRAMMING

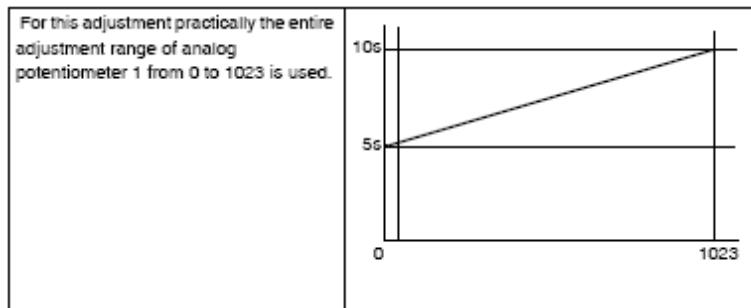
The numerical values, from 0 to 1023 for analog potentiometer 1, and from 0 to 511 for analog potentiometer 2, corresponding to the analog values provided by these potentiometers are contained in the following two input words:

- ▲ %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.

6.1.1.3 EXAMPLE

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

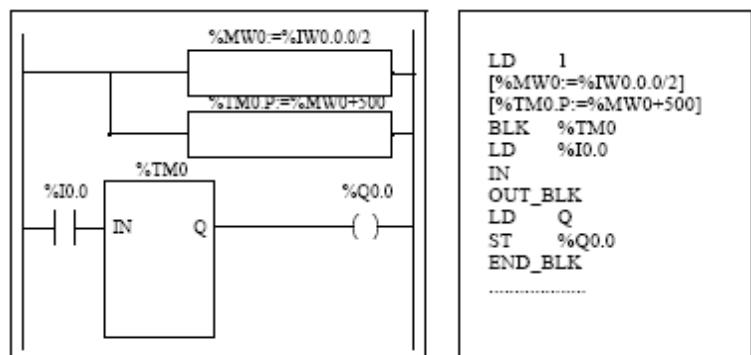


The following parameters are selected at configuration for the time-delay block

%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:



6.1.2 ANALOG CHANNEL

6.1.2.1 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.

6.1.2.2 PRINCIPLE

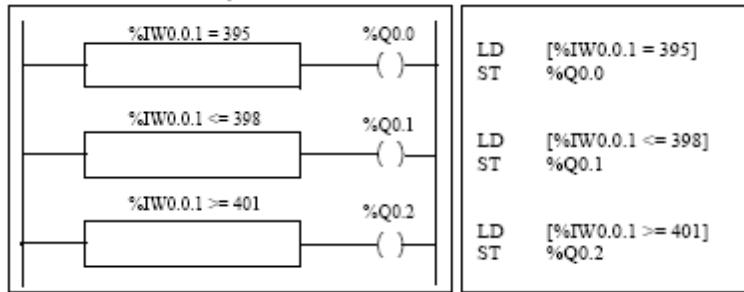
An analog to discrete converter samples an input voltage from 0 to 10 V to a discrete value from 0 to 511. This value is stored in system word %IW0.0.1. The value is linear through the entire range, so that each increment is approximately 20 mV (10 V/512). Once the system detects value 511, the channel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

6.1.2.3 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:



7 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 98

Addressing Analog Inputs and Outputs 99

Configuring Analog Inputs and Outputs 100

Analog Module Status Information 104

Example of Using Analog Modules 105

7.1.1 ANALOG MODULE OVERVIEW

7.1.1.1 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
TWDAMM6HT	4 In, 2 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

7.1.1.2 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 discrete output, the default setpoint is zero.

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

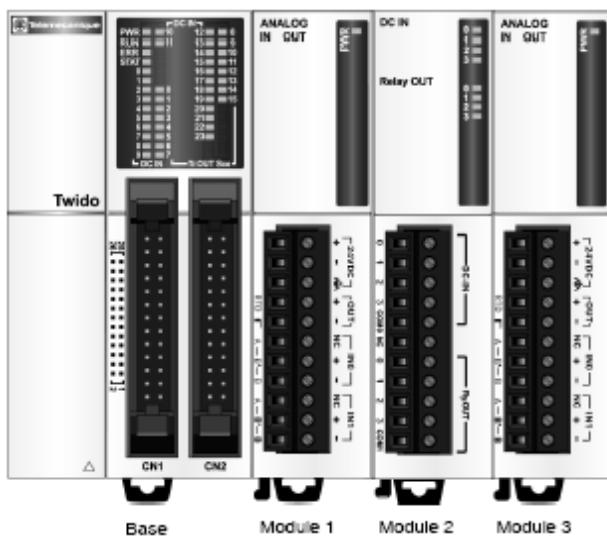
7.1.2 ADDRESSING ANALOG INPUTS AND OUTPUTS

7.1.2.1 INTRODUCTION

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

7.1.2.2 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 discrete 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	%IWO.0.0			
Built-in analog channel	%IWO.0.1			
Analog input channel 1		%IWO.1.0		%IWO.3.0
Analog input channel 2		%IWO.1.1		%IWO.3.1
Analog output channel 1		%QWO.1.0		%QWO.3.0
Discrete input channels			%IO.2.0 - %IO.2.3	
Discrete output channels			%QO.2.0 - %QO.2.3	

7.1.3 CONFIGURING ANALOG INPUTS AND OUTPUTS

7.1.3.1 INTRODUCTION

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

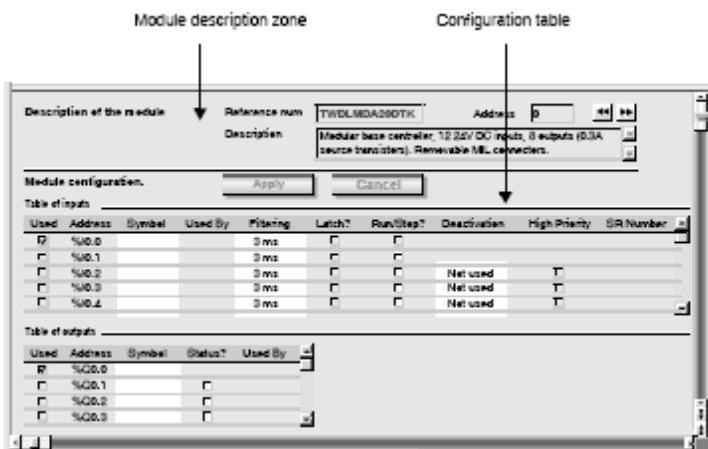
7.1.3.2 CONFIGURING ANALOG I/O

Use the Configuration Editor to set parameters of analog I/O modules that you added as expansion modules when you described the system (see (See Positioning a Module, TwidoSuite Programming Software, Online Help)).

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

7.1.3.3 CONFIGURATION EDITOR CONTENTS

The upper part of the dialog box shows a Description zone with the module reference number and a short description.



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.

7.1.3.4 DESCRIPTION

The Description zone displays a short summary of the module.

7.1.3.5 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 (%IWi.0, %IWi.1), 1 Output (%QWi.0)
TWDAMM3HT	2 Inputs (%IWi.0, %IWi.1), 1 Output (%QWi.0)
TWDAMM6HT	4 Inputs (%IWi.0 to %IWi.3), 2 Outputs (%QWi.0, %QWi.1)
TWDAMI2HT	2 Inputs (%IWi.0, %IWi.1)
TWDAMO1HT	1 Output (%QWi.0)
TWDAVO2HT	2 Outputs (%QWi.0, %QWi.1)
TWDAMI4LT	4 Inputs (%IWi.0 to %IWi.3)
TWDAMI8HT	8 Inputs (%IWi.0 to %IWi.7)
TWDARI8HT	8 Inputs (%IWi.0 to %IWi.7)

7.1.3.6 SYMBOL

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

7.1.3.7 INPUT AND OUTPUT TYPE

This identifies the mode of a channel. The choices depend on the channel and type of module. For the TWDAM01HT, TWDAMM3HT and TWDALM3LT, you can configure the single output channel type as:

Type
Not used
0 - 10 V
4 - 20 mA

For the TWDAMM6HT, you can configure the 4 input and the 2 output channel types as:

Input type
0 - 10 V
4 - 20 mA

For the TWDAMI2HT and TWDAMM3HT, you can configure the 2 input channel types as:

Type
Not used
0 - 10 V
4 - 20 mA

For the TWDALM3LT, you can configure the 2 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 4 input types as:

Input type	Type
Voltage	Not used 0-10 V
Current	Not used 0-20 mA

Input type	Type
Voltage	Not used 0-10 V
Current	Not used 0-20 mA

For the TWDAMI8HT, you can configure the 8 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 TwidoSuite for a current type of configuration, you may permanently damage the analog module. Ensure that the wiring is in agreement with the TwidoSuite configuration.

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

7.1.3.8 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 TWDAM01HT TWDAMM3HT TWDAMM6HT TWDAMI2HT TWDAMI4LT	
	-2048	2047		TWDAV02HT	
	0	1023		TWDAMI8HT TWDARI8HT	
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 TwidoSuite 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 TwidoSuite 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)	

7.1.3.9 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 Farenheit.

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 Farenheit.

Here is a table of corresponding min./max. Tref values between units:

Unit	Min. value	Max. value
Kelvin	1	650
Celsius	-272	376
Farenheit	-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.

7.1.4 ANALOG MODULE STATUS INFORMATION

7.1.4.1 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	For standard analog module, %SW8x is described as follows: Bit [0] All analog channels in normal state Bit [1] Module in initialization state Bit [2] Power supply default Bit [3] Configuration default Bit [4] Conversion in running for input channel 0 Bit [5] Conversion in running for input channel 1 Bit [6] Invalid parameter for input channel 0 Bit [7] Invalid parameter for input channel 1 Bit [8 & 9] Not used Bit [10] Overflow value for input channel 0 Bit [11] Overflow value for input channel 1 Bit [12] Underflow value for input channel 0 Bit [13] Underflow value for input channel 1 Bit [14] Not used Bit [15] Invalid parameter for output channel
%SW80 cont'd	Base I/O Status cont'd	For TWDAMI4LT analog module, %SW8x is described as follows: Bit [0 & 1] Channel 0 state 0 0: Analog channel in normal state 0 1: Invalid parameter for input channel 1 0: Unavailable input value (module in initialization state, conversion in running), 1 1: Invalid value for input channel (overflow or underflow value) Bit [2 & 3] Channel 1 state (same description as bit [0 & 1]) Bit [4 & 5] Channel 2 state (same description as bit [0 & 1]) Bit [6 & 7] Channel 3 state (same description as bit [0 & 1]) Bit [8 to 15] Not used

System Word	Function	Description
%SW80 cont'd	Base I/O Status cont'd	For TWDAMI8HT analog module, %SW8x is described as follows: Bit [0 & 1] Channel 0 state 0 0: Analog channel in normal state 0 1: Invalid parameter for input channel 1 0: Unavailable input value (module in initialization state, conversion in running), 1 1: Invalid value for input channel (overflow or underflow value) Bit [2 & 3] Channel 1 state (same description as bit [0 & 1]) Bit [4 & 5] Channel 2 state (same description as bit [0 & 1]) Bit [6 & 7] Channel 3 state (same description as bit [0 & 1]) Bit [8 & 9] Channel 4 state (same description as bit [0 & 1]) Bit [10 & 11] Channel 5 state (same description as bit [0 & 1]) Bit [12 & 13] Channel 6 state (same description as bit [0 & 1]) Bit [14 & 15] Channel 7 state (same description as bit [0 & 1])
%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	

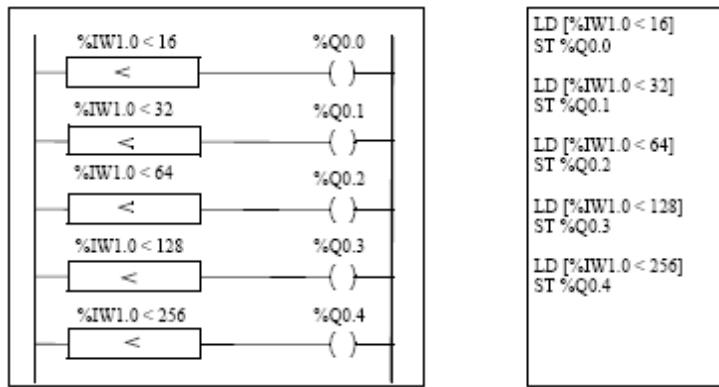
7.1.5 EXAMPLE OF USING ANALOG MODULES

7.1.5.1 INTRODUCTION

This section provides an example of using Analog modules available with Twido.

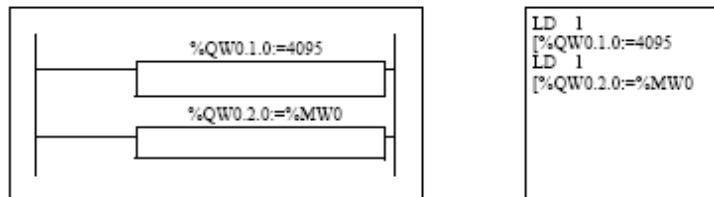
7.1.5.2 EXAMPLE: ANALOG INPUT

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.



7.1.5.3 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:



- ▲ 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

8 INSTALLING THE AS-INTERFACE V2 BUS

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 107

General functional description 108

Software set up principles 110

Description of the configuration screen for the AS-Interface bus 111

Configuration of the AS-Interface bus 112

Description of the AS-Interface Window in Online Mode 117

Modification of Slave Address 120

Updating the AS-Interface bus configuration in online mode 120

Automatic addressing of an AS-Interface V2 slave 123

How to insert a slave device into an existing AS-Interface V2 configuration 124

Automatic replacement of a faulty AS-Interface V2 slave 124

Addressing I/Os associated with slave devices connected to the AS-Interface V2 bus 125

Programming and diagnostics for the AS-Interface V2 bus 126

AS-Interface V2 bus interface module operating mode: 130

8.1.1 PRESENTATION OF THE AS-INTERFACE V2 BUS

8.1.1.1 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 integrated (expansion bus, supply, processor, modules, ASInterface slave devices connected to the bus) then ensure its software implementation.

This second aspect will be carried out from the different TwidoSuite editors:

- either in local mode,
- or in online mode.

8.1.1.2 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•40DRFcompact controller.

8.1.2 GENERAL FUNCTIONAL DESCRIPTION

8.1.2.1 GENERAL INTRODUCTION

For the AS-Interface configuration, TwidoSuite 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

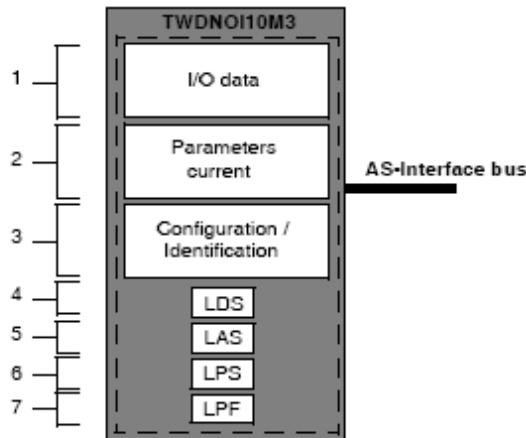
- ▲ Control bus status

For this reason, all data coming from or going to the AS-Interface Master are stored in specific objects (words and bits).

8.1.2.2 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 (COI, 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 TwidoSuite.
7	LPF	List of slaves having a device fault.

8.1.2.3 STRUCTURE OF SLAVE DEVICES

The standard address slaves each have:

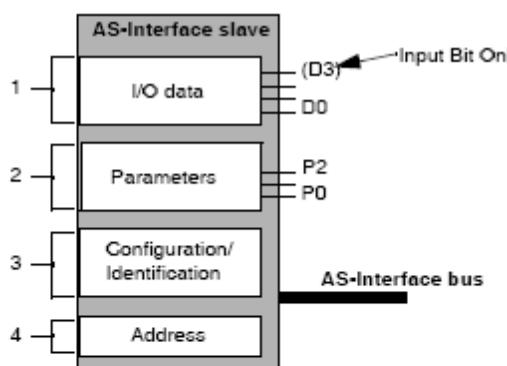
- ▲ 4 input/output bits
- ▲ 4 parametering bits

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: <ul style="list-style-type: none">• 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.

8.1.3 SOFTWARE SET UP PRINCIPLES

8.1.3.1 AT A GLANCE

To respect the philosophy adopted in TwidoSuite, the user should adopt a step-by-step approach when creating an AS-Interface application.

8.1.3.2 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. 189).

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 TWDNDI10M3 on the expansion bus.
	Configuration of the module channel	Choice of "master" modes.
	Declaration of slave devices	Selection for each device: <ul style="list-style-type: none">• 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: <ul style="list-style-type: none">• the AS-Interface Window 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, TwidoSuite will not permit another one to be declared.

8.1.3.3 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.

8.1.4 DESCRIPTION OF THE CONFIGURATION SCREEN FOR THE AS-INTERFACE BUS

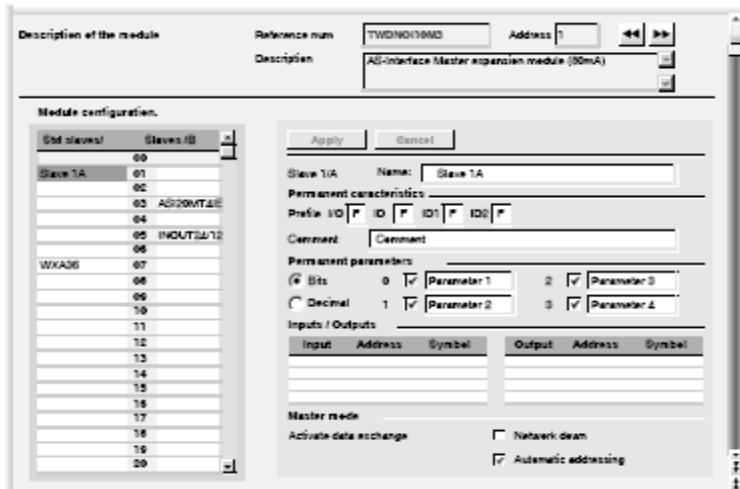
8.1.4.1 AT A GLANCE

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.

8.1.4.2 ILLUSTRATION OF OFFLINE MODE

Illustration of the configuration screen in offline mode:



8.1.4.3 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 xx/A/B	Configuration of the selected slave: <ul style="list-style-type: none"> 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 2 buttons:

Buttons	Description
Apply	Saves the AS-Interface Bus current configuration data. The configuration can then be transferred to the Twido controller.
Cancel	Discard all changes in progress.

Note: Changes in the configuration screen can only be made in offline mode.

8.1.5 CONFIGURATION OF THE AS-INTERFACE BUS

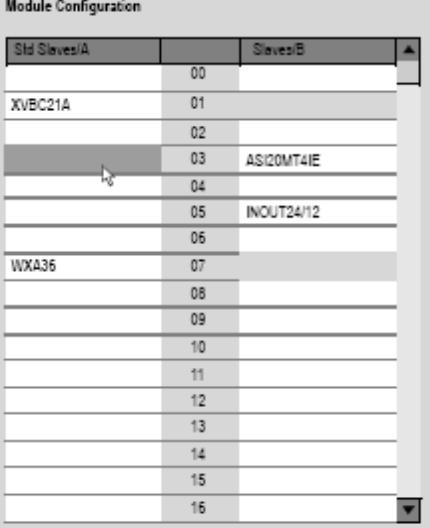
8.1.5.1 INTRODUCTION

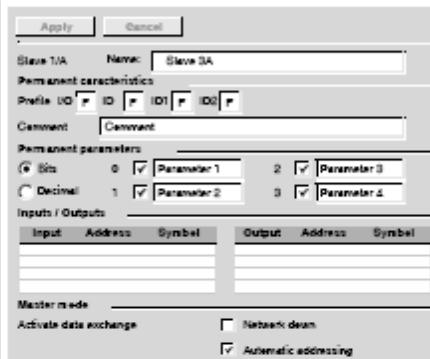
AS-Interface bus configuration takes place in the configuration screen in local mode.

Once the AS-Interface Master and the master modes have been selected, configuration of the AS-Interface bus consists of configuring the slave devices.

8.1.5.2 PROCEDURE FOR DECLARING AND CONFIGURING A SLAVE

Procedure for creating or modifying a slave on the AS-Interface V2 bus:

Step	Action
1	Select the desired address cell (not grayed out) in the bus image: 

Step	Action
2	<p>In the slave configuration screen, enter or modify:</p> <ul style="list-style-type: none"> the name of the new profile (limited to 13 characters), a comment (optional). <p>Or click Insert from catalog button  in the function bar and select a slave from the pre-configured AS-Interface profile family.</p> <p>Illustration of a Configuration Screen for a slave:</p>  <p>Note:</p> <ul style="list-style-type: none"> 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
3	<p>Enter:</p> <ul style="list-style-type: none"> the ID code (corresponds to the input/output configuration), the ID code (identifier), (plus ID1 and for an extended type). <p>Note:</p> <p>The "Inputs" and "Outputs" fields show the number of input and output channels. They are automatically implemented when the IO code is entered.</p>
4	<p>For each parameter define:</p> <ul style="list-style-type: none"> 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). <p>Note:</p> <p>The selected parameters are the image of permanent parameters to be provided to the AS-Interface Master.</p>

Step	Action
5	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.
6	Confirm the slave configuration by clicking on the Apply button. The result is the check that: <ul style="list-style-type: none"> ● the ID 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 Insert from catalog, you can create and configure slaves in "Private family" (other than those in the Schneider AS-Interface catalog).

8.1.5.3 AS-INTERFACE CATALOG

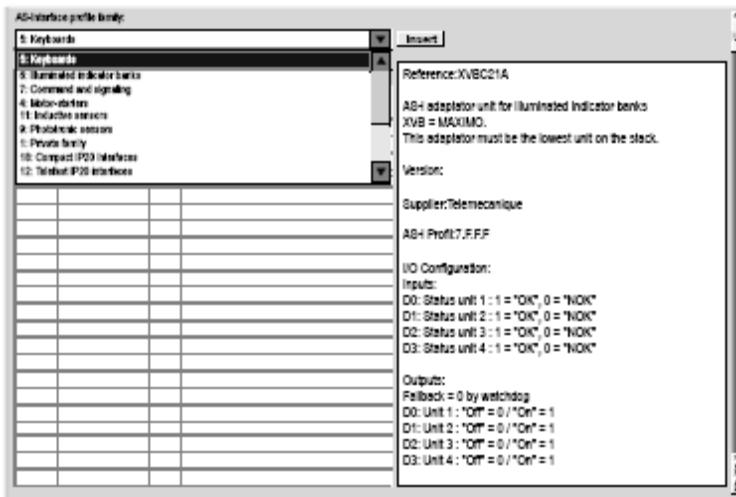
The Insert from catalog button



can be used to facilitate configuration of slaves on the bus. When you use a slave from the Schneider products family, use this button to simplify and speed up configuration.

Clicking on Insert from catalog opens the following pane:

The drop-down menu gives you access to all product families of the Schneider ASInterface catalog:



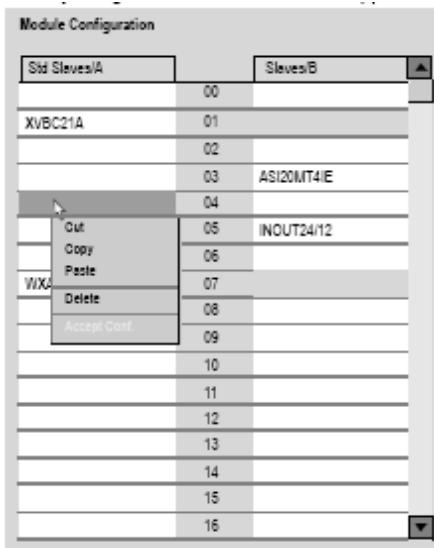
After you have selected a product, the list of corresponding slaves appears. Click on the required slave and validate by clicking "Insert"

Note:

- ▲ Click the product name in the AS-Interface catalog to display its characteristics in the right pane.
- ▲ You can add and configure slaves that are not part of the Schneider catalog. Simply select the private family and configure the new slave.

8.1.5.4 SHORTCUT MENU

When you right click, a shortcut menu appears:



The shorcut menu is used to:

- ▲ Cut (Ctrl+X)
- ▲ Copy (Ctrl+C)
- ▲ Paste (Ctrl+V)

- ▲ Delete (Del)

8.1.6 DESCRIPTION OF THE AS-INTERFACE WINDOW IN ONLINE MODE

8.1.6.1 AT A GLANCE

When the PC is connected to the controller (after uploading the application to the controller), the AS-Interface Window displays online features.

In online mode, the AS-Interface Window dynamically provides an image of the physical bus that includes the:

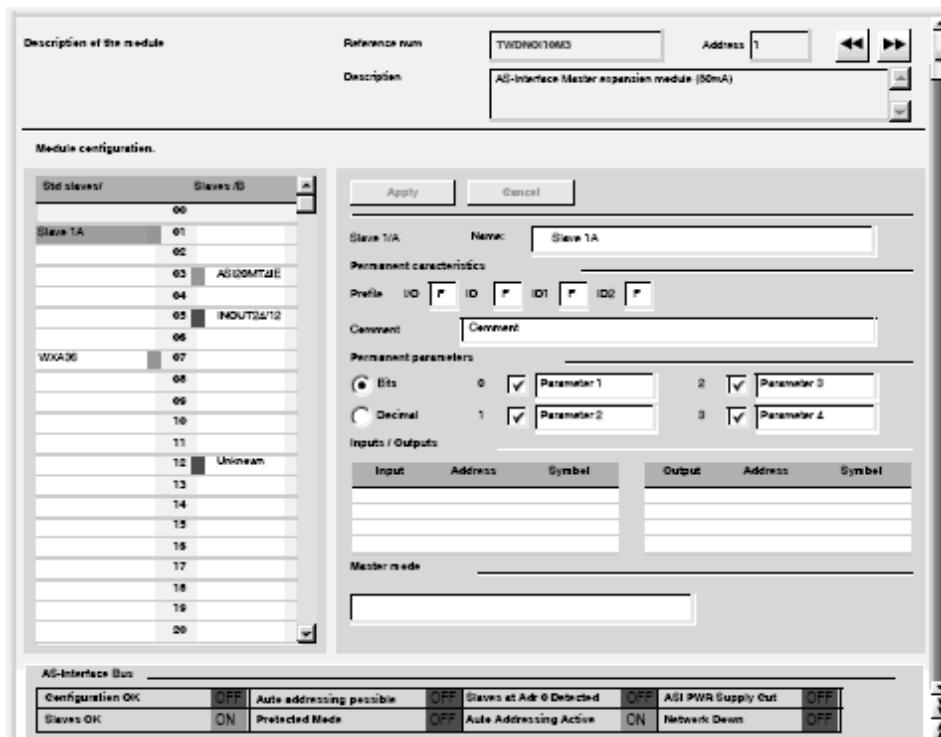
- ▲ List of expected slaves (entered) during configuration with their name, and the list of detected slaves (with unknown names, but otherwise expected),
- ▲ Status of the AS-Interface module and the slave devices,
- ▲ Image of the profile, parameters and input/output values of the selected slaves.

It also enables the user:

- ▲ To obtain diagnostics of the slaves on which an error has occurred (See Displaying Slave Status, p. 181),
- ▲ To modify the address of a slave in online mode (See Modification of Slave Address, p. 182),
- ▲ To transmit the image of the slaves to the configuration screen (See Updating the AS-Interface bus configuration in online mode, p. 184),
- ▲ To address all the slaves with the desired addresses (during the first debugging).

8.1.6.2 ILLUSTRATION OF THE AS-INTERFACE WINDOW

The illustration of the AS-Interface Window (in online mode only) looks like this:



8.1.6.3 DESCRIPTION OF THE AS-INTERFACE WINDOW

The AS-Interface Window provides the same information as the configuration screen (See Description of the Screen in Offline Mode, p. 171).

The differences are listed in the following table:

Schedule	Description
AS-interface V2 configuration	Image of the physical bus. Includes slave status: <ul style="list-style-type: none"> ● 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 xx/A/B	Image of the configuration of the selected slave: <ul style="list-style-type: none"> ● 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. <ul style="list-style-type: none"> ● 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.

8.1.6.4 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").

8.1.7 MODIFICATION OF SLAVE ADDRESS

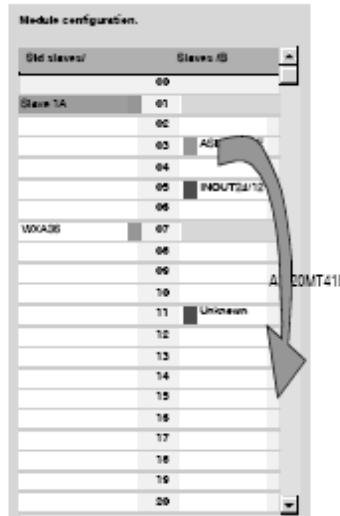
8.1.7.1 AT A GLANCE

From the AS-Interface Window, the user can modify the address of a slave in online mode.

8.1.7.2 MODIFICATION OF SLAVE ADDRESS

The following table shows the procedure for modifying a slave address:

Step	Description
1	Access the AS-Interface Window.
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



Step	Description																																										
	<p>Result: All the slave parameters are automatically checked to see if the operation is possible. Illustration of result:</p> <p>Module configuration.</p> <p>Std slaves Slaves (B)</p> <table border="1"> <tr><td>00</td><td></td></tr> <tr><td>XVBC21A</td><td>01</td></tr> <tr><td></td><td>02</td></tr> <tr><td></td><td>03 AS120MT41E</td></tr> <tr><td></td><td>04</td></tr> <tr><td></td><td>05 INOUT24/12</td></tr> <tr><td></td><td>06</td></tr> <tr><td>WXA3S</td><td>07</td></tr> <tr><td></td><td>08</td></tr> <tr><td></td><td>09</td></tr> <tr><td></td><td>10</td></tr> <tr><td></td><td>11 Unknown</td></tr> <tr><td></td><td>12 Unknown</td></tr> <tr><td></td><td>13</td></tr> <tr><td></td><td>14</td></tr> <tr><td></td><td>15 Unknown</td></tr> <tr><td></td><td>16</td></tr> <tr><td></td><td>17</td></tr> <tr><td></td><td>18</td></tr> <tr><td></td><td>19</td></tr> <tr><td></td><td>20</td></tr> </table>	00		XVBC21A	01		02		03 AS120MT41E		04		05 INOUT24/12		06	WXA3S	07		08		09		10		11 Unknown		12 Unknown		13		14		15 Unknown		16		17		18		19		20
00																																											
XVBC21A	01																																										
	02																																										
	03 AS120MT41E																																										
	04																																										
	05 INOUT24/12																																										
	06																																										
WXA3S	07																																										
	08																																										
	09																																										
	10																																										
	11 Unknown																																										
	12 Unknown																																										
	13																																										
	14																																										
	15 Unknown																																										
	16																																										
	17																																										
	18																																										
	19																																										
	20																																										
	<p>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.</p>																																										

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.

8.1.8 UPDATING THE AS-INTERFACE BUS CONFIGURATION IN ONLINE MODE

8.1.8.1 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.

8.1.8.2 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 AS-Interface window for the detected address.

The following table describes the procedure for transferring the image of the "Unknown" slave to the configuration screen:

Step	Description
1	Access the AS-Interface window.
2	Select the desired slave in the "AS-interface V2 Configuration" zone.

Step	Description
3	<p>Right click on the mouse to select "Transfer Conf".</p> <p>Illustration:</p>  <p>Result: The image of the selected slave (image of the profile and parameters) is then transferred to the configuration screen.</p>
4	Repeat the operation for each of the slaves whose image you would like to transfer to the configuration screen.

8.1.8.3 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:

Module configuration.	
Std slaves/	Slaves/B
00	
XVBC21A	01
02	
03	AS120MTSE
04	
05	X INOUT24/12
06	
WXA26	07
08	
09	
10	
11	Unknown
12	
13	
14	
15	Unknown
16	
17	
18	
19	
20	

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. 182)), 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.

8.1.8.4 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. 172).

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.
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.

8.1.9 AUTOMATIC ADDRESSING OF AN AS-INTERFACE V2 SLAVE

8.1.9.1 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 TwidoSuite.

TwidoSuite software offers an automatic slave addressing utility so that an ASInterface console does not have to be used.

The automatic addressing utility is used for:

- ▲ replacing a faulty slave,
- ▲ inserting a new slave.

8.1.9.2 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.

8.1.10 HOW TO INSERT A SLAVE DEVICE INTO AN EXISTING AS-INTERFACE V2 CONFIGURATION

8.1.10.1 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. 188),
- 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.

8.1.10.2 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.

8.1.11 AUTOMATIC REPLACEMENT OF A FAULTY AS-INTERFACE V2 SLAVE

8.1.11.1 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. 188).

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).

8.1.12 ADDRESSING I/Os ASSOCIATED WITH SLAVE DEVICES CONNECTED TO THE AS-INTERFACE V2 BUS

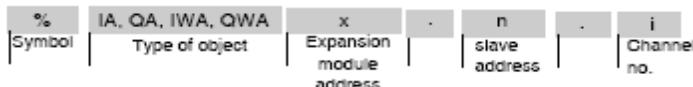
8.1.12.1 AT A GLANCE

This page presents the details relating to the addressing of discrete or analog I/Os of slave devices.

To avoid confusion with Remote I/Os, new symbols are available with an ASInterface syntax: %IA for example.

8.1.12.2 ILLUSTRATION

Reminder of the principles of addressing:



8.1.12.3 SPECIFIC VALUES

The table below gives specific values to AS-Interface V2 slave objects:

Part	Values	Comment
IA	-	Image of the physical discrete input of the slave.
QA	-	Image of the physical discrete 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	-

8.1.12.4 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	Discrete output 1 of slave 5B of the AS-Interface module situated in position 2 on the expansion bus.
%IA1.12A.2	Discrete input 2 of slave 12A of the AS-Interface module situated in position 1 on the expansion bus.

8.1.12.5 IMPLICIT EXCHANGES

The objects described below are exchanged implicitly, in other words they are exchanged automatically on each PLC cycle.

8.1.13 PROGRAMMING AND DIAGNOSTICS FOR THE AS-INTERFACE V2 BUS

8.1.13.1 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 ASInterface 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 AS-Interface window or the animation table.

8.1.13.2 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 AS-Interface V2 bus interface module operating mode., p. 197))
	2	system stopped (= 1 if the Offline (See Offline Mode, p. 197) 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.

8.1.13.3 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_CMDn %MWx: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.
l	Length of the instruction in number of words (2).

8.1.13.4 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).

%MWx	%MWx+1	Action
11	Result	Reads the list of detected slaves (LDS table) with addresses from 16B to 31B (1 bit per slave).
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.
32	Param	Writes a new parameter in an AS-Interface slave (PI table). Param: <ul style="list-style-type: none">● Byte 0: New parameter to write – 0 to 15● Byte 1: Address - 0 to 31 (for 0A to 31A) and 100 to 131 (for 0B to 31B)
33	Param	Reads a parameter on an AS-Interface slave (PI table). Param: <ul style="list-style-type: none">● Byte 0: New parameter to write – 0 to 15● Byte 1: Address - 0 to 31 (for 0A to 31A) and 100 to 131 (for 0B to 31B)

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.

8.1.13.5 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							

8.1.13.6 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.

8.1.13.7 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

8.1.14 AS-INTERFACE V2 BUS INTERFACE MODULE OPERATING MODE:

8.1.14.1 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. 193) instruction in a user program allows you to enter or exit these modes.

8.1.14.2 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 TwidoSuite. 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 ASInterface module into protected mode.

8.1.14.3 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. 196) instruction, which also allows you to exit the mode and return to protected mode.

8.1.14.4 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. 193) instruction.



9 INSTALLING AND CONFIGURING THE CANOPEN FIELDBUS

AT 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

9.1 CANopen Fieldbus Overview 132

9.2 Implementing the CANopen Bus 142

9.1 CANOPEN FIELDBUS OVERVIEW

9.1.1 AT A GLANCE

9.1.1.1 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.

9.1.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

CANopen Knowledge Base 132

About CANopen 134

CANOpen Boot-Up 136

Process Data Object (PDO) Transmission 138

Access to Data by Explicit Exchanges (SDO) 139

"Node Guarding" and "Life Guarding" 140

Internal Bus Management 142

9.1.2 CANOPEN KNOWLEDGE BASE

9.1.2.1 INTRODUCTION

The following explanations of technical terms and acronyms are helpful for understanding the basic knowledge of CANopen network communication.

9.1.2.2 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).

9.1.2.3 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.

9.1.2.4 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.

9.1.2.5 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.



9.1.3 ABOUT CANOPEN

9.1.3.1 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.

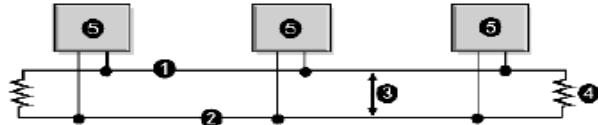
9.1.3.2 THE CANOPEN PROTOCOL

The CANopen protocol was created as a subset of CAL. 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.

9.1.3.3 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.

9.1.3.4 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:

- discrete and analog input/output modules (DS401),
- motors (DS402),
- control devices (DSP403),
- closed loop controllers (DSP404),
- PLCs (DS405),
- encoders (DS406).

9.1.3.5 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).

9.1.3.6 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).

9.1.3.7 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.

9.1.3.8 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/>)..

9.1.3.9 COMMUNICATION ON A CANOPEN NETWORK

The communication profile is based on CAL services and protocols.

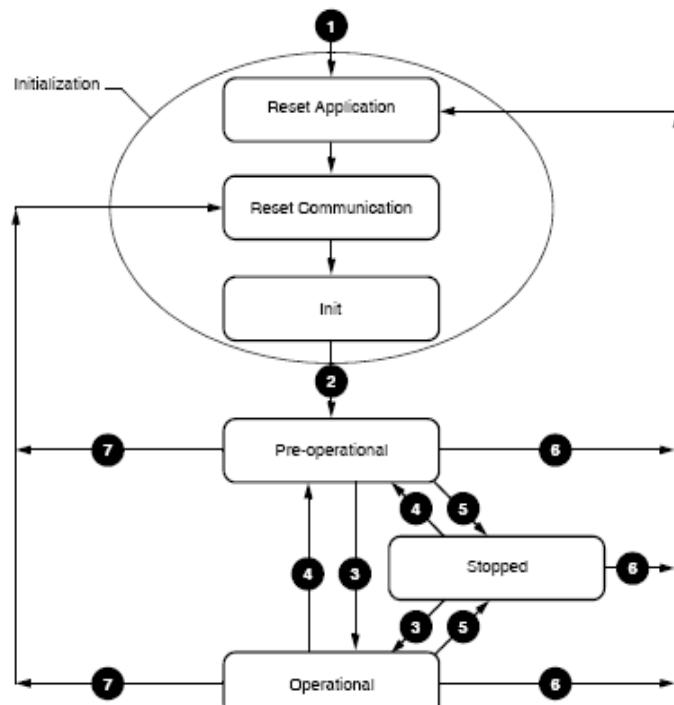
It provides the user with access to two types of exchange: SDO and PDO.

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.

9.1.4 CANOPEN BOOT-UP

9.1.4.1 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

9.1.4.2 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

9.1.4.3 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.

9.1.4.4 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 - 1FFFH) are saved in the object directory. The device then automatically goes into the "Init" state.

9.1.4.5 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.

9.1.4.6 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.

9.1.4.7 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.

9.1.4.8 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.

9.1.5 PROCESS DATA OBJECT (PDO) TRANSMISSION

9.1.5.1 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.

9.1.5.2 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).

9.1.5.3 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").

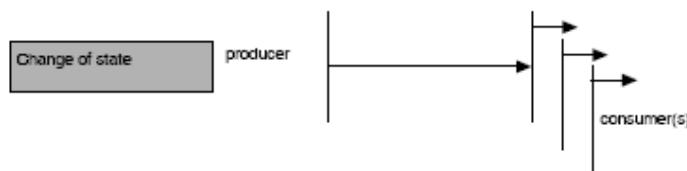
In contrast, an input is updated by the transmission of a RPDO by the master module which is then the "consumer".

9.1.5.4 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



9.1.5.5 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.

9.1.6 ACCESS TO DATA BY EXPLICIT EXCHANGES (SDO)

9.1.6.1 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.

9.1.6.2 TYPES OF SDO

There are two types of SDO:

- ▲ read SDOs (Download SDO),
- ▲ write SDOs (Upload SDO).

9.1.6.3 CLIENT/SERVER MODEL

The SDO protocol is based on a 'Client / Server' model.

For a Download SDO

The client sends a request indicating the object to be read.

The server return the data contained within the object.

For an Upload SDO

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.

For an unprocessed SDO

In both cases, if an SDO was not able to be processed, the server returns an error code (abort code).

9.1.7 "NODE GUARDING" AND "LIFE GUARDING"

9.1.7.1 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.

9.1.7.2 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.

9.1.7.3 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.

9.1.7.4 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.

9.1.7.5 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.

9.1.7.6 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.

9.1.7.7 "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

9.1.8 INTERNAL BUS MANAGEMENT

9.1.8.1 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.

9.1.8.2 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.

9.2 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 142](#)

[Hardware Setup 143](#)

[Configuration Methodology 144](#)

[Declaration of CANopen Master 145](#)

[Network CANopen Slave Declaration 145](#)

[CANopen Objects Mapping 149](#)

[CANopen Objects Linking 149](#)

CANopen Objects Symbolization 153

Addressing PDOs of the CANopen master 154

Programming and diagnostics for the CANopen fieldbus 155

9.2.1 OVERVIEW

9.2.1.1 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: • TWDLO*24DRF • TWDLOA*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	TwidoSuite

9.2.1.2 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.

9.2.2 HARDWARE SETUP

9.2.2.1 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 TwidoHW - Installing an expansion module). Follows these steps:



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

Step	Action	Description
1	Installation Preparation	Consult the <i>Twido Programmable Controllers Hardware Reference Guide (TWD USE 10AE)</i> for instructions on: <ul style="list-style-type: none">• 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 TwidoHW - 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 TwidoHW - 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.

9.2.3 CONFIGURATION METHODOLOGY

9.2.3.1 OVERVIEW

The CANopen configuration is performed via the CANopen Configuration tool available on TwidoSuite.

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.

9.2.3.2 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: <ul style="list-style-type: none">• 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: <ul style="list-style-type: none">• 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 TwidoSuite user interface program will not permit any other CANopen module to be added.

9.2.3.3 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.

9.2.4 DECLARATION OF CANOPEN MASTER

9.2.4.1 PROCEDURE

The table below shows the different steps when declaring the master CANopen.

Step	Action	Comment
1	Select Describe step from the TwidoSuite interface.	See .
2	Display the product catalog and select a TWDNCO1M module to add to the system description.	<p>See (See Positioning a Module, TwidoSuite Programming Software, Online Help).</p> <p>Note: A TWDNCO1M master can be inserted in any available expansion slot numbered 1 to 7 on the Twido bus.</p> <p>At this stage, you may continue adding any other expansion module (up to 7) that you want to include into your Twido system.</p> <p>Note: Only one TWDNCO1M CANopen master module is allowed.</p> <p>Only TWDC*A24DRF, TWDC*A40DRF, TWDLMDA20*** and TWDLMDA40*** controllers are supported</p>

9.2.5 NETWORK CANOPEN SLAVE DECLARATION

9.2.5.1 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.)

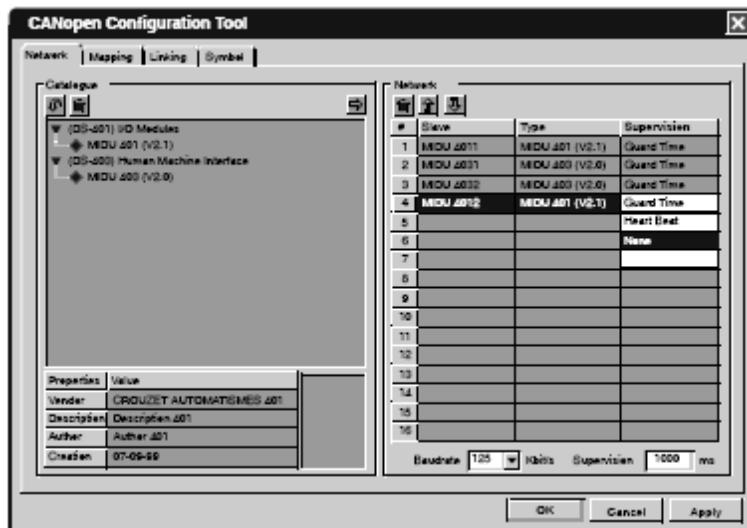
9.2.5.2 CANOPEN CONFIGURATOR

From the TwidoSuite user interface, select the Program → Configure → Configure the Hardware task and select the TWDNCO1M module in the PCE View (See PCE View, TwidoSuite Programming Software, Online Help).

Result: The CANopen Configuration Tool appears on screen in the DSO view (DSO View, TwidoSuite Programming Software, Online Help), as shown in the following sub-section.

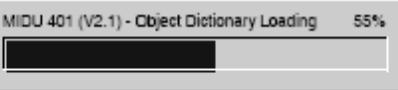
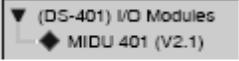
9.2.5.3 NETWORK DIALOGBOX

The network CANopen configuration and slave declaration is performed via the TwidosSoft CANopen Configurator Network dialogbox, as shown below:



9.2.5.4 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 TwidoSuite.
5	To display the device properties of a CANopen slave: 1. Click twice on the device type listed in the catalog. Example:  Result:  2. 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: <ul style="list-style-type: none">• 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.)
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 TwidoSuite project.

9.2.5.5 BUILDING THE CANOPEN NETWORK

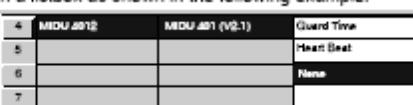
The table below describes how to declare slave devices on the Twido CANopen network. (Note that you may only declare slaves which EDS profiles have been prior added to or are already stored in the Catalog.)

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: <ul style="list-style-type: none">● 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 will 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 TwidoSuite project.

9.2.5.6 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: 125, 250 (default value), 500. 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.

Step	Action
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:
	
4	Select the error control protocol you wish to use to manage communications between the TWDNCO1M master module and the selected slave device: <ul style="list-style-type: none">● 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: <ul style="list-style-type: none">● 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 TwidoSuite "Reset" command (Online → Firmware / Reset),● a TwidoSuite load configuration command (Online → Download),● a command for firmware download to the TWDNCO1M master module via TwidoSuite (Online → Firmware Download).
6	Press the Apply button to confirm changes and save the network configuration to the TwidoSuite project.

9.2.6 CANOPEN OBJECTS MAPPING

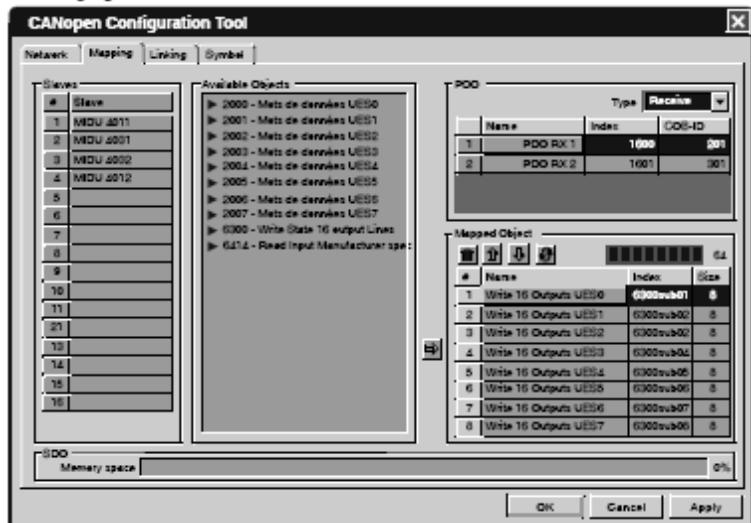
9.2.6.1 OVERVIEW

The Mapping dialogbox of the CANopen Configuration Tool allows you configure the PDOs of each slave device declared on the network.

9.2.6.2 MAPPING DIALOGBOX

Select the Mapping tab from the CANopen Configuration Tool (See CANopen Configurator, p. 221).

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



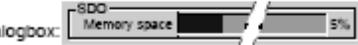
9.2.6.3 OBJECTS MAPPING

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. 221.)

Slaves	
#	Slave
1	MIDU 4011
2	MIDU 4081
3	MIDU 4082
4	MIDU 4012
e	

Step	Action																																													
3	<p>Results:</p> <ol style="list-style-type: none"> All of the CANopen objects supported by the selected slave are displayed in the Available Objects frame, as shown in the example below: <p>1. All of the CANopen objects supported by the selected slave are displayed in the Available Objects frame, as shown in the example below:</p> <p>Available Objects</p> <ul style="list-style-type: none"> > 2000 - Mots de données UES0 > 2001 - Mots de données UES1 > 2002 - Mots de données UES2 > 2003 - Mots de données UES3 > 2004 - Mots de données UES4 > 2005 - Mots de données UES5 > 2006 - Mots de données UES6 > 2007 - Mots de données UES7 > 6300 - Write State 16 output Lines > 6414 - Read Input Manufacturer spec <ol style="list-style-type: none"> 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: <p>2. 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:</p> <p>PDO</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Index</th> <th>COB-ID</th> </tr> </thead> <tbody> <tr> <td>1 PDO TX 1</td> <td>1A00</td> <td>1B1</td> </tr> <tr> <td>2 PDO TX 2</td> <td>1A01</td> <td>2B1</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Name</th> <th>Index</th> <th>COB-ID</th> </tr> </thead> <tbody> <tr> <td>1 PDO RX 1</td> <td>1B00</td> <td>2B1</td> </tr> <tr> <td>2 PDO RX 2</td> <td>1B01</td> <td>3B1</td> </tr> </tbody> </table> <ol style="list-style-type: none"> The predefined mapping of each selected PDO is showing in the Mapped Objects frame as well. : <p>Mapped Object</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Index</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>1 Write 16 Outputs UES0</td> <td>6300va001</td> <td>8</td> </tr> <tr> <td>2 Write 16 Outputs UES1</td> <td>6300va002</td> <td>8</td> </tr> <tr> <td>3 Write 16 Outputs UES2</td> <td>6300va003</td> <td>8</td> </tr> <tr> <td>4 Write 16 Outputs UES3</td> <td>6300va004</td> <td>8</td> </tr> <tr> <td>5 Write 16 Outputs UES4</td> <td>6300va005</td> <td>8</td> </tr> <tr> <td>6 Write 16 Outputs UES5</td> <td>6300va006</td> <td>8</td> </tr> <tr> <td>7 Write 16 Outputs UES6</td> <td>6300va007</td> <td>8</td> </tr> <tr> <td>8 Write 16 Outputs UES7</td> <td>6300va008</td> <td>8</td> </tr> </tbody> </table>	Name	Index	COB-ID	1 PDO TX 1	1A00	1B1	2 PDO TX 2	1A01	2B1	Name	Index	COB-ID	1 PDO RX 1	1B00	2B1	2 PDO RX 2	1B01	3B1	Name	Index	Size	1 Write 16 Outputs UES0	6300va001	8	2 Write 16 Outputs UES1	6300va002	8	3 Write 16 Outputs UES2	6300va003	8	4 Write 16 Outputs UES3	6300va004	8	5 Write 16 Outputs UES4	6300va005	8	6 Write 16 Outputs UES5	6300va006	8	7 Write 16 Outputs UES6	6300va007	8	8 Write 16 Outputs UES7	6300va008	8
Name	Index	COB-ID																																												
1 PDO TX 1	1A00	1B1																																												
2 PDO TX 2	1A01	2B1																																												
Name	Index	COB-ID																																												
1 PDO RX 1	1B00	2B1																																												
2 PDO RX 2	1B01	3B1																																												
Name	Index	Size																																												
1 Write 16 Outputs UES0	6300va001	8																																												
2 Write 16 Outputs UES1	6300va002	8																																												
3 Write 16 Outputs UES2	6300va003	8																																												
4 Write 16 Outputs UES3	6300va004	8																																												
5 Write 16 Outputs UES4	6300va005	8																																												
6 Write 16 Outputs UES5	6300va006	8																																												
7 Write 16 Outputs UES6	6300va007	8																																												
8 Write 16 Outputs UES7	6300va008	8																																												

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 TwidoSuite project.
10	Repeat steps 5 through 9 for each PDO mapping you wish to configure.
11	Notes on memory usage: <ul style="list-style-type: none"> 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:  .

9.2.7 CANOPEN OBJECTS LINKING

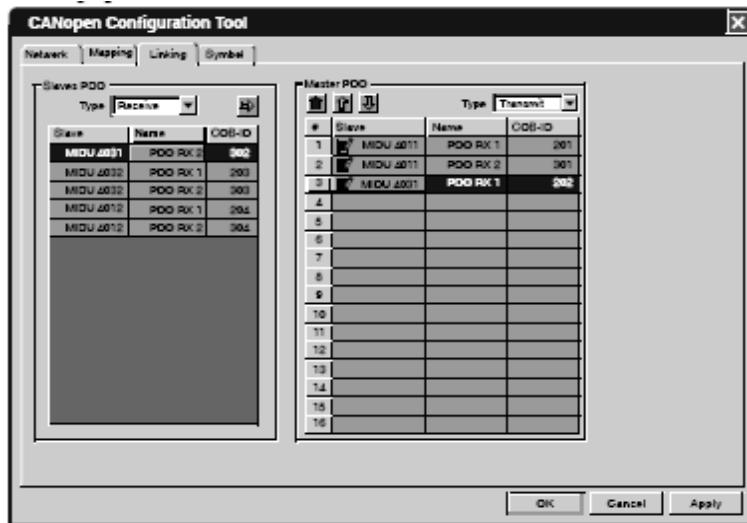
9.2.7.1 OVERVIEW

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 TWDNC01M CANopen master module PDOs.

9.2.7.2 LINKING DIALOGBOX

Select the Linking tab from the CANopen Configuration Tool (See CANopen Configurator, p. 221).

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



9.2.7.3 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: 
	Note: Selecting Receive or Transmit in the Slave PDOs frame automatically toggles the Master PDOs to the opposite type: Transmit or Receive, respectively.
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 TwidoSuite project.
6	Repeat steps 1 through 5 for each slave PDO you wish to link to the CANopen master.

9.2.8 CANOPEN OBJECTS SYMBOLIZATION

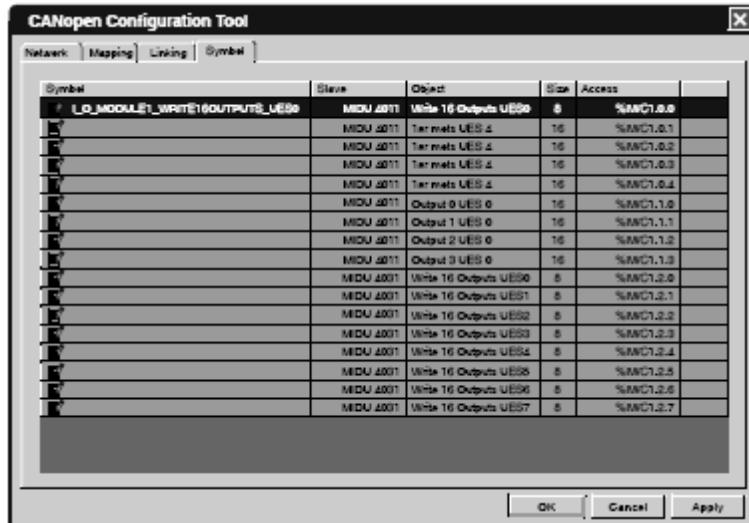
9.2.8.1 OVERVIEW

The Symbol dialogbox allows you to define a symbolization of the variables associated with the CANopen master.

9.2.8.2 SYMBOL DIALOGBOX

Select the Symbol tab from the CANopen Configuration Tool (See CANopen Configurator, p. 221).

Result: The CANopen Configuration Tool appears on screen, as shown in the following figure:



9.2.8.3 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. 51.
3	Press the Apply button to confirm changes to the symbols table and save to the TwidoSuite project.
4	Repeat steps 1 to 3 for each variable you wish to symbolize.

9.2.9 ADDRESSING PDOS OF THE CANOPEN MASTER

9.2.9.1 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: %IWC for example.

9.2.9.2 ILLUSTRATION

Reminder of the addressing principles:



9.2.9.3 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.)

9.2.9.4 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.

9.2.9.5 IMPLICIT EXCHANGES

The objects described below are exchanged implicitly, in other words they are exchanged automatically on each PLC cycle.

9.2.10 PROGRAMMING AND DIAGNOSTICS FOR THE CANOPEN FIELDBUS

9.2.10.1 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.

9.2.10.2 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.

9.2.10.3 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: <ul style="list-style-type: none"> • = 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): <ul style="list-style-type: none"> - "error free". • = 3 => Node State Operational (module is in operational state): <ul style="list-style-type: none"> - "not error free". • = 4 => Node State Preoperational (module is in preoperational state): <ul style="list-style-type: none"> - 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): <ul style="list-style-type: none"> - 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]	
			<ul style="list-style-type: none"> • = 6 => Node State Preoperational (module is in preoperational state): <ul style="list-style-type: none"> - expected modules only (those declared as expected in the configuration table); - module is present but its current state does not allow to set it to operational; - "error free". • = 7 => Node State Preoperational (module is in preoperational state): <ul style="list-style-type: none"> - expected modules only (those declared as expected in the configuration table); - module is present but its current state does not allow to set it to operational; - "not error free". • = 8 => Wrong module (a module was detected with different device identity information); <ul style="list-style-type: none"> - "error free". • = 9 => Wrong module (a module was detected with different device identity information); <ul style="list-style-type: none"> - "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); <ul style="list-style-type: none"> - "error free". • = 11 => Slave configuration error; <ul style="list-style-type: none"> - "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); <ul style="list-style-type: none"> - "error free". • = 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); <ul style="list-style-type: none"> - "not error free". • = 14 => Unexpected module (a module was detected that is not in the configuration table); <ul style="list-style-type: none"> - "error free". • = 15 => Unexpected module (a module was detected that is not in the configuration table); <ul style="list-style-type: none"> - "not error free".

9.2.10.4 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_CMDn %MWx: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.
l	Length of the instruction in number of words (2 or 6).

9.2.10.5 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]														
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			Index	Len	Sub						3 => Start Read SDO command. 4 => Start Write SDO command.					
	Node										Node = 1-16 => Node address					
						Data 1		Data 2		Payload according to the length field (Len) value	PDO object index.					
										Payload according to the length field (Len) value	Sub = 0-255 => Object sub-index Len = Length of data in byte					

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.

9.2.10.6 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
```

10 CONFIGURING THE TWIDOPORT ETHERNET GATEWAY

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

10.1 Normal Configuration and Connection of TwidoPort 160

10.2 TwidoPort's Telnet Configuration 165

10.3 Communication Features 174

10.1 NORMAL CONFIGURATION AND CONNECTION OF TWIDOPORT

10.1.1 AT A GLANCE

10.1.1.1 SUBJECT OF THIS SECTION

This section provides information on how to perform a normal configuration of the ConneXium TwidoPort module with the TwidoSuite application program, module connectivity and BootP configuration information, as well.

10.1.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Normal Configuration with TwidoSuite 160

BootP Configuration 164

10.1.2 NORMAL CONFIGURATION WITH TWIDOSUITE



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

10.1.2.1 FOREWORD

Configure TwidoPort with these instructions:

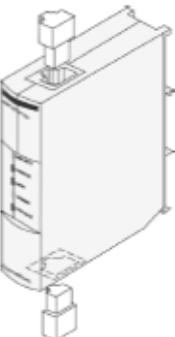
Note: Plug 'n play feature

When TwidoPort is configured with TwidoSuite, 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 TwidoSuite and upgrade the Twido firmware to 3.4 or higher.

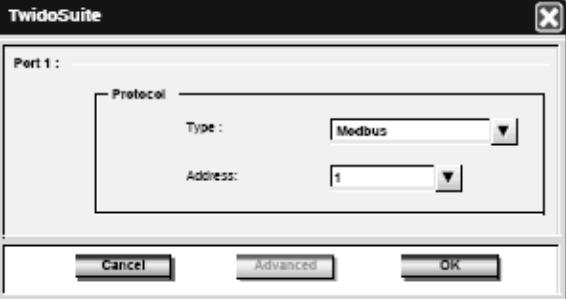
10.1.2.2 INSTALLING THE 499TWD01100 TWIDOPORT MODULE

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: <ul style="list-style-type: none"> • 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 How to Install the TwidoPort Ethernet Interface Module.
3	Protective Earth (PE) Grounding	Attach a grounded wire to the M3 screw terminal on the bottom of TwidoPort.
4	Serial and Ethernet Connections  Top plug: from Twido (serial) Bottom plug: from Ethernet, either a straight or crossover cable	<p>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.</p> <p>Connect the RJ-45 plug from a standard Ethernet network cable (not supplied) into TwidoPort's Ethernet port.</p>

10.1.2.3 DECLARING THE 499TWD01100 TWIDOPORT MODULE

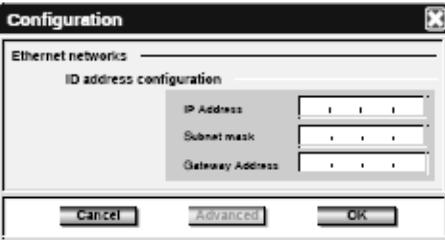
The table below shows the different stages when declaring the 499TWD01100 TwidoPort module.

Step	Action	Comment
1	Select Port 1 (or Port 2 if installed) to configure in the Describe window.	See (See Configuring an Object, TwidoSuite Programming Software, Online Help).
2	Configure the Feature dialog box (See Configuring an Object, TwidoSuite Programming Software, Online Help) that appears, as explained in the following steps.	
3	Select Modbus in the Protocol Type box.	
4	Select Describe step from the TwidoSuite interface.	See .
5	Display the product catalog and select and add a 499TWD01100 module to the system description.	See (See Positioning a Module, TwidoSuite Programming Software, Online Help). At this stage, you may continue adding any other optional module supported by your Twido controller. Note: Only one 499TWD01100 TwidoPort module is allowed.
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.	

10.1.2.4 CONFIGURING THE 499TWD01100 TWIDOPORT MODULE

Note: The Ethernet parameters of TwidoPort can be configured when the TwidoSuite 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 (See IP Addressing, TwidoSuite Programming Software, Online Help) and (See Assigning IP Addresses, TwidoSuite Programming Software, Online Help).	
1	Select the 499TWD01100 TwidoPort module to configure TwidoPort's IP parameters, see (See Configuring an Object, TwidoSuite Programming Software, Online Help). Result: The TwidoPort Configuration window appears on screen, as shown in the following sub-section.	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 notes 1 and 2.)	Caution: For good device communication, the IP addresses of the PC running the TwidoSuite 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 notes 1 and 3.)	Caution: For good device communication, the subnet mask configured on the PC running the TwidoSuite application and TwidoPort's subnet mask must match. As default, the TwidoSuite 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 notes 1 and 4.)	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 transfer it to the Twido controller.	
6	Power off the Twido controller, then power on again.	
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.	

10.1.2.5 SETTING UP ETHERNET CONNECTION IN TWIDOSUITE

To allow the PC running TwidoSuite and the Twido controller to communicate over the Ethernet network.

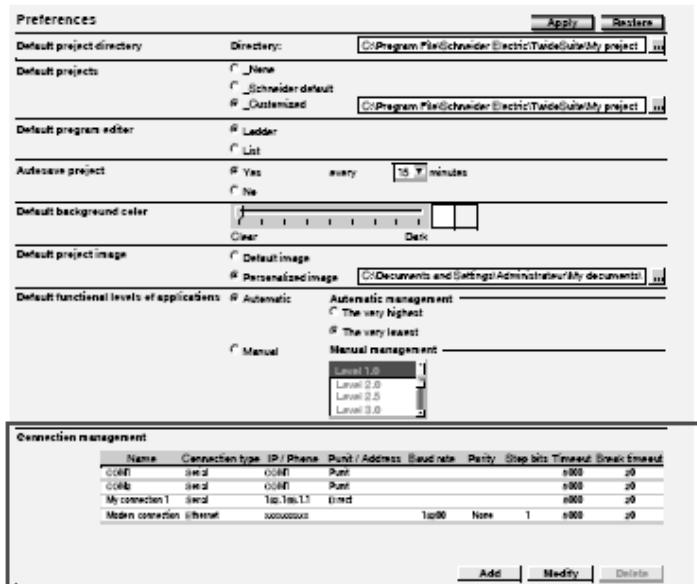
Select



Preferences

Result:

The following Connections Management dialog box appears:



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: <ul style="list-style-type: none"> • 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.

Step	Action
3	In the Connection Type field, click to unfold the dropdown list that includes: Serial, Ethernet, and USB (if any). Select Ethernet 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 controller you wish to connect to. IP Address: Enter the static IP address that you have previously specified for your Twido controller.
5	The Punit / Address field can be filled in when IP / Phone has been selected. For a Ethernet connection, the default Punit/Address value is Direct. This default must be changed to 1 (or to whatever address you have previously used for the controller). For a Serial Type connection, default value is Punit. When any of those is selected, the 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 transferred. (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.)
6	Use the default settings in Timeout and Break timeout fields, unless you have specific timeout needs. (For more details, please refer to .)
7	Click the OK button to save the new connection settings and close the Connections management dialog box. Result: All newly-added connections are appended in the Preferences → Connections Management table and in the Program → Debug → Connect → Select a connection table.

10.1.3 BOOTP CONFIGURATION

10.1.3.1 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]
----	----	--------	--------

10.1.3.2 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.

10.2 TWIDOPORT'S TELNET CONFIGURATION

AT A GLANCE

SUBJECT OF THIS SECTION

This section describes how to configure the ConneXium TwidoPort module with a Telnet session.

WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Introducing Telnet Configuration 165

Telnet Main Menu 165

IP/Ethernet Settings 166

Serial Parameter Configuration 167

Configuring the Gateway 167

Security Configuration 168

Ethernet Statistics 170

Serial Statistics 171

Saving the Configuration 172

Restoring Default Settings 172

Upgrading the TwidoPort Firmware 172

Forget Your Password and/or IP Configuration? 174

10.2.1 INTRODUCING TELNET CONFIGURATION

10.2.1.1 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).

10.2.1.2 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
```

10.2.2 TELNET MAIN MENU

10.2.2.1 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:

Telmeccanique 499 TWD 01 100 Configuration and Diagnostics
 © 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:

```

10.2.3 IP/ETHERNET SETTINGS

10.2.3.1 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. 254).
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: <ul style="list-style-type: none"> • 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.)

10.2.3.2 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.

10.2.3.3 EXAMPLE OF ETHERNET SETTINGS

The following figure shows an example of TwidoPort's Ethernet settings:

```
Telemecanique 499 TWD 01 100 Configuration and Diagnose
(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>eturn to Main Menu
Select Command or Parameter<1..N> to change:
```

10.2.4 SERIAL PARAMETER CONFIGURATION

10.2.4.1 FOREWORD

10.2.4.2 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. 254).
2	Select (type) a to change the serial settings.	See the following figure.
3	Verify or reset the settings.	Other parameters include: <ul style="list-style-type: none"> • Baud Rate • Data Bits • Parity • Stop Bits • Protocol
4	Select R and press Enter.	The Telnet main menu appears. (You may have to press Enter again to update the screen.)

10.2.4.3 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: RIU

Commands: R>eturn to Main Menu
Select Command or Parameter<1..N> to change:
```

10.2.5 CONFIGURING THE GATEWAY

10.2.5.1 FOREWORD

Note: Usually, it is not necessary to configure TwidoPort's gateway parameters.

10.2.5.2 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. 254).	
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 Enter .	The Telnet main menu appears. (You may have to press Enter again to update the screen.)	
Note	Twido does not support any broadcast Modbus messages.		

10.2.5.3 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>turn to Main Menu
Select Command or Parameter<1..4> to change: _
```

10.2.6 SECURITY CONFIGURATION

10.2.6.1 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. 254).
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 note below.)
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:	
	<ul style="list-style-type: none"> • minimum length: 4 characters • maximum length: 10 characters • allowed characters: a to z, A to Z (no spaces) 	

10.2.7 ETHERNET STATISTICS

10.2.7.1 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. 254).
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.)

10.2.7.2 THE ETHERNET MODULE STATISTICS SCREEN

TwidoPort's Ethernet Module Statistics screen:

```

Telemecanique 499 IWD 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, G>clear Statistics, R>turn to Main Menu

```

10.2.8 SERIAL STATISTICS

10.2.8.1 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. 254).
2	Select (type) s 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.)

10.2.8.2 THE SERIAL STATISTICS SCREEN

TwidoPort's Serial Statistics screen:

```

Telenecanique 499 IVD 01 100 Configuration and Diagnostics
(c) 2004 Schneider Automation Inc
----- SERIAL STATISTICS -----
Serial Bus Statistics
  Bus Message Count: 8284
  Bus Conn. 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

```

10.2.9 SAVING THE CONFIGURATION

10.2.9.1 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. 254).
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 *Security Configuration*, p. 258.

10.2.9.2 THE SAVE CONFIGURATION CONFIRMATION SCREEN

TwidoPort's Save Configuration confirmation screen:

```

Telemecanique 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.

```

10.2.10 RESTORING DEFAULT SETTINGS

10.2.10.1 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 Telnet Main Menu, p. 254).
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 Saving the Configuration, p. 261), above.

10.2.10.2 THE DEFAULT CONFIGURATION SCREEN

TwidoPort's Default Configuration screen:

```

Telemecanique 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...

```

10.2.11 UPGRADING THE TWIDOPORT FIRMWARE

10.2.11.1 FOREWORD

Note:

Copyright:  2012 by Géza HUSI, Péter SZEMES, István BARTHA

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.

10.2.11.2 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</i> , p. 254).
2	Select (type) r to initiate the firmware upgrade.	Five seconds after selecting r (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 192.168.44.113
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 . <i>(See notes 1 and 2.)</i>	A message indicates that the ftp was successful. <i>(See note 3.)</i>
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.		

10.2.11.3 FIRMWARE UPGRADE IN PROGRESS

The following figure shows a typical Firmware Upgrade In-Progress screen:

```
Telnetcanique 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.
```

10.2.11.4 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:

```

Telenecanique 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

```

10.2.12 FORGET YOUR PASSWORD AND/OR IP CONFIGURATION?

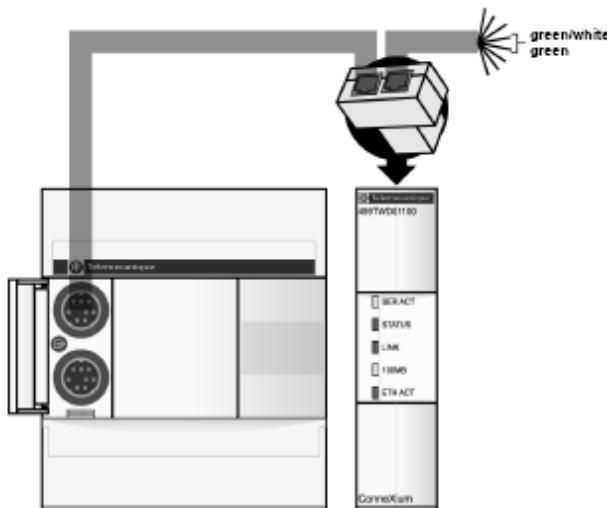
10.2.12.1 CONNECTING IN BACKUP MODE

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. (See the following illustration.)
2	Connect via ftp to TwidoPort. (See note.)	TwidoPort uses the following default IP configuration: <ul style="list-style-type: none"> • 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.	

10.2.12.2 FTP CONNECTION

The following illustration shows how to connect to TwidoPort via ftp in backup mode:



10.3 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 175

Modbus/TCP Communications Protocol 175

Locally Supported Modbus Function Codes 176

10.3.1 ETHERNET FEATURES

10.3.1.1 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.

10.3.1.2 ETHERNET FEATURES

TwidoPort supports the following Ethernet features:

- ▲ Auto-negotiation

TwidoPort supports 10/100TX auto-negotiation. It communicates only in halfduplex 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 straightthrough or crossover cables.

10.3.2 MODBUS/TCP COMMUNICATIONS PROTOCOL

10.3.2.1 ABOUT MODBUS



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

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.

10.3.2.2 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.

10.3.2.3 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.

10.3.3 LOCALLY SUPPORTED MODBUS FUNCTION CODES

10.3.3.1 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 note 1.)
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.

11 OPERATOR DISPLAY OPERATION

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 177

Controller Identification and State Information 179

System Objects and Variables 181

Serial Port Settings 186

Time of Day Clock 187

Real-Time Correction Factor 188

11.1.1 OPERATOR DISPLAY

11.1.1.1 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.

11.1.1.2 DISPLAYS AND FUNCTIONS



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

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

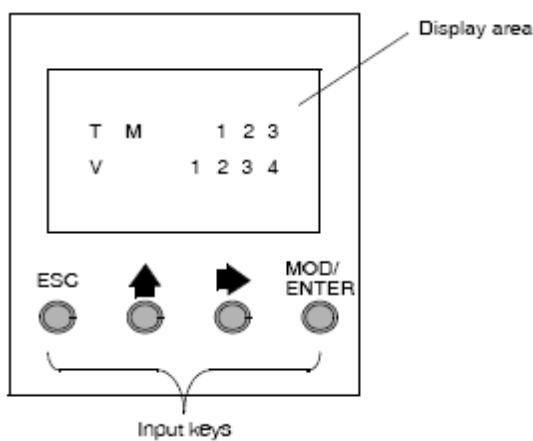
Display and modify the RTC Correction value for the optional RTC.

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.

11.1.1.3 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.



11.1.1.4 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.

11.1.1.5 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.

11.1.1.6 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.

11.1.2 CONTROLLER IDENTIFICATION AND STATE INFORMATION

11.1.2.1 INTRODUCTION

The initial display or screen of the Twido optional Operator Display shows the Controller Identification and State Information

11.1.2.2 EXAMPLE



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

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:



11.1.2.3 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. 564).

- ▲ 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.

11.1.2.4 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.

11.1.3 SYSTEM OBJECTS AND VARIABLES

11.1.3.1 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).

11.1.3.2 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	%Ox.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
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	%ONWx.z	Value	Read/Write
Grafoet	%Xx	Step Bit	Read

Object	Variable/Attribute	Description	Access
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	%OAx.y.z	Value	Read/Write/Force
AS-Interface analog slave output	%QWAx.y.z	Value	Read/Write
CANopen slave PDO input	%IWGx.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.

11.1.3.3 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.

11.1.3.4 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

11.1.3.5 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	

11.1.3.6 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	

11.1.3.7 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	

11.1.3.8 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

11.1.3.9 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

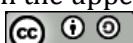
In the following example, memory word number 67 contains the value +123.

M	W	6	7
+		1	2 3

11.1.3.10 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

Copyright:  2012 by Géza HUSI, Péter SZEMES, István BARTHA

- ▲ 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	

11.1.3.11 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

11.1.3.12 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	

11.1.4 SERIAL PORT SETTINGS

11.1.4.1 INTRODUCTION

The operator display allows you to display the protocol settings and change the addresses of all serial ports configured using TwidoSuite. 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

11.1.4.2 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 TwidoSuite as equal. TwidoSuite will require that you do a download to become equal again.

11.1.5 TIME OF DAY CLOCK

11.1.5.1 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.

M	A	R	2	8
1	4	h	2	2

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.

11.1.5.2 DISPLAYING AND MODIFYING TIME OF DAY CLOCK

To display and modify the Time of Day Clock:

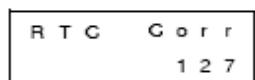
Step	Action
1	Press the key until the Timer/Date Display is shown. The month value ("JAN", "FEB") will be displayed in the upper-left corner of the display area. The value "RTC" will be displayed in the upper-left corner if no month has been initialized.
2	Press the MOD/ENTER key to enter the edit mode.
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.

11.1.6 REAL-TIME CORRECTION FACTOR

11.1.6.1 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.



11.1.6.2 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.

12. PLC PROGRAMMING LANGUAGES

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:

12.1 Ladder Language 189

12.2 Instruction Set Language 206

12.3 Grafcet 212

12.1 LADDER LANGUAGE

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 189

Programming Principles for Ladder Diagrams 191

Ladder Diagram Blocks 193

Ladder Language Graphic Elements 195

Special Ladder Instructions OPEN and SHORT 197

Programming Advice 198

Ladder/List Reversibility 198

Guidelines for Ladder/List Reversibility 201

Program Documentation 202

12.1.1 INTRODUCTION TO LADDER DIAGRAMS

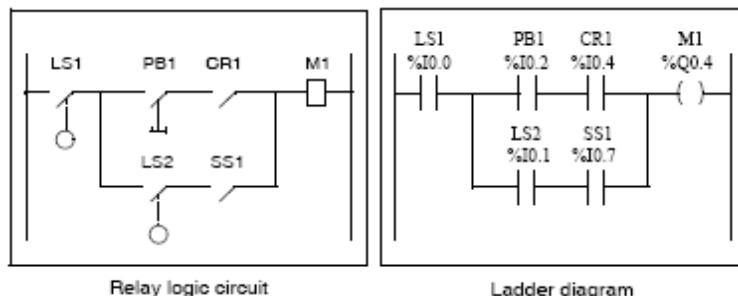
12.1.1.1 INTRODUCTION

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
().
- ▲ All outputs are represented by coil symbols
().
- ▲ Numerical operations are included in the graphical Ladder instruction set.

12.1.1.2 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.

12.1.1.2 LADDER RUNGS

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:

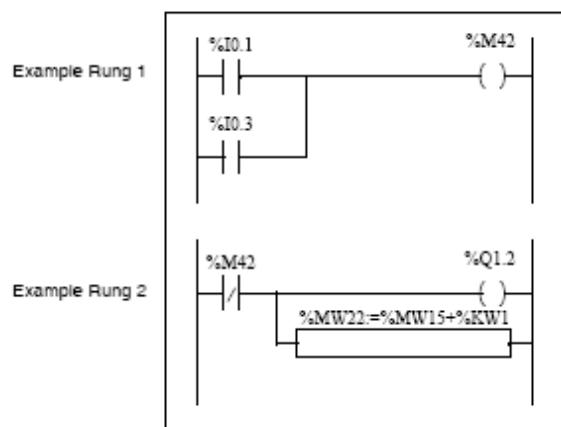
- ▲ Inputs/outputs of the controller (push buttons, sensors, relays, pilot lights, etc.)

- ▲ 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.

12.1.1.3 EXAMPLE OF LADDER RUNGS

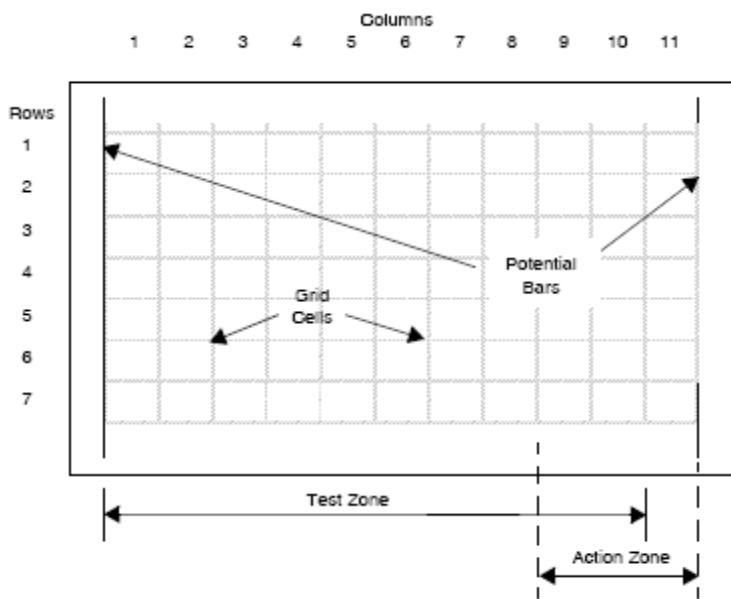
The following diagram is an example of a Ladder program composed of two rungs.



12.1.2 PROGRAMMING PRINCIPLES FOR LADDER DIAGRAMS

12.1.2.1 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.



12.1.2.2 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.

12.1.2.3 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.

12.1.2.4 SECTIONS/ SUBROUTINES

Each section/subroutine consists of:

- ▲ A section header with with section number (automatically assigned by the program), section/subroutine label, a user-defined section/subroutine title and four lines of user comments. See .
- ▲ A sequence of rungs below the section/subroutine header.

12.1.2.5 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)
- ▲ Rung title
- ▲ Rung comments

For more details about using the rung header to document your programs, see Program Documentation, p. 311.

12.1.3 LADDER DIAGRAM BLOCKS

12.1.3.1 INTRODUCTION

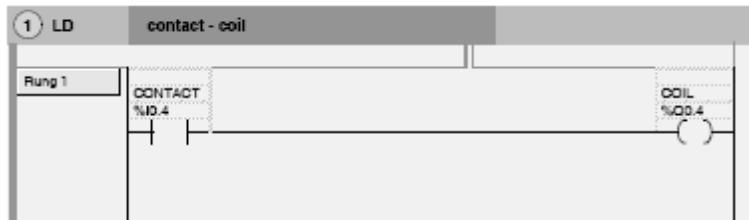
Ladder diagrams consist of blocks representing program flow and functions such as the following:

- ▲ Contacts
- ▲ Coils
- ▲ Program flow instructions
- ▲ Function blocks
- ▲ Comparison blocks
- ▲ Operation blocks

12.1.3.2 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 operation blocks occupy multiple cells.

The following are examples of a contact and a coil.



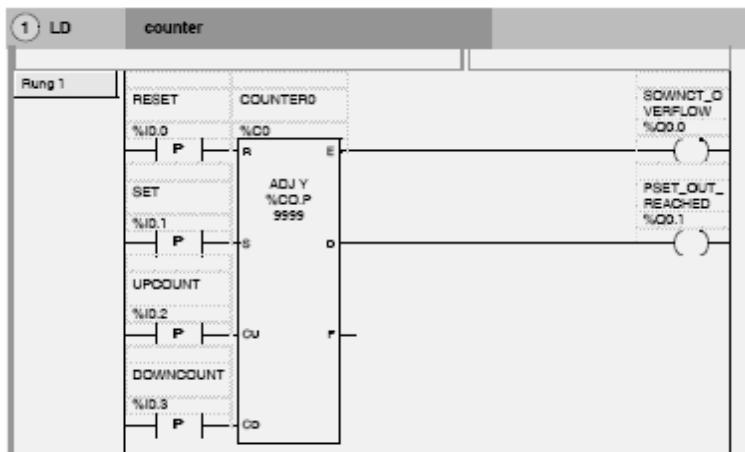
12.1.3.3 FUNCTION BLOCKS

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.

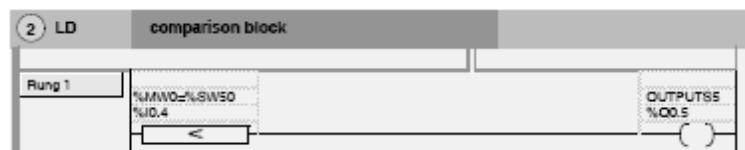


12.1.3.4 COMPARISON BLOCKS

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.



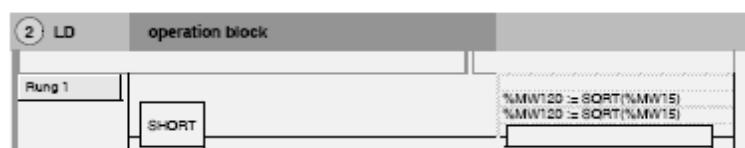
To edit the comparison block, click on the field just above the comparison box and type in your expression. Both symbols and addresses may be used as operands here. The comment field however is disabled.

12.1.3.5 OPERATION BLOCKS

Operation 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.

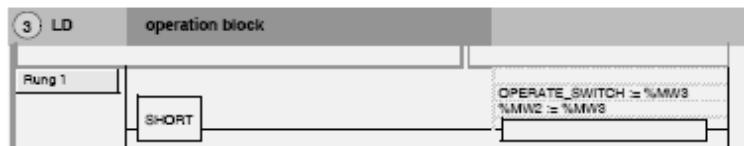
Operation blocks are horizontally oriented and occupy four columns by one row of the programming grid.

The following is an example of an operation block.



To edit the operation block, click on the field just above the operation box and type in your expression. Both symbols and addresses may be used as operands here. The expression will be displayed twice as shown in the above example. The comment field (top box), however, is disabled.

If symbols have been previously defined, the expression will be displayed with both addresses (lower box) and symbols (upper box) as shown in the following example of an operation block. Here, the address %MW2 has been previously defined with the symbol OPERATE_SWITCH



12.1.4 LADDER LANGUAGE GRAPHIC ELEMENTS

12.1.4.1 INTRODUCTION

Instructions in Ladder diagrams consist of graphic elements.

12.1.4.2 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	+P	LDR	Rising edge: detecting the change from 0 to 1 of the controlling bit object.
Contact for detecting a falling edge	-N	LOF	Falling edge: detecting the change from 1 to 0 of the controlling bit object.

12.1.4.3 LINK ELEMENTS

The graphic link elements are used to insert/delete ladder loops:

Name	Graphic element	Function
Insert a link		Insert an empty ladder loop
Delete a link		Delete an empty ladder loop Note: 1. If the ladder loop contains any elements, you must first delete all ladder elements before you can delete the ladder loop. 2. The keyboard shortcut used to remove an element is DELETE.

12.1.4.4 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		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	Placed at the end of subroutines to return to the main program.
Stop program		END	Defines the end of the program.

12.1.4.5 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).

12.1.4.6 OPERATE AND COMPARISON BLOCKS

Comparison blocks are programmed in the test zone, and operation 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

12.1.5 SPECIAL LADDER INSTRUCTIONS OPEN AND SHORT

12.1.5.1 INTRODUCTION

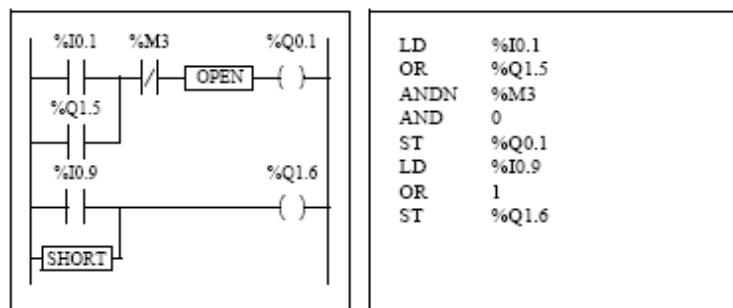
The OPEN and SHORT instructions provide a convenient method for debugging and troubleshooting Ladder programs. These special instructions alter the logic of a rung by either shorting or opening the continuity of a rung as explained in the following table.

Instruction	Description	List Instruction
OPEN	At the beginning of the rung.	LD 0
	Within a rung: Creates a break in the continuity of a ladder rung regardless of the results of the last logical operation.	AND 0
SHORT	At the beginning of the rung.	LD 1
	Within a rung: Allows the continuity to pass through the rung regardless of the results of the last logical operation.	OR 1

In List programming, the LD, OR and AND instructions are used to create the OPEN and SHORT instructions using immediate values of 0 and 1 respectively.

12.1.5.2 EXAMPLES

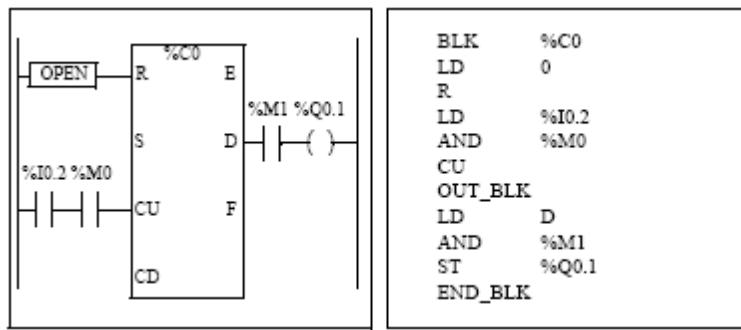
The following are examples of using the OPEN and SHORT instructions.



```

LD  %I0.1
OR  %Q1.5
ANDN %M3
AND  0
ST   %Q0.1
LD   %I0.9
OR   1
ST   %Q1.6
  
```

When no RESET input is required, as you cannot delete the link segment to the function block R-input, use the OPEN element to permanently break the input segment.



12.1.6 PROGRAMMING ADVICE

12.1.6.1 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.).

12.1.6.2 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.

12.1.6.3 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.

12.1.6.4 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).

12.1.6.5 TIME AND SCHEDULE BLOCK MANAGEMENT

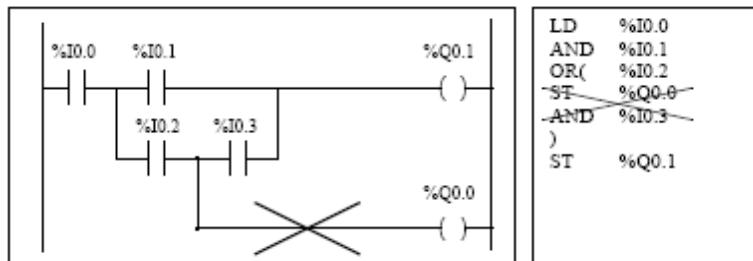
The state of system bit %S51, which indicates any RTC faults, should be checked.

12.1.6.6 SYNTAX AND ERROR CHECKING

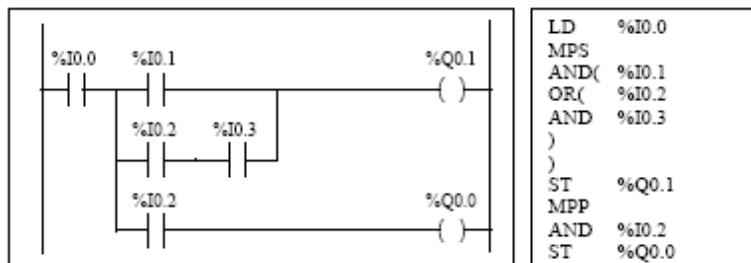
When a program is entered, TwidoSuite checks the syntax of the instructions, the operands, and their association.

12.1.6.7 ADDITIONAL NOTES ON USING PARENTHESES

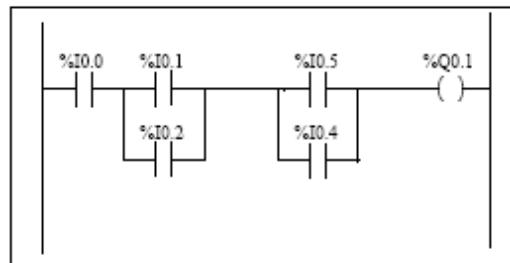
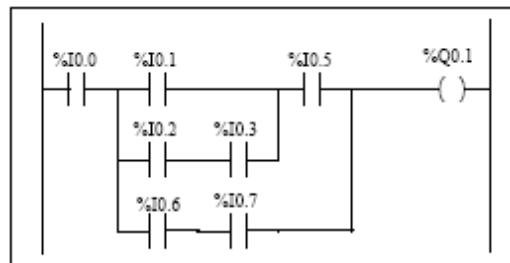
Assignment operations should not be placed within parentheses:



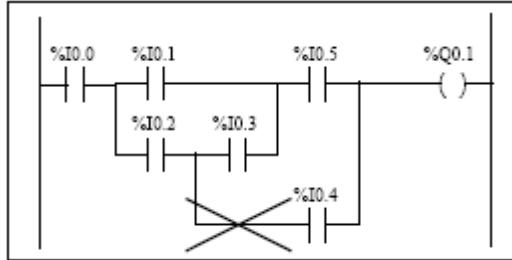
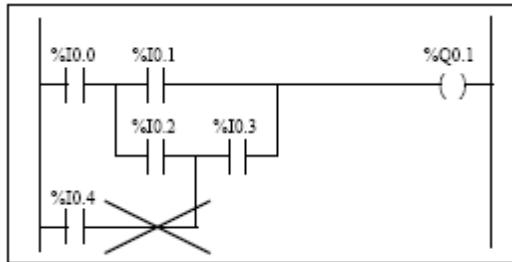
In order to perform the same function, the following equations must be programmed:



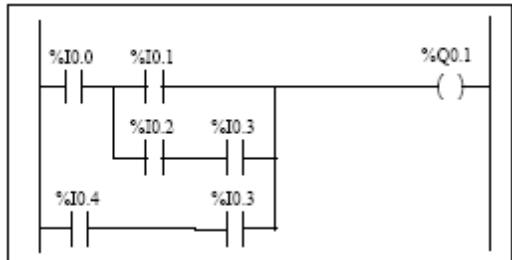
If several contacts are parallelized, they must be nested within each other or completely separate:



The following schematics cannot be programmed:



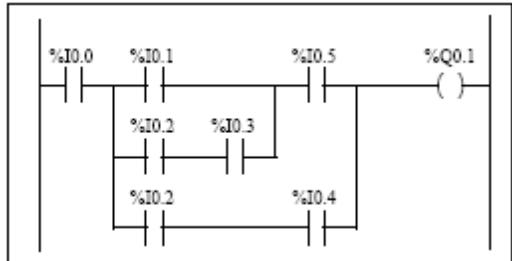
In order to execute schematics equivalent to those, they must be modified as follows:



```

LD  %IO.0
AND( %IO.1
OR( %IO.2
AND %IO.3
)
)
OR( %IO.4
AND %IO.3
)
ST  %Q0.1

```



```

LD  %IO.0
AND( %IO.1
OR( %IO.2
AND %IO.3
)
)
AND %IO.5
OR( %IO.2
AND %IO.4
)
)
ST  %Q0.1

```

12.1.7 LADDER/LIST REVERSIBILITY

12.1.7.1 INTRODUCTION

Program reversibility is a feature of the TwidoSuite programming software that provides conversion of application programs from Ladder to List and from List back to Ladder.

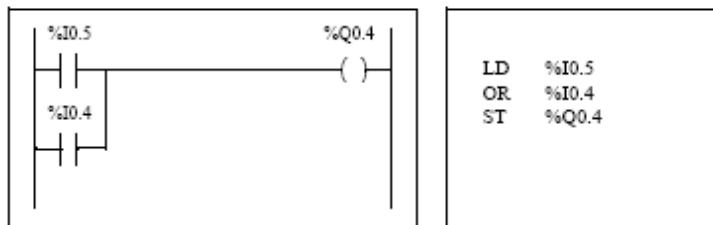
Use TwidoSuite to set the default display of programs: either List or Ladder format (by setting user preferences). TwidoSuite can also be used to toggle List and Ladder views.

12.1.7.2 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. TwidoSuite 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.

12.1.7.3 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. 309.

12.1.8 GUIDELINES FOR LADDER/LIST REVERSIBILITY

12.1.8.1 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. 358.

12.1.8.2 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.

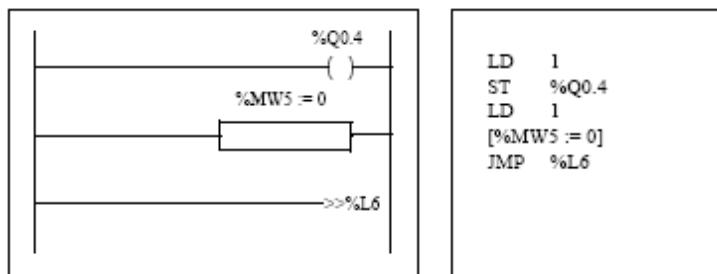
The following table identifies all List programming instructions that will not reverse to Ladder.

List Instruction	Operand	Description
JMPCN	%Li	Jump Conditional Not
N	none	Negation (Not)
ENDCN	none	End Conditional Not

12.1.8.3 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 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.

12.1.8.4 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.

12.1.9 PROGRAM DOCUMENTATION

12.1.9.1 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 TwidoSuite programming software uses these comments for reversibility. When reversing a program from List to Ladder, TwidoSuite uses some of the List comments to construct a rung header. For this, the comments inserted between List sequences are used for rung headers.

12.1.9.2 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 %Q0.1 (* 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 %LS:
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
```

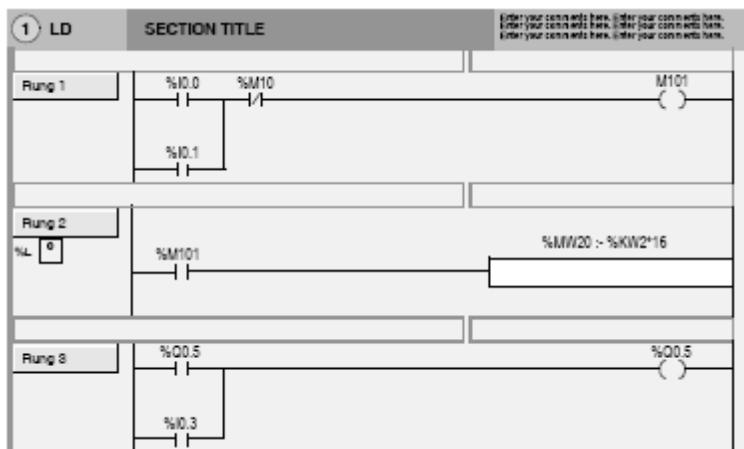
12.1.9.3 REVERSING LIST COMMENTS TO LADDER

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.

12.1.9.4 EXAMPLE OF RUNG HEADER COMMENTS

The following is an example of a Ladder program with rung header comments.



12.1.9.5 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.

12.2 INSTRUCTION LIST LANGUAGE

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 204

Operation of List Instructions 206

List Language Instructions 207

Using Parentheses 209

Stack Instructions (MPS, MRD, MPP) 210

12.2.1 OVERVIEW OF LIST PROGRAMS

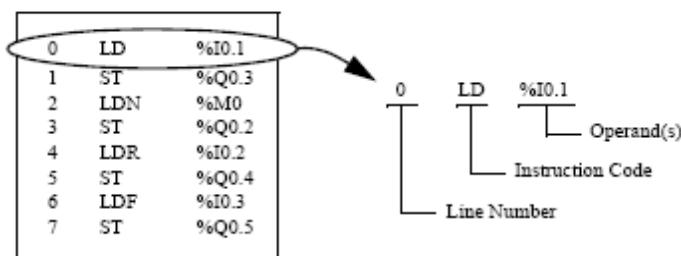
12.2.1.1 INTRODUCTION

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)

12.2.1.2 EXAMPLE OF A LIST PROGRAM

The following is an example of a List program.



12.2.1.3 LINE NUMBER

Line numbers are generated automatically when you enter an instruction. Blank lines and Comment lines do not have line numbers.

12.2.1.4 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).

12.2.1.5 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.

12.2.2 OPERATION OF LIST INSTRUCTIONS

12.2.2.1 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.

12.2.2.2 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.

12.2.2.3 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 %TMO	Starts the timer %TMO
Word instruction	[%MW10 := %MW50+100]	Addition operation
Program instruction	SR5	Calls subroutine #5
Grafset instruction	-*-8	Step #8

12.2.3 LIST LANGUAGE INSTRUCTIONS

12.2.3.1 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.

12.2.3.2 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.
AND(Logic AND (8 parenthesis levels)

Name	Equivalent graphic element	Function
OR(Logic OR (8 parenthesis levels)
XOR, XORN, XORR, XORF		Exclusive OR
MPS MRD MPP		Switching to the coils.
N	-	Negation (NOT)

12.2.3.3 ACTION INSTRUCTIONS

The following table describes action instructions in List language.

Name	Equivalent graphic element	Function
ST	$\neg(\)-$	The associated operand takes the value of the test zone result.
STN	$\neg(\)-$	The associated operand takes the reverse value of the test zone result.
S	$\neg(S)-$	The associated operand is set to 1 when the result of the test zone is 1.
R	$\neg(R)-$	The associated operand is set to 0 when the result of the test zone is 1.
JMP	$\rightarrow\rightarrow \%Li$	Connect unconditionally to a labeled sequence, upstream or downstream.
SRn	$\rightarrow\rightarrow \%SrI$	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.

12.2.3.4 FUNCTION BLOCK INSTRUCTIONS

The following table describes function blocks in List language.

Name	Equivalent graphic element	Function
Timers, counters, registers, and so on.		<p>For each of the function blocks, there are instructions for controlling the block.</p> <p>A structured form is used to hardwire the block inputs and outputs directly.</p> <p>Note: Outputs of function blocks can not be connected to each other (vertical shorts).</p>

12.2.4 USING PARENTHESES

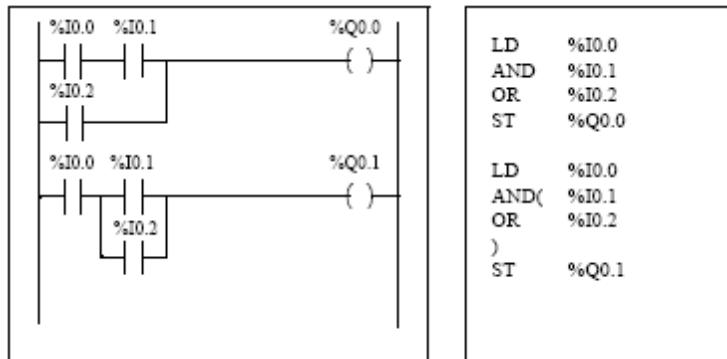
12.2.4.1 INTRODUCTION

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.

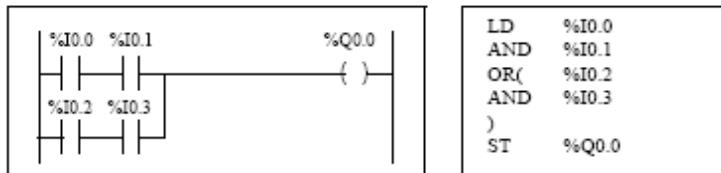
12.2.4.2 EXAMPLE USING AN AND INSTRUCTION

The following diagrams are examples of using a parentheses with an AND instruction: AND(...).



12.2.4.3 EXAMPLE USING AN OR INSTRUCTION

The following diagrams are examples of using parentheses with an OR instruction: OR(...).



12.2.4.4 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
I	Comparison	See <i>Comparison instructions, p. 384</i>

12.2.4.5 NESTING PARENTHESIS

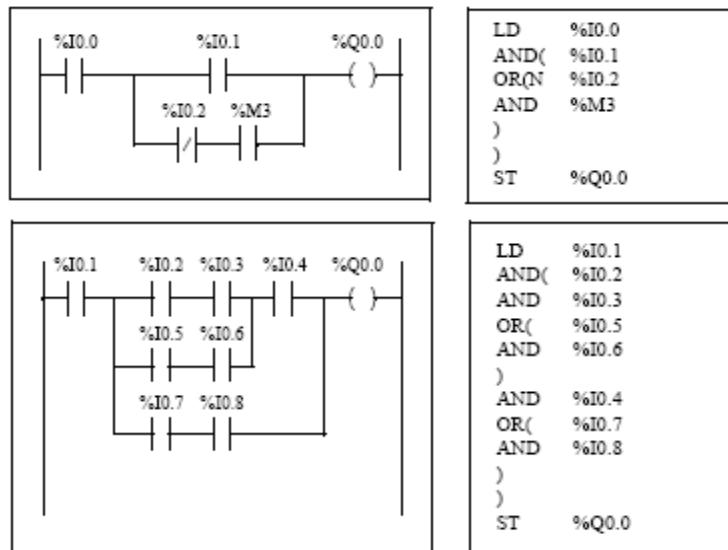
It is possible to nest up to eight levels of parentheses.

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.

12.2.4.6 EXAMPLES OF NESTING PARENTHESES

The following diagrams provide examples of nesting parentheses.



12.2.4 STACK INSTRUCTIONS (MPS, MRD, MPP)

12.2.4.1 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.

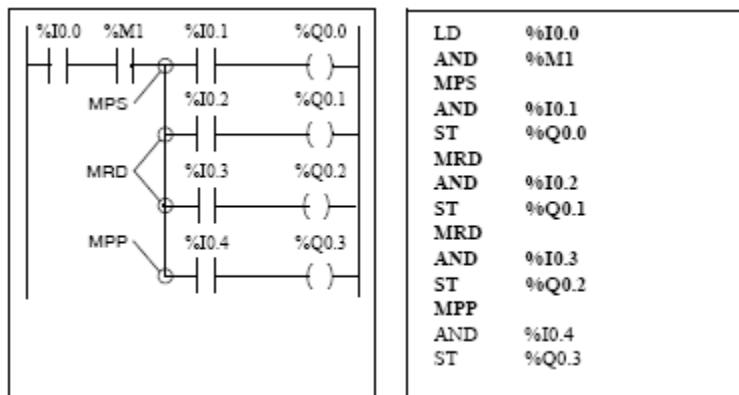
12.2.4.2 OPERATION OF STACK INSTRUCTIONS

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.

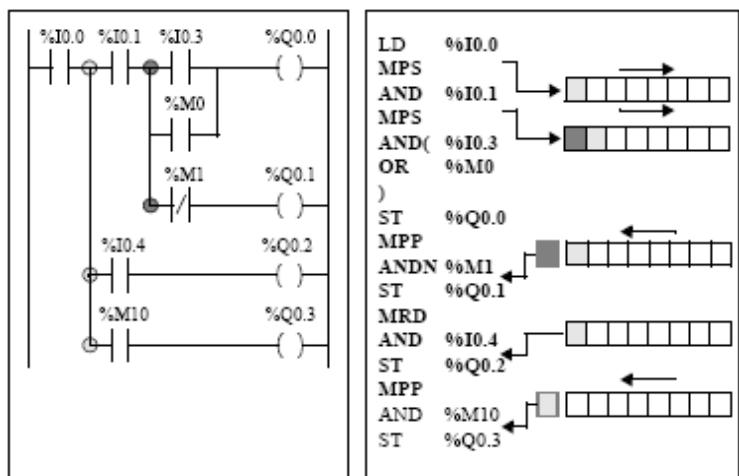
12.2.4.3 EXAMPLES OF STACK INSTRUCTIONS

The following diagrams are examples of using stack instructions



12.2.4.4 EXAMPLES OF STACK OPERATION

The following diagrams display how stack instructions operate.



12.3 GRAFCET

AT A GLANCE

SUBJECT OF THIS CHAPTER

This chapter describes programming using Grafset Language.

WHAT'S IN THIS CHAPTER?

This chapter contains the following topics:

Topic Page

Description of Grafset Instructions 212

Description of Grafset Program Structure 214

Actions Associated with Grafset Steps 216

12.3.1 DESCRIPTION OF GRAFCET INSTRUCTIONS

12.3.1.1 INTRODUCTION

Grafset instructions in TwidoSuite offer a simple method of translating a control sequence (Grafset chart).

The maximum number of Grafset 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.

12.3.1.2 GRAFCET INSTRUCTIONS

The following table lists all instructions and objects required to program a Grafset chart:

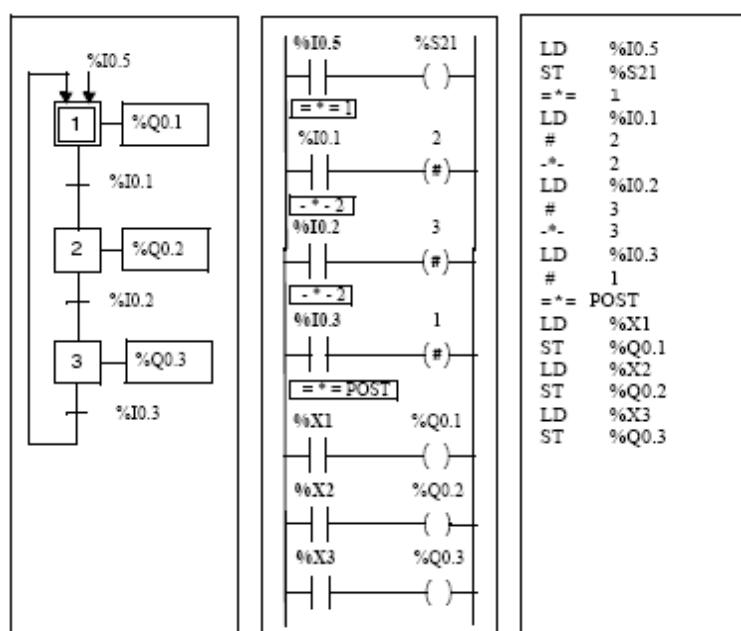
Graphic representation (1)	Transcription in TwidoSuite language	Function
Initial step	=*= i	Start the initial step (2)
Transition	# i	Activate step i after deactivating the current step
Step	-*- i	Start step i and validate the associated transition (2)
	#	Deactivate the current step without activating any other steps
	#Di	Deactivate step i and the current step
	=*= POST	Start post-processing and end sequential processing
	%Xi	Bit associated with step i, can be tested and written (maximum number of steps depends on controller)
x_i	LD %Xi, LDN %Xi AND %Xi, ANDN %Xi, OR %Xi, ORN %Xi XOR %Xi, XORN %Xi	Test the activity of step i
x_i	S %Xi	Activate step i
x_i (s)	R %Xi	Deactivate step i
x_i (R)		

(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.

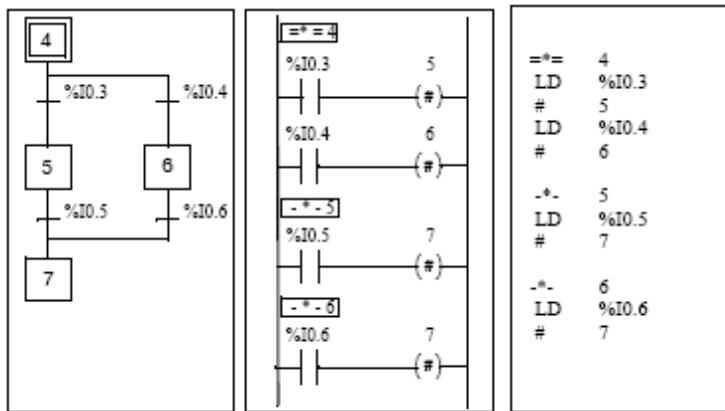
12.3.1.3 GRAFCET EXAMPLES

Linear sequence:



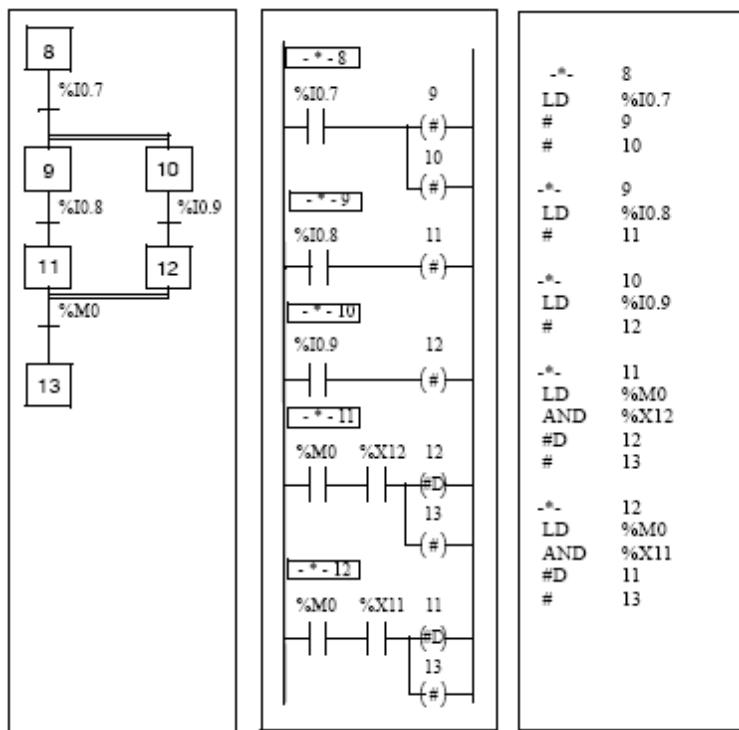
Not supported Twido Ladder Twido Instruction Language program List program

Alternative sequence:



Not supported Twido Ladder Twido Instruction Language program List program

Simultaneous sequences:



Not supported Twido Ladder Twido Instruction Language program 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 prepositioned during preprocessing using system bit %S23 and the instruction S %Xi.

12.3.2 DESCRIPTION OF GRAFCET PROGRAM STRUCTURE

12.3.2.1 INTRODUCTION

A TwidoSuite Grafcet program has three parts:

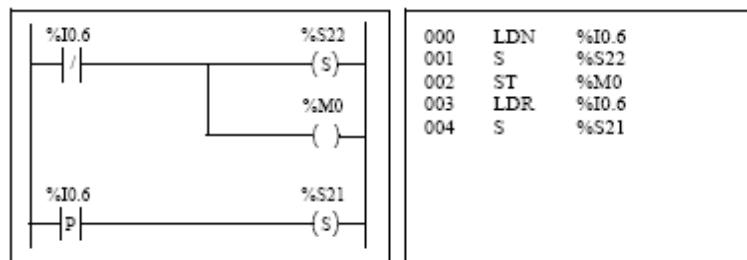
- ▲ Preprocessing
- ▲ Sequential processing
- ▲ Post-Processing

12.3.2.2 PREPROCESSING

Preprocessing consists of the following:

- ▲ Power returns
- ▲ Faults
- ▲ Changes of operating mode
- ▲ Pre-positioning Grafset steps
- ▲ Input logic

The rising edge of input %I0.6 sets bit %S21 to 1. This disables the active steps and enables the initial 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 Grafset 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	Grafset initialization	All active steps are deactivated and the initial steps are activated.
%S22	Grafset re-initialization	All steps are deactivated.
%S23	Grafset 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, Grafset is frozen (no updates are taken into account).

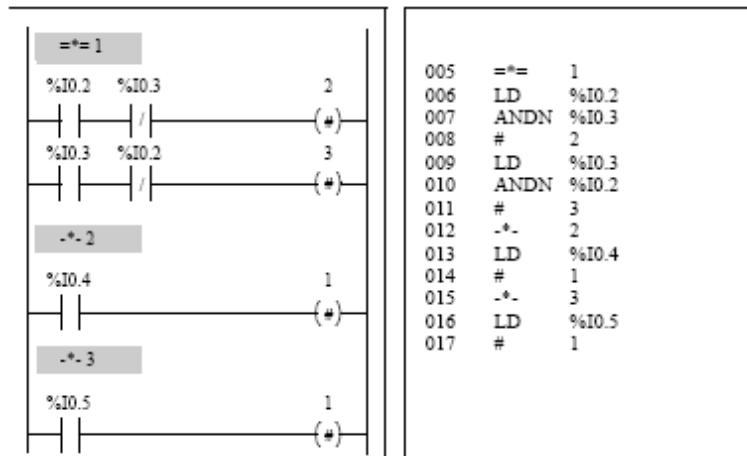
12.3.2.3 SEQUENTIAL PROCESSING

Sequential processing takes place in the chart (instructions representing the chart):

- ▲ Steps
- ▲ Actions associated with steps
- ▲ Transitions

▲ Transition conditions

Example:



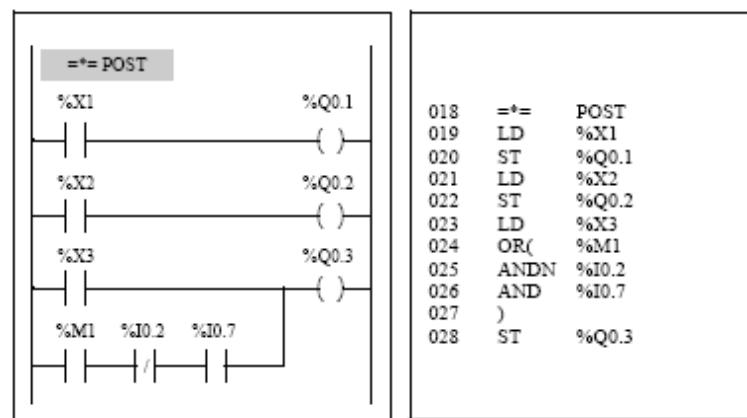
Sequential processing ends with the execution of the "= * = POST" instruction or with the end of the program.

12.3.2.4 POST-PROCESSING

Post-processing consists of the following:

- ▲ Commands from the sequential processing for controlling the outputs
- ▲ Safety interlocks specific to the outputs

Example:



12.3.3 ACTIONS ASSOCIATED WITH GRAFCET STEPS

12.3.3.1 INTRODUCTION

A TwidoSuite Grafcet program offers two ways to program the actions associated with steps:

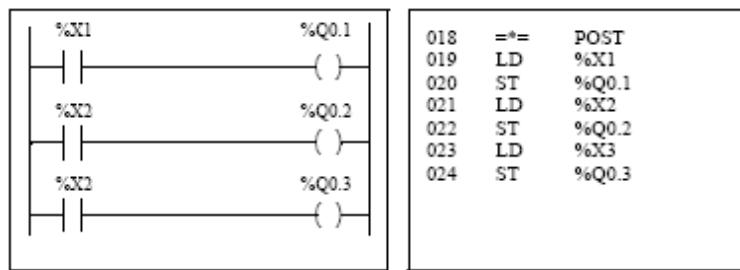
- ▲ In the post-processing section

- ▲ Within List instructions or Ladder rungs of the steps themselves

12.3.3.2 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 Grafcel application. You can use Set and Reset List instructions or energize coils in a Ladder program to activate Grafcel steps (%Xi).

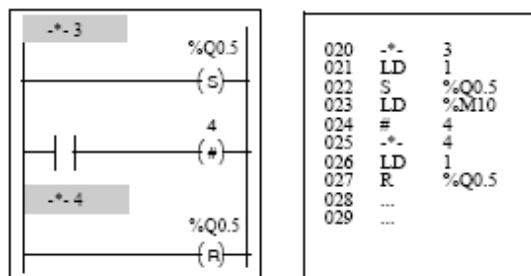
Example:



12.3.3.3 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 Grafcel.

Example:



13. BASIC INSTRUCTIONS

This chapter contains the following sections:

Section Topic Page

13.1 Boolean Processing 218

13.2 Basic Function Blocks 227

13.3 Numerical Processing 242

13.4 Program Instructions 254

13.1 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 212

Understanding the Format for Describing Boolean Instructions 214

Load Instructions (LD, LDN, LDR, LDF) 215

Assignment instructions (ST, STN, R, S) 215

Logical AND Instructions (AND, ANDN, ANDR, ANDF) 217

Logical OR Instructions (OR, ORN, ORR, ORF) 218

Exclusive OR, instructions (XOR, XORN, XORR, XORF) 219

NOT Instruction (N) 220

13.1.1 BOOLEAN INSTRUCTIONS



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
```

13.1.1.1 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.

13.1.1.2 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.

13.1.1.3 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.

13.1.1.4 EDGE DETECTION

The following table summarizes the instructions and timing for detecting edges:

13.1.2 UNDERSTANDING THE FORMAT FOR DESCRIBING BOOLEAN INSTRUCTIONS

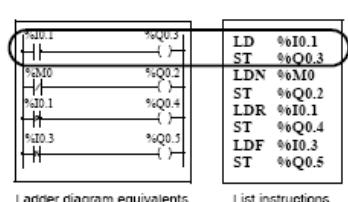
Each Boolean instruction in this section is described using the following information:

- ▲ 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.

13.1.2.1 EXAMPLES

The following illustration shows how examples are given for each instruction.



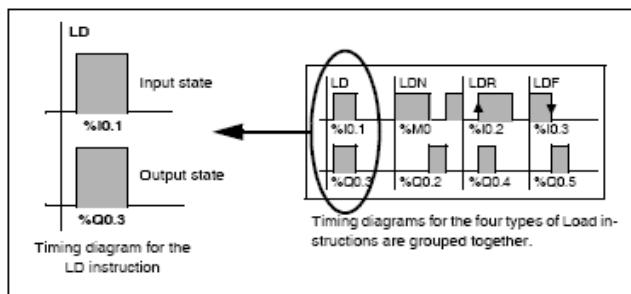
13.1.2.2 PERMITTED OPERANDS

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:O)
%*Xk	Word bit (for example, %MWi:Xk)
[]	Comparison expression (for example, [%MWi<1000])

13.1.2.3 TIMING DIAGRAMS

The following illustration shows how timing diagrams are displayed for each instruction.

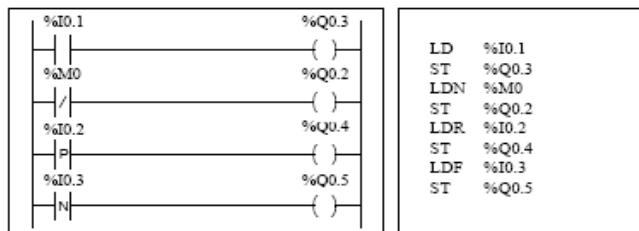


13.1.3 LOAD INSTRUCTIONS (LD, LDN, LDR, LDF)

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).

13.1.3.1 EXAMPLES

The following diagrams are examples of Load instructions.



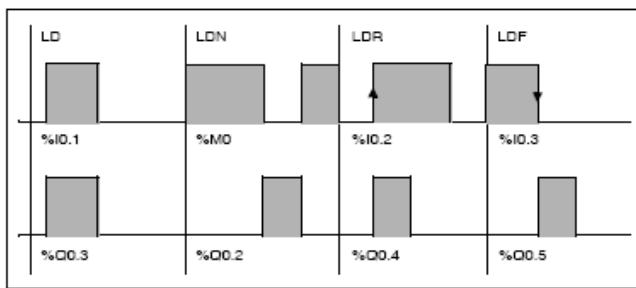
13.1.3.2 PERMITTED OPERANDS

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, %IWx:y.z:Xk, %Q, %OA, %M, %G, %X, %BLK.x, %*:Xk,[]
LDN	— —	0/1, %I, %IA, %IWx:y.z:Xk, %Q, %OA, %M, %G, %X, %BLK.x, %*:Xk,[]
LDR	— P —	%I, %IA, %M
LDF	— N —	%I, %IA, %M

13.1.3.3 TIMING DIAGRAM

The following diagram displays the timing for Load instructions.

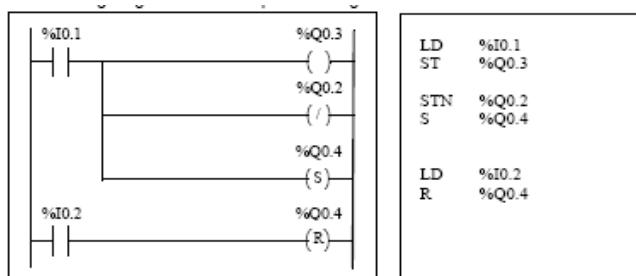


13.1.4 ASSIGMENT INSTRUCTIONS (ST, STN, R, S)

The assignment instructions ST, STN, S, and R correspond respectively to the direct, inverse, set, and reset coils.

13.1.4.1 EXAMPLES

The following diagrams are examples of assignment instructions.



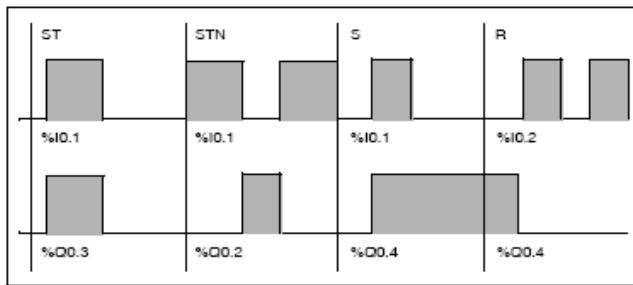
13.1.4.2 PERMITTED OPERATIONS

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, %G, %BLK.x, %*:Xk
STN	(/)	%Q, %QA%M, %G, %BLK.x, %*:Xk
S	(s)	%Q, %QA, %M, %G, %X, %BLK.x, %*:Xk
R	(R)	%Q, %QA, %M, %G, %X, %BLK.x, %*:Xk

13.1.4.3 TIMING OPERATIONS

The following diagram displays the timing for assignment instructions.

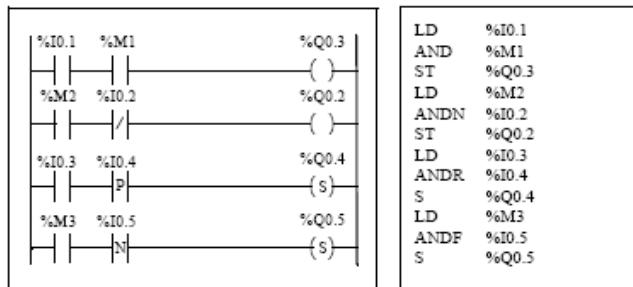


13.1.5 LOGICAL AND INSTRUCTIONS (AND, ANDN, ANDR, ANDF)

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.

13.1.5.1 EXAMPLES

The following diagrams are examples of logic AND instructions.



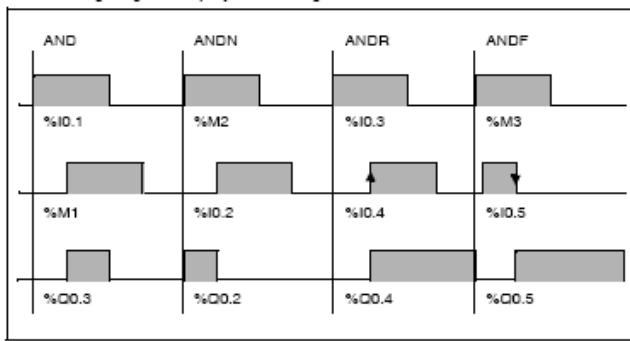
13.1.5.1 PERMITTED OPERANDS

The following table lists the types of AND instructions with ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
AND		I/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.X, %*XK, []
ANDN		I/1, %I, %IA, %Q, %QA, %M, %S, %X, %BLK.X, %*XK, []
ANDR		%I, %IA, %M
ANDF		%I, %IA, %M

13.1.5.2 TIMING DIAGRAM

The following diagram displays the timing for the AND instructions.



13.1.6 LOGICAL OR INSTRUCTIONS (OR, ORN, ORR, ORF)

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.

13.1.6.1 EXAMPLES

The following diagrams are examples of logic OR instructions.

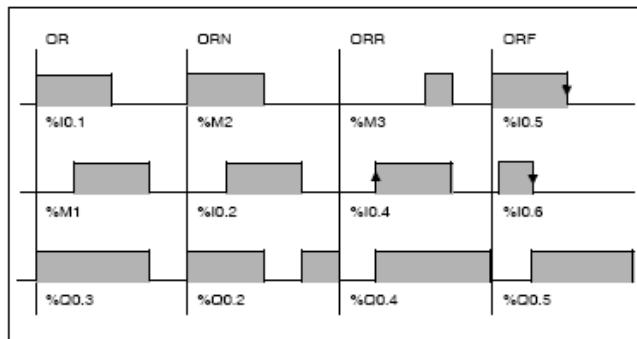
13.1.6.2 PERMITTED OPERANDS

The following table lists the types of OR instructions with Ladder equivalents and permitted operands.

List Instruction	Ladder Equivalent	Permitted Operands
OR		I0/I1, %I0, %IA, %Q, %QA, %M, %S, %X, %BLK.X, %*:Xk
ORN		I0/I1, %I0, %IA, %Q, %QA, %M, %S, %X, %BLK.X, %*:Xk
ORR		%I0, %IA, %M
ORF		%I0, %IA, %M

13.1.6.3 TIMING DIAGRAM

The following diagram displays the timing for the OR instructions.



13.1.7 EXCLUSIVE OR INSTRUCTIONS (XOR, XORN, XORR, XORF)

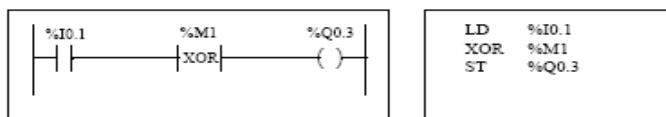
13.1.7.1 INTRODUCTION

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.

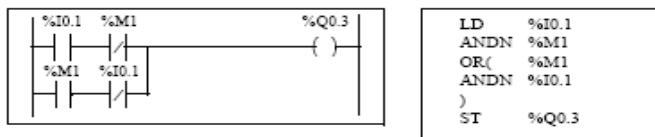
13.1.7.2 EXAMPLES

The following example shows the use of XOR instructions.

Schematic using XOR instruction:



Schematic NOT using XOR instruction :



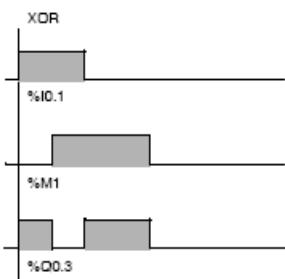
13.1.7.3 PERMITTED OPERANDS

The following table lists the types of XOR instructions and permitted operands.

List instruction	Permitted Operands
XOR	%I, %IA, %Q, %QA, %M, %S, %X, %BLKX, %*XX
XORN	%I, %IA, %Q, %QA, %M, %S, %X, %BLKX, %*XX
XORR	%I, %IA, %M
XORF	%I, %IA, %M

13.1.7.4 TIMING DIAGRAM

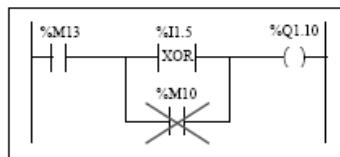
The following diagram displays the timing for the XOR instructions.



13.1.7.5 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.



13.1.8 NOT INSTRUCTION (N)

13.1.8.1 INTRODUCTION

The NOT (N) instruction negates the Boolean result of the preceding instruction.

13.1.8.2 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

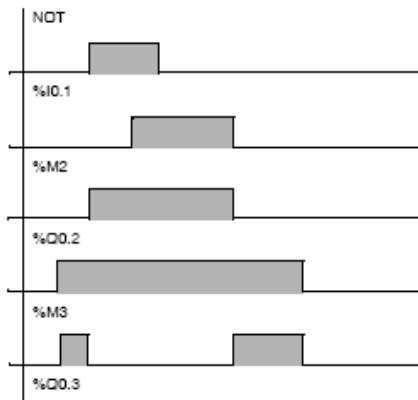
Note: The NOT instruction is not reversible.

13.1.8.3 PERMITTED OPERANDS

Not applicable.

13.1.8.4 TIMING DIAGRAM

The following diagram displays the timing for the NOT instruction.



13.2 BASIC FUNCTION BLOCKS

13.2.1 AT A GLANCE

13.2.1.1 AIM OF THIS SECTION

This section provides descriptions and programming guidelines for using basic function blocks.

13.2.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Basic Function Blocks 228

Standard function blocks programming principles 229

Timer Function Block (%TMi) 231

TOF Type of Timer 232

TON Type of Timer 232

TP Type of Timer 233

Programming and Configuring Timers 233

Up/Down Counter Function Block (%Ci) 236

- Programming and Configuring Counters 238
 Shift Bit Register Function Block (%SBRi) 239
 Step Counter Function Block (%SCi) 241

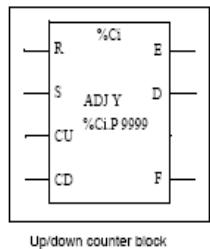
13.2.2 BASIC FUNCTION BLOCKS

13.2.2.1 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.

13.2.2.2 EXAMPLE OF A FUNCTION BLOCK

The following illustration is an example of an up/down Counter function block.



13.2.2.3 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. 358).
- ▲ By specifying the block type (for example, LD %Ci.E). Inputs can be accessed in the form of instructions.

13.2.2.4 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.

13.2.2.5 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	%TMi	0 - 127	Word	Current Value	%TMi.V	no
				Preset value	%TMi.P	yes
			Bit	Timer output	%TMi.Q	no
Up/Down Counter	%Ci	0 - 127	Word	Current Value	%Ci.V	no
				Preset value	%Ci.P	yes
				Underflow output (empty)	%Ci.E	no
			Bit	Preset output reached	%Ci.D	no
				Overflow output (full)	%Ci.F	no

13.2.3 STANDARD FUNCTION BLOCKS PROGRAMMING PRINCIPLES

13.2.3.1 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).

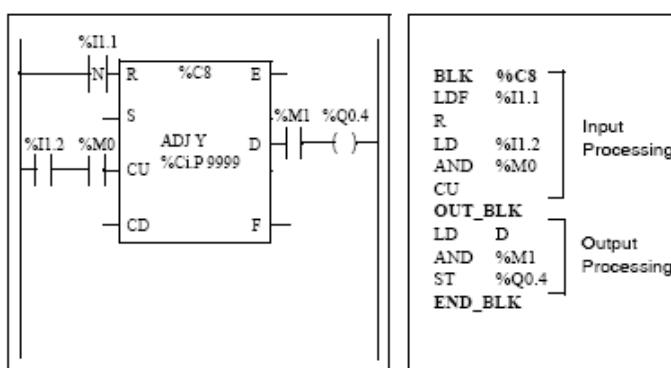
13.2.3.2 REVERSIBLE PROGRAMMING

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.

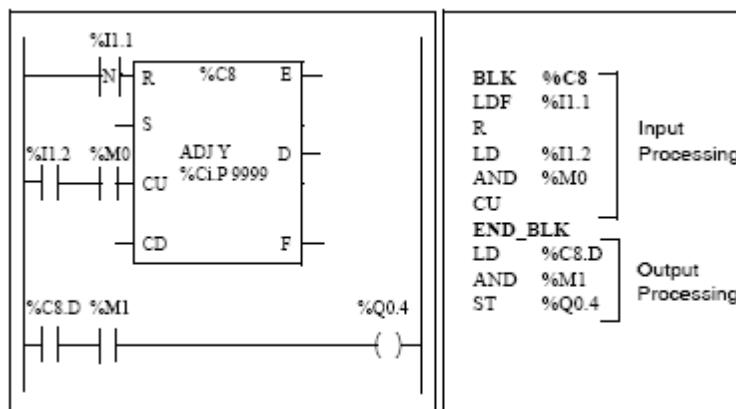
13.2.3.3 EXAMPLE WITH OUTPUT WIRING

The following example shows reversible programming of a counter function block with wired outputs



13.2.3.4 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).

13.2.4 TIMER FUNCTION BLOCK (%TMI)

13.2.4.1 INTRODUCTION

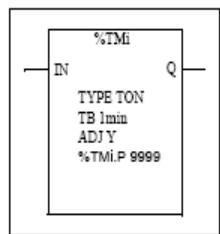
There are three types of Timer function blocks:

- ▲ 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 TwidoSuite.

13.2.4.2 ILLUSTRATION

The following is an illustration of the Timer function block.



Timer function block

13.2.4.3 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.

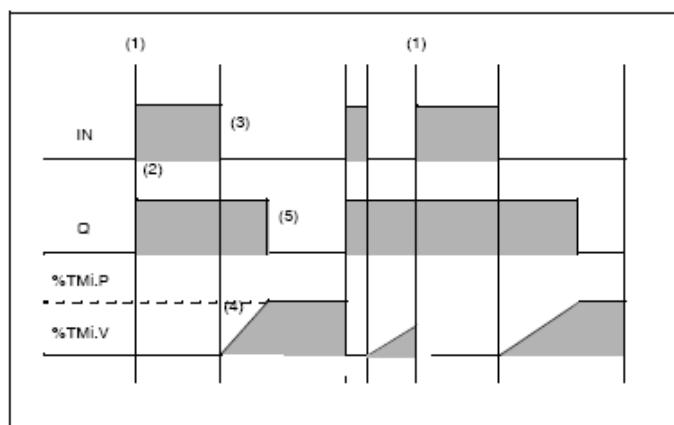
13.2.5 TOF TYPE OF TIMER

13.2.5.1 INTRODUCTION

Use the TOF (Timer Off-Delay) type of timer to control off-delay actions. This delay is programmable using TwidoSuite.

13.2.5.2 TIMING DIAGRAM

The following timing diagram illustrates the operation of the TOF type timer.



13.2.5.3 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.

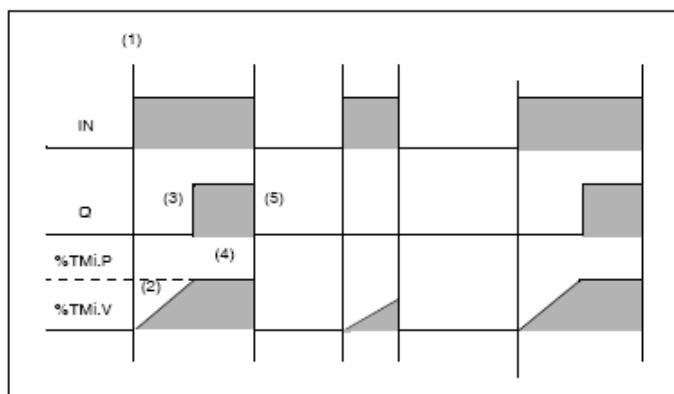
13.2.6 TON TYPE OF TIMER

13.2.6.1 INTRODUCTION

The TON (Timer On-Delay) type of timer is used to control on-delay actions. This delay is programmable using the TwidoSuite.

13.2.6.2 TIMING DIAGRAM

The following timing diagram illustrates the operation of the TON type timer.



13.2.6.3 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.

13.2.7 TP TYPE OF TIMER

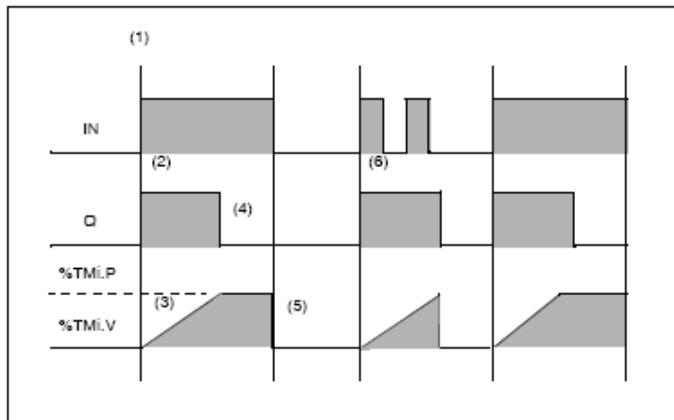
13.2.7.1 INTRODUCTION

The TP (Timer - Pulse) type of timer is used to create pulses of a precise duration.

This delay is programmable using the TwidoSuite.

13.2.7.2 TIMING DIAGRAM

The following timing diagram illustrates the operation of the TP type timer.



13.2.7.3 OPERATION

The following table describes the operation of the TP type timer.

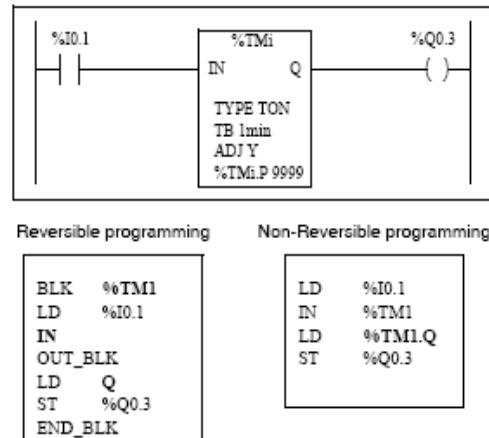
Phase	Description
1	The timer starts on the rising edge of the IN input. The current value %TMi.V is set to 0 if the timer has not already started.
2	The %TMi.Q output bit is set to 1 when the timer starts.
3	The current value %TMi.V of the timer increases from 0 to %TMi.P in increments of one unit per 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.

13.2.8 PROGRAMMING AND CONFIGURING TIMERS

13.2.8.1 INTRODUCTION

Timer function blocks (%TMi) 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.

13.2.8.2 EXAMPLES



The following illustration is a timer function block with examples of reversible and non-reversible programming.

er function block with examples of

13.2.8.3 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 (%TMi.P): 0 to 9999
- ▲ Adjust: Checked or Not Checked

13.2.8.4 SPECIAL CASES

The following table contains a list of special cases for programming the Timer function block.

13.2.8.5 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.

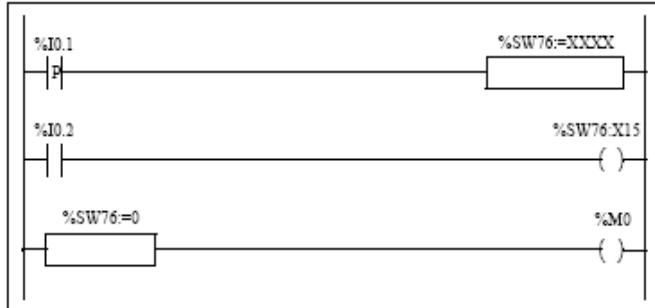
13.2.8.6 PROGRAMMING EXAMPLE

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
.....

```



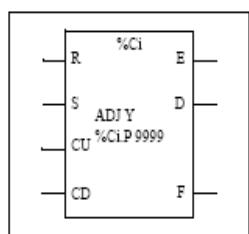
13.2.9 UP/DOWN COUNTER FUNCTION BLOCK (%Ci)

13.2.9.1 INTRODUCTION

The Counter function block (%Ci) provides up and down counting of events. These two operations can be done simultaneously.

13.2.9.2 ILLUSTRATION

The following is an illustration of the up/down Counter function block.



Up/down counter function block

13.2.9.3 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 ≤ %Ci.P ≤ 9999. Word can be read, tested, and written (default value: 9999).
Edit using the Animation Tables Editor	ADJ	<ul style="list-style-type: none"> 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).

13.2.9.4 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.

13.2.9.5 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)	<ul style="list-style-type: none"> The current value %C1.V is set to 0. Output bits %C1.E, %C1.D, and %C1.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 (%C1.V).
Effect of modifying the preset %C1.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).

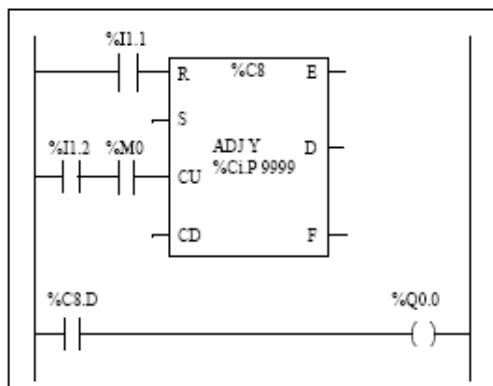
13.2.10 PROGRAMMING AND CONFIGURING COUNTERS

13.2.10.1 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.

13.2.10.2 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
LD %I1.2
AND %M0
CU
END_BLK
LD %C8.D
ST %Q0.0
```

```
LD %I1.1
R %C8
LD %I1.2
AND %M0
CU %C8
LD %C8.D
ST %Q0.0
```

Reversible Programming

Non-Reversible programming

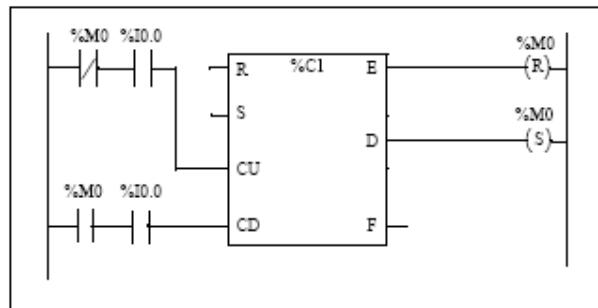
13.2.10.3 CONFIGURATION

The following parameters must be entered during configuration:

- ▲ Preset value (%Ci.P): set to 5000 in this example
- ▲ Adjust: Yes

13.2.10.4 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.

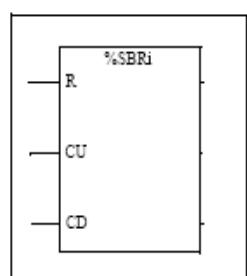
13.2.11 SHIFT BIT REGISTER FUNCTION BLOCK (%SBRI)

13.2.11.1 INTRODUCTION

The Shift Bit Register function block (%SBRI) provides a left or right shift of binary data bits (0 or 1).

13.2.11.2 ILLUSTRATION

The following is an example of a Shift Register function block.



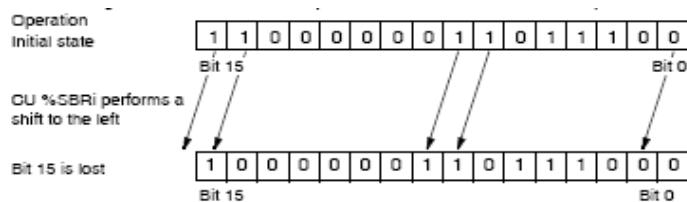
13.2.11.3 PARAMETERS

The Shift Bit Register function block has the following parameters.

Parameter	Label	Value
Register number	%SBR <i>i</i>	0 to 7
Register bit	%SBR <i>i.j</i>	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 %SBR <i>i.j</i> 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.

13.2.11.4 OPERATION

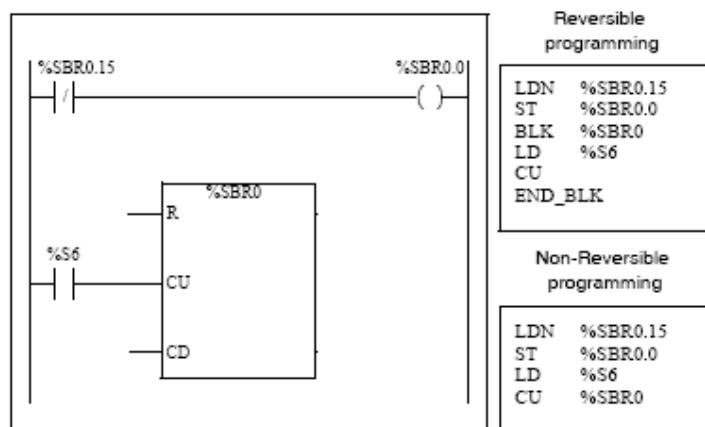
The following illustration shows a bit pattern before and after a shift operation.



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.

13.2.11.5 PROGRAMMING

In the following example, a bit is shifted to the left every second while Bit 0 assumes the opposite state to Bit 15.



13.2.11.6 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.

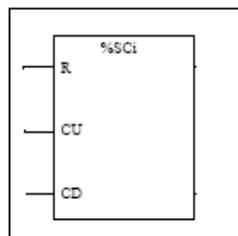
13.2.12 STEP COUNTER FUNCTION BLOCK (%SCI)

13.2.12.1 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 (step counter bit %SCI.j) is set to 1. The step counter can control output bits (%Qi.j), internal bits (%Mi) or AS interface slave output bits (%QAx.y.z). Only one step of a step counter can be active at a time.

13.2.12.2 ILLUSTRATION

The following is an example of a Step Counter function block.



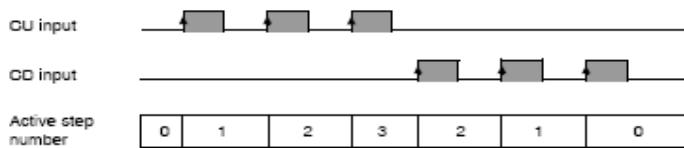
13.2.12.3 PARAMETERS

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.

13.2.12.4 TIMING DIAGRAM

The following timing diagram illustrates the operation of the step function block.



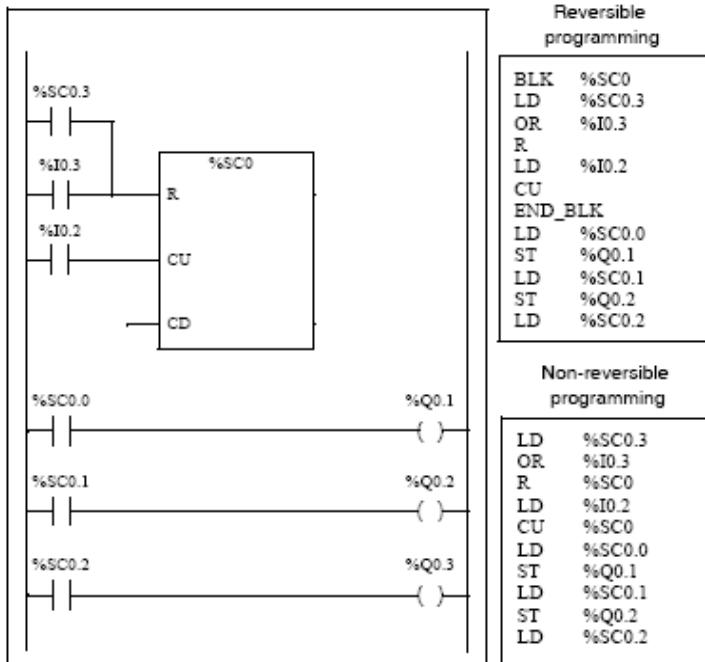
13.2.12.5 PROGRAMMING

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.



13.2.12.6 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.

13.3 NUMERICAL PROCESSING

13.3.1 AT A GLANCE

13.3.1.1 AIM OF THIS SECTION

This section provides an introduction to Numerical Processing including descriptions and programming guidelines.

13.3.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

- Introduction to Numerical Instructions 243
- Assignment Instructions 243
- Comparison Instructions 247
- Arithmetic Instructions on Integers 248
- Logic Instructions 250
- Shift Instructions 251
- Conversion Instructions 252
- Single/double word conversion instructions 263

13.3.2 INTRODUCTION TO NUMERICAL INSTRUCTIONS

13.3.2.1 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.

13.3.3 ASSIGNMENT INSTRUCTIONS

13.3.3.1 INTRODUCTION

Assignment instructions are used to load operand Op2 into operand Op1.

13.3.3.2 ASSIGNMENT

Syntax for Assignment instructions.

$$[\text{Op1} := \text{Op2}] \quad <=> \quad \text{Op2} \rightarrow \text{Op1}$$

Assignment operations can be performed on:

- ▲ Bit strings
- ▲ Words
- ▲ Double words
- ▲ Floating word

- Word tables
- Double word tables
- Floating word tables

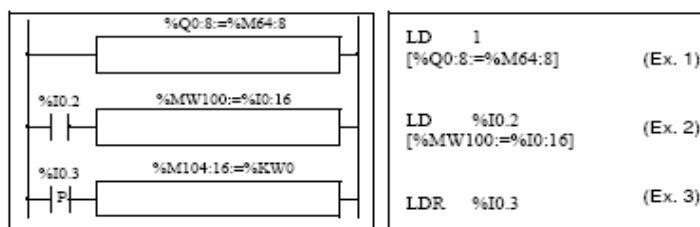
13.3.3.3 ASSIGNMENT OF BIT STRINGS

Operations can be performed on the following bit strings (see Structured Objects, p. 46):

- 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

13.3.3.4 EXAMPLES

Examples of bit string assignments.



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).

13.3.3.5 BIT STRING ASSIGNMENTS

Syntax for bit string assignments.

Operator	Syntax	Operand 1 (Op1)	Operand 2 (Op2)
<code>:=</code>	<code>[Op1 := Op2]</code> Operand 1 (Op1) assumes the value of operand 2 (Op2)	<code>%MWi,%QWi, %QWAI,%SWi %MWi[%MWi], %MDi, %MDi[%MWi] %Mi:L, %Qi:L, %Si:L, %Xi:L</code>	Immediate value, %MWi, %KWi, %IW, %IWAI, %INWi, %QWi, %QWAI %ONWi, %SWi, %BLKx, %MWi[%MWi], %KWi[%MWi], %MDi[%MWi], %KDi[%MWi], %Mi:L, %Qi:L, %Si:L, %Xi:L, %Ii:L

Note: The abbreviation %BLK.x (for example, %C0.P) is used to describe any function block word.

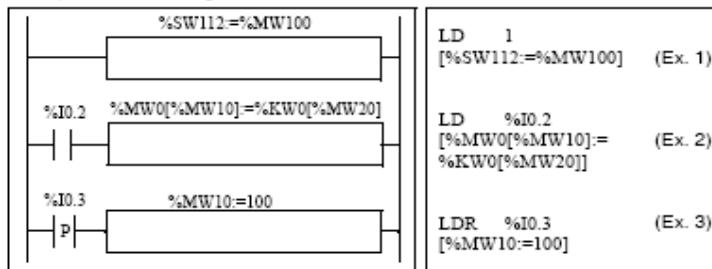
13.3.3.6 ASSIGNMENT OF WORDS

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)

13.3.3.7 EXAMPLES

Examples of word assignments.



13.3.3.8 SYNTAX

Syntax for word assignments.

Operator	Syntax
<code>:=</code>	<code>[Op1 := Op2]</code> Operand 1 (Op1) assumes the value of operand 2 (Op2)

The following table gives details on 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, %Di:L, %Si:L, %Xi:L	Immediate value, %MWi, %KWi, %IW, %IWAi, %QWi, %QWAI, %SWi, %MWi[%MWi], %KWi[%MWi], %MDi, %MDi[%MWj], %KD, %KD[%MWj], %INW, %ML, %QL, %ONW, %Si:L, %Xi:L, %i: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, ...).

13.3.3.9 ASSIGNMENT OF WORD, DOUBLE WORD AND FLOATING POINT TABLES

Assignment operations can be performed on the following object tables (see Tables of words, p. 47):

- 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

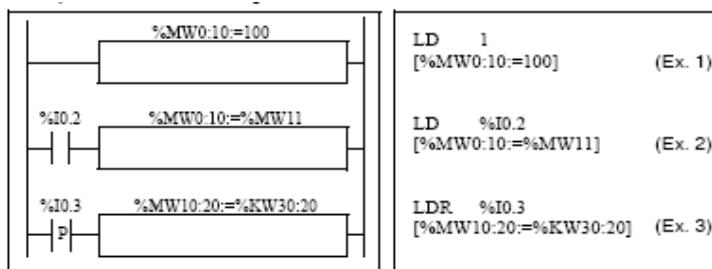
Table length (L) should be the same for both tables.

- 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.

13.3.3.10 EXAMPLES

Examples of word table assignments:



13.3.3.11 SYNTAX

Syntax for word, double word and floating point table assignments

Operator	Syntax
<code>:=</code>	<code>[Op1 := Op2]</code> 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, %KWi, %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.

13.3.4 COMPARISON INSTRUCTIONS

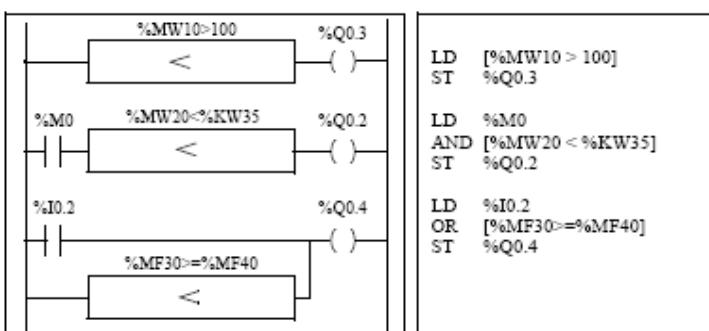
13.3.4.1 INTRODUCTION

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

13.3.4.2 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.



13.3.4.3 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, %ONWi, %OWi, %OWAi, %ONWi, %SWi, %BLK.x	Immediate value, %MWi, %KWi, %INWi, %IW, %IWAi, %ONWi, %OW, %OWAi, %SWi, %BLK.x, %MWi [%MWi], %KWi [%MW]
Double words	%MDi, %KDi	Immediate value, %MDi, %KDi, %MDi [%MW], %KD [%MW]
Floating word	%MFi, %KFi	Immediate floating point value, %MFi, %KFi, %MFi [%MWi], %KFi [%MWi]

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
```

13.3.5 ARITHMETIC INSTRUCTIONS ON INTEGERS

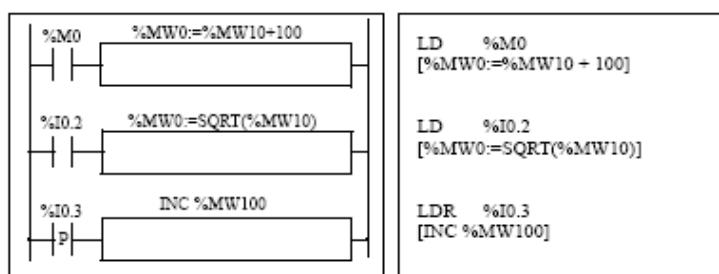
13.3.5.1 INTRODUCTION

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

13.3.5.2 STRUCTURE

Arithmetic operations are performed as follows:



13.3.5.3 SYNTAX

The syntax depends on the operators used as shown in the table below.

Operator	Syntax
+,-,*,/,REM	[Op1 := Op2 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, %OWi, %OWAi, %SWi	Immediate value, %MWi, %KWi, %INW, %IW, %IWAi, %INW, %OW, %OWAi, %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.

13.3.5.4 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 21474836487.

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 divider 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

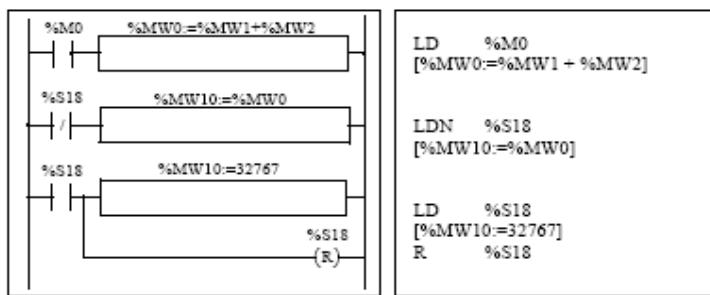
- ▲ Overflow during operation

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).

13.3.5.5 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.

13.3.6 LOGIC INSTRUCTIONS

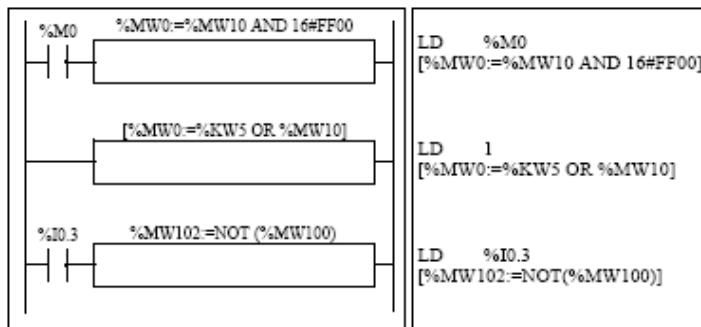
13.3.6.1 INTRODUCTION

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

13.3.6.2 STRUCTURE

Logic operations are performed as follows:



13.3.6.3 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 Ops]	%MWi, %QWi, %OWAi, %SWi	Immediate value (1), %MWi, %KWi, %IW, %IWAI, %OW, %OWAI, %SWi, %BLK.x
NOT	[Op1:=NOT(Op2)]		

Note: (1) With NOT, Op2 cannot be an immediate value.

13.3.6.4 EXAMPLE

The following is an example of a logical AND instruction: [%MW15:=%MW32 AND %MW12]

13.3.7 SHIFT INSTRUCTIONS

13.3.7.1 INTRODUCTION

Shift instructions move bits of an operand a certain number of positions to the right or to the left.

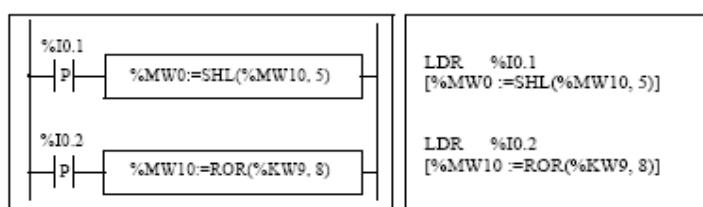
The following table lists the types of Shift instructions.

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. 564) is used for capacity overrun.

13.3.7.2 STRUCTURE

Shift operations are performed as follows:



13.3.7.3 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, %OWi, %OWAi, %SWi	%MWi, %KWi, %IW, %IWAI, %OW, %OWAi, %SWi, %BLK.X
Double word	%MDi	%MDi, %KDi

13.3.8 CONVERSION INSTRUCTIONS

13.3.8.1 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

13.3.8.2 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

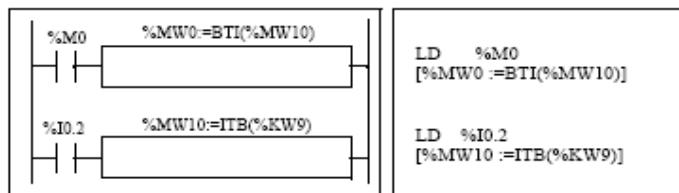
Word %MW5 is converted to word %MW12 by using instruction BTI.

Word %MW12 is converted to word %MW5 by using instruction ITB.



13.3.8.3 STRUCTURE

Conversion operations are performed as follows:



13.3.8.4 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	%MW _i , %QW _i , %QWA _i , %SW _i	%MW _i , %KW _i , %IW, %IWA _i , %OW, %QWA _i , %SW _i , %BLK.x
Double word	%MD _i	%MD _i , %KD _i

13.3.8.5 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.

13.3.9 SINGLE/DOUBLE WORD CONVERSION INSTRUCTIONS

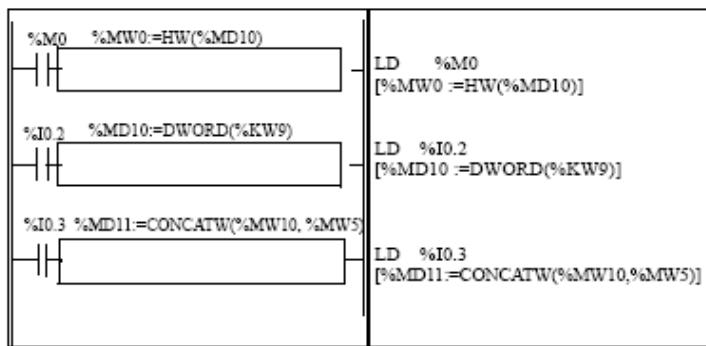
13.3.9.1 INTRODUCTION

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.

13.3.9.2 STRUCTURE

Conversion operations are performed as follows:



13.3.9.3 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)	%MW _i	%MD _i , %KD _i	[.]
CONCATW	Op1 = Operator (Op2, Op3))	%MD _i	%MW _i , %KW _i , immediate value	%MW _i , %KW _i , immediate value
DWORD	Op1 = Operator (Op2)	%MD _i	%MW _i , %KW _i	[.]

13.4 PROGRAM INSTRUCTIONS

13.4.1 AT A GLANCE

13.4.1.1 AIM OF THIS SECTION

This section provides an introduction to Program Instructions.

13.4.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

END Instructions 254

NOP Instruction 255

Jump Instructions 256

Subroutine Instructions 257

13.4.2 END INSTRUCTIONS

13.4.2.1 INTRODUCTION

The End instructions define the end of the execution of a program scan.

13.4.2.2 END, ENDC, AND ENDCN

Three different end instructions are available:

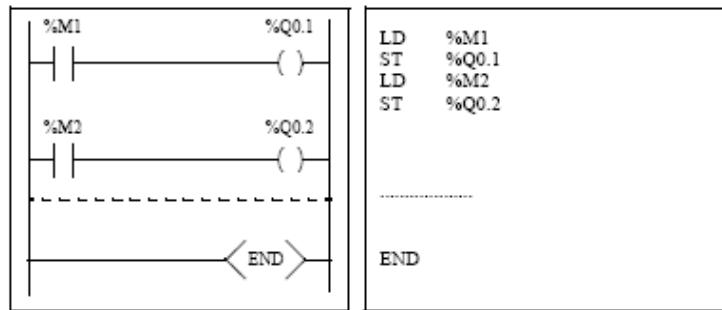
- ▲ END: unconditional end of program
- ▲ ENDC: end of program if Boolean result of preceding test instruction is 1
- ▲ ENDCN: end of program if Boolean result of preceding test instruction is 0

By default (normal mode) when the end of program is activated, the outputs are updated and the next scan is started.

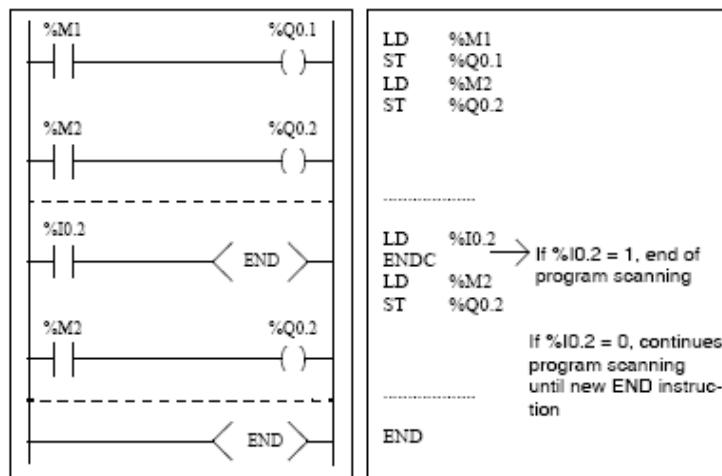
If scanning is periodic, when the end of period is reached the outputs are updated and the next scan is started.

13.4.2.3 EXAMPLES

Example of an unconditional END instruction.



Example of a conditional END instruction.



13.4.3 NOP INSTRUCTION

13.4.3.1 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.

13.4.4 JUMP INSTRUCTIONS

13.4.4.1 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).

13.4.4.2 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

13.4.4.3 EXAMPLES

Examples of jump instructions.

```

000 LD    %M15
001 JMPC  %L8
002 LD    [%MW24-%MW12]
003 ST    %M15
004 JMP   %L12
005 %L8:
006 LD    %M12
007 AND   %M13
008 ST    %M12
009 JMPCN %L12
010 OR    %M11
011 S     %Q0.0
012 %L12:
013 LD    %I0.0

```

13.4.4.4 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.

13.4.5 SUBROUTINE INSTRUCTIONS

13.4.5.1 INTRODUCTION

The Subroutine instructions cause a program to perform a subroutine and then return to the main program.

13.4.5.2 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.

13.4.5.3 EXAMPLE

Examples of subroutine instructions.

```

000 LD    %M15
001 AND   %M5
002 ST    %Q0.0
003 LD    [%MW24-%MW12]
004 SR8
005 LD    %I0.4
006 AND   M13
007 -
008 -
009 -
010 END

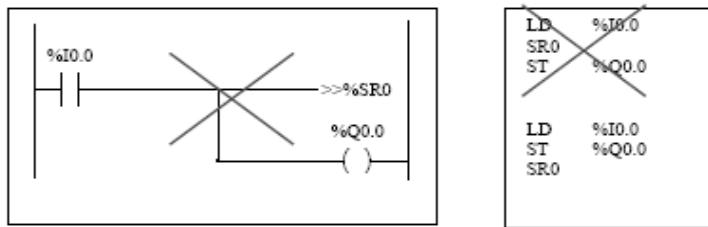
011 SR8:
012 LD    1
013 IN    %TM0
014 LD    %TM0.Q
015 ST    %M15
016 RET
  
```

13.4.5.4 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.



14. ADVANCED INSTRUCTIONS

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

14.1 Advanced Function Blocks 259

14.2 Clock Functions 286

14.3 Twido PID Quick Start Guide 292

14.4 PID Function 306

14.5 Floating point instructions 344

14.6 Instructions on Object Tables 350

14.1 ADVANCED FUNCTION BLOCKS

14.1.1 AT A GLANCE

14.1.1.1 AIM OF THIS SECTION

This section provides an introduction to advanced function blocks including programming examples.

14.1.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Bit and Word Objects Associated with Advanced Function Blocks 260

Programming Principles for Advanced Function Blocks 261



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

- LIFO/FIFO Register Function Block (%Ri) 263
- LIFO Operation 263
- FIFO,operation 264
- Programming and Configuring Registers 264
- Pulse Width Modulation Function Block (%PWM) 266
- Pulse Generator Output Function Block (%PLS) 268
- Drum Controller Function Block (%DR) 269
- Drum Controller Function Block %DRi Operation 270
- Programming and Configuring Drum Controllers 271
- Fast Counter Function Block (%FC) 272
- Very Fast Counter Function Block (%VFC) 274
- Transmitting/Receiving Messages - the Exchange Instruction (EXCH) 282
- Exchange Control Function Block (%MSGx) 283

14.1.2 BIT AND WORD OBJECTS ASSOCIATED WITH ADVANCED FUNCTION BLOCKS

14.1.2.1 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)

14.1.2.2 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 TwidoSuite 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.SO	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
	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

14.1.3 PROGRAMMING PRINCIPLES FOR ADVANCED FUNCTION BLOCKS

14.1.3.1 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 TwidoSuite 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.

14.1.3.2 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. TwidoSuite 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	

14.1.3.3 USING DEDICATED INPUTS AND OUTPUTS

TwidoSuite 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.

14.1.4 LIFO/FIFO REGISTER FUNCTION BLOCK (%RI)

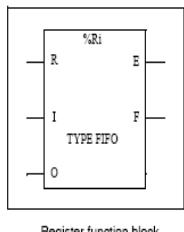
14.1.4.1 INTRODUCTION

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.

14.1.4.2 ILLUSTRATION

The following is an illustration of the register function block.



14.1.4.3 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.

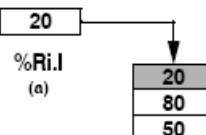
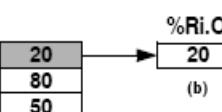
14.1.5 LIFO OPERATION

14.1.5.1 INTRODUCTION

In LIFO operation (Last In, First Out), the last data item entered is the first to be retrieved.

14.1.5.2 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. 
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. 
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.	

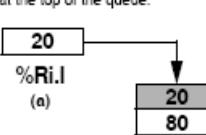
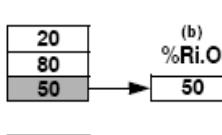
14.1.6 FIFO OPERATION

14.1.6.1 INTRODUCTION

In FIFO operation (First In, First Out), the first data item entered is the first to be retrieved.

14.1.6.2 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).	

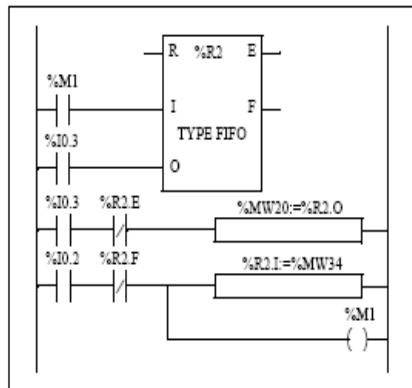
14.1.7 PROGRAMMING AND CONFIGURING REGISTERS

14.1.7.1 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)

14.1.7.2 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]			
ST	%M1		

Reversible program

Non-reversible program

14.1.7.3 CONFIGURATION

The only parameter that must be entered during configuration is the type of register:

- ▲ FIFO (default), or
- ▲ LIFO

14.1.7.4 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 (%SO=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) or a controller stop	Has no effect on the current value of the register, nor on the state of its output bits.

14.1.8 PULSE WIDTH MODULATION FUNCTION BLOCK (%PWM)

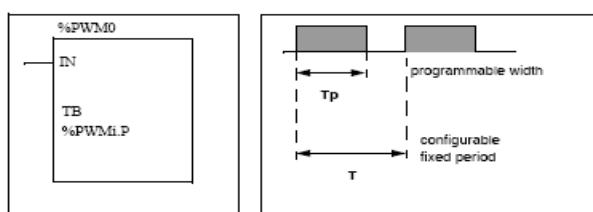
14.1.8.1 INTRODUCTION

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.

14.1.8.2 ILLUSTRATION

PWM block and timing diagram:



14.1.8.3 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 <= 32767 with time base 10 ms or 1 s 0 < %PWMI.P <= 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 Tp is thus equal to: $Tp = T * (%PWMI.R/100)$. 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.

14.1.8.4 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

14.1.8.5 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.



14.1.8.6 PROGRAMMING AND CONFIGURATION

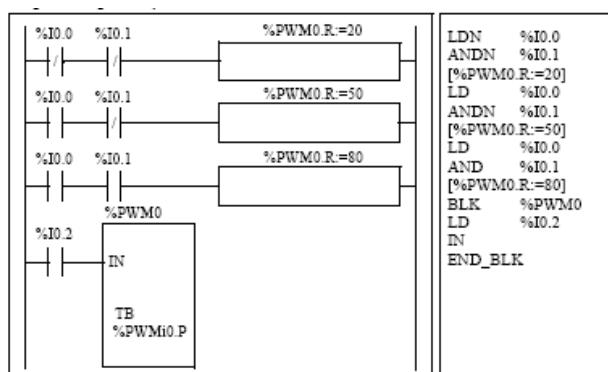
In this example, the signal width is modified by the program according to the state of controller inputs %I0.0.0 and %I0.0.1.

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:



```

LDN   %I0.0
ANDN %I0.1
[%PWM0.R:=20]
LD   %I0.0
ANDN %I0.1
[%PWM0.R:=50]
LD   %I0.0
AND  %I0.1
[%PWM0.R:=80]
BLK
LD   %I0.2
IN
END_BLK
    
```

14.1.8.7 SPECIAL CASES

The following table shows a list of special operating of the PWM function block.

Special case	Description
Effect of a cold restart (%SO=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 (%SI=1)	Has no effect.
Effect due to the fact that outputs are dedicated to the %PWM block	Forcing output %DO.0.0 or %DO.0.1 using a programming device does not stop the signal generation.

14.1.9 PULSE GENERATOR OUTPUT FUNCTION BLOCK (%PLS)

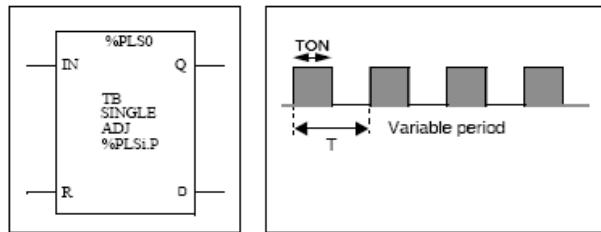
14.1.9.1 INTRODUCTION

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.

14.1.9.2 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=[whole part(%PLSi.P)/2]*TB for the 10ms to 1s time bases.

14.1.9.3 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. <ul style="list-style-type: none"> • 1 < %PLSi.P <= 32767 for time base 10 ms or 1 s • 0 < %PLSi.P <= 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.P >= 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 <= %PLSi.N <= 32767 in standard mode or 0 <= %PLSi.ND <= 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.

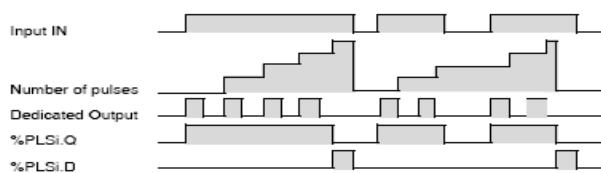
14.1.9.4 RANGE OF PERIODS

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

14.1.9.5 OPERATION

The following is an illustration of the %PLS function block.



14.1.9.6 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 %D0.0.0 or %D0.0.1 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.

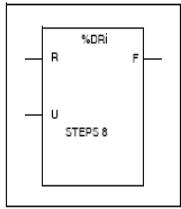
14.1.10 DRUM CONTROLLER FUNCTION BLOCK (%DR)

14.1.10.1 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, internal bits %Mi or AS interface slave output bits %QAx.y.z known as control bits.

14.1.10.2 ILLUSTRATION

The following is an illustration of the drum controller function block.



Drum controller function block

14.1.10.3 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.

14.1.11 DRUM CONTROLLER FUNCTION BLOCK %DRI OPERATION

14.1.11.1 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), memory word (%Mi) or AS interface slave output (%QAx.y.z). 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.

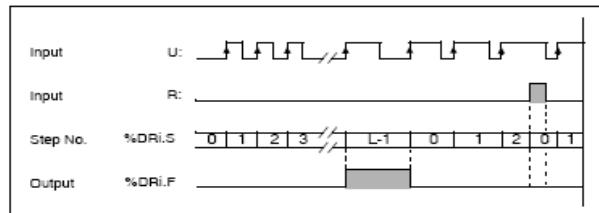
Column	0	1	2	D	O	F
Control bits	%Q0.1	%Q0.3	%Q1.5	%Q0.6	%Q0.5	%Q1.0
0 steps	0	0	1	1	1	0
1 steps	1	0	1	1	0	0
5 steps	1	1	1		0	0
6 steps	0	1	1		0	0
7 steps	1	1	1	1	0	0

14.1.11.2 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.

14.1.11.3 TIMING DIAGRAM

The following diagram illustrates the operation of the drum controller.



14.1.11.4 SPECIAL CASES

The following table contains a list of special cases for drum controller operation.

Special case	Description
Effects of a cold restart (%SD=1)	Resets the drum controller to step 0 (update of control bits).
Effect of a warm restart (%SD=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.

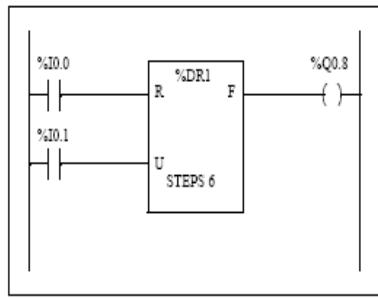
14.1.12 PROGRAMMING AND CONFIGURING DRUM CONTROLLERS

14.1.12.1 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.

14.1.12.2 PROGRAMMING EXAMPLE

The following illustration is a drum controller function block with examples of reversible and non-reversible programming.



Ladder diagram

```

BLK  %DR1
LD  %I0.0
R
LD  %I0.1
U
OUT_BLK
LD  F
ST  %Q0.8
END_BLK

```

14.1.12.3 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.



14.1.13 FAST COUNTER FUNCTION BLOCK (%FC)

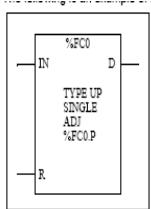
14.1.13.1 INTRODUCTION

The Fast Counter function block (%FC) serves as either an up-counter or a downcounter. It can count the rising edge of discrete 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 accomodate 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.

14.1.13.2 ILLUSTRATION

The following is an example of a Fast Counter function block in single-word mode.



14.1.13.3 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.

14.1.13.4 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.

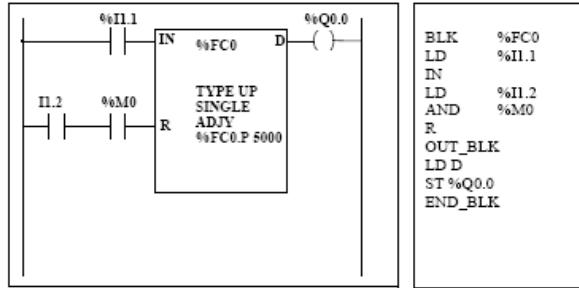
14.1.13.5 OPERATION

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.

14.1.13.6 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.



14.1.13.7 SPECIAL CASES

The following table contains a list of special operating cases for the %FC function block:

Special case	Description
Effect of cold restart (%SD=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.

14.1.14 VERY FAST COUNTER FUNCTION BLOCK (%VFC)

14.1.14.1 INTRODUCTION

The Very Fast Counter function block (%VFC) can be configured by TwidoSuite 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 discrete input up to frequencies of 20kHz in single word or double word computational mode. The TWDLCA•40DRF Compact controllers can accomodate 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).

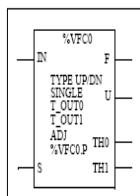
14.1.14.2 DEDICATED I/O ASSIGNMENTS

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 discrete 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.								

14.1.14.3 ILLUSTRATION

Here is a block representation of the Very Fast Counter (%VFC) in single-word mode:



14.1.14.4 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 decides the direction for %VFC0 and %I0.6 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	%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	%VFCi.S1: 0 -> 65535 %VFCi.S1D: 0 -> 4294967295	CM	Read and Write (1)
Frequency Measure Time Base (%VFCi.T)	Configuration item for 100 or 1000 millisecond time base.	1000 or 100	FM	Read and Write (1)

Function	Description	Values	%VFC Use	Run-time Access
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: <ul style="list-style-type: none">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

14.1.14.5 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

14.1.14.6 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.

14.1.14.7 NOTES ON FUNCTION BLOCK OUTPUTS

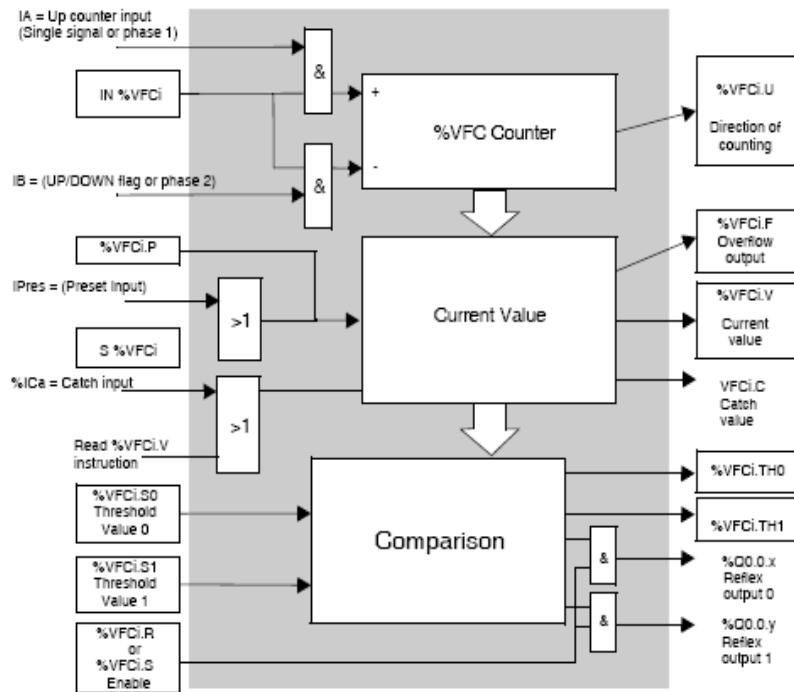
For all functions, the current value is compared to two thresholds (%VFCi.S0 or %VFCi.S0D and %VFCi.S1 or %VFCi.S1D). According to the result of this comparison two bit objects (%VFCi.TH0 and %VFCi.TH1) are set to 1 if the current value is greater or equal to the corresponding threshold, or reset to 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)

14.1.14.8 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.

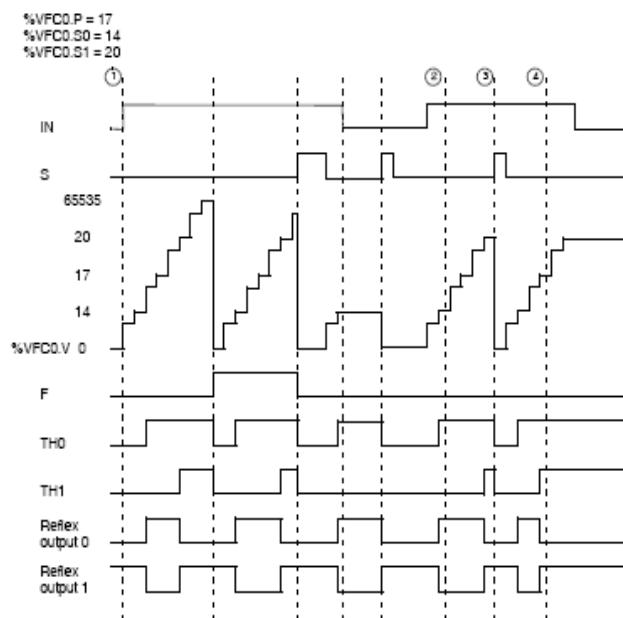
14.1.14.9 SINGLE UP COUNTER OPERATION

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

14.1.14.10 SINGLE DOWN COUNTER OPERATION

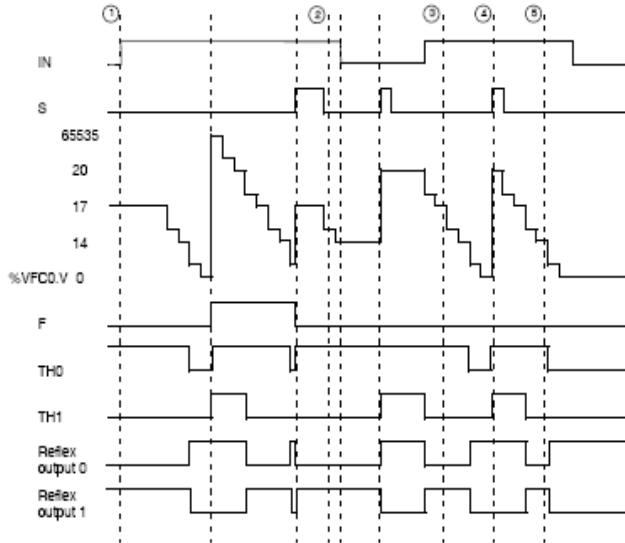
The following is an example of using %VFC in a single down counter mode. The following configuration elements have been set for this example:

%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		
%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

14.1.14.11 UP-DOWN COUNTER OPERATION

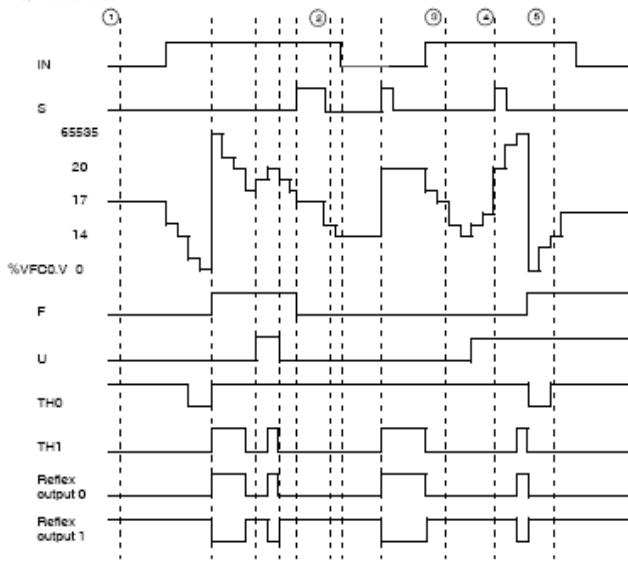
The following is an example of using %VFC in an up-down counter mode. The following configuration elements have been set for this example:

%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

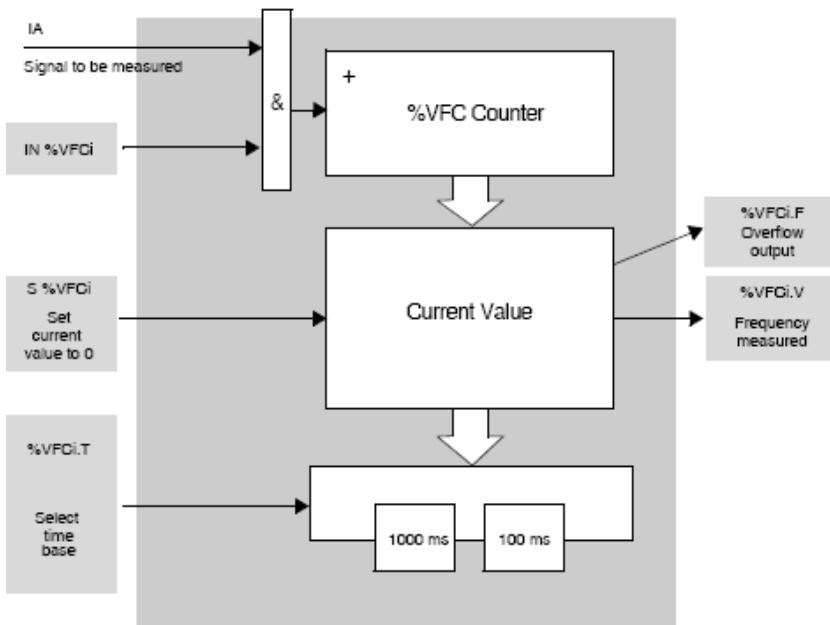
14.1.14.12 FREQUENCY METER FUNCTION DESCRIPTION

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

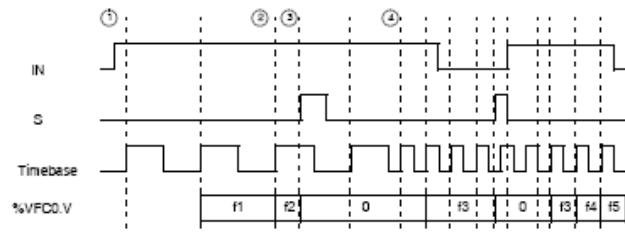
14.1.14.13 FREQUENCY METER FUNCTION DIAGRAM

The following is a frequency meter function diagram:



14.1.14.14 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.

14.1.14.15 SPECIAL CASES

The following table shows a list of special operating of the %VFC function block.

Special case	Description
Effect of cold restart (%SO=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.

14.1.15 TRANSMITTING/RECEIVING MESSAGES - THE EXCHANGE INSTRUCTION (EXCH)

14.1.15.1 INTRODUCTION

A Twido controller can be configured to communicate with Modbus slave devices or can send and/or receive messages in character mode (ASCII).

TwidoSuite 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.

14.1.15.2 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 (%MW*i*: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.

14.1.15.3 SYNTAX

The following is the format for the EXCH instruction:

[EXCH*x* %MW*i*: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 %MW*i*:L are such as *i*+*L* <= 255.

The Twido controller must finish the exchange from the first EXCH*x* 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 .

14.1.16 EXCHANGE CONTROL FUNCTION BLOCK (%MSGX)

14.1.16.1 INTRODUCTION

Note: The "x" in %MSG*x* 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 .

The %MSG*x* 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.

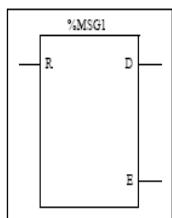
▲ Transmission of priority messages:

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.

14.1.16.2 ILLUSTRATION

The following is an example of the %MSGx function block.



14.1.16.3 PARAMETERS

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	<p>State 1, comm. done, if:</p> <ul style="list-style-type: none"> • End of transmission (if transmission) • End of reception (end character received) • Error • Reset the block <p>State 0, request in progress.</p>
Fault (Error) output	%MSGx.E	<p>State 1, comm. done, if:</p> <ul style="list-style-type: none"> • Bad command • Table incorrectly configured • Incorrect character received (speed, parity, etc.) • Reception table full (not updated) <p>State 0, message length OK, link OK.</p>

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. 571.

14.1.16.4 RESET INPUT ®

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.

14.1.16.5 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.

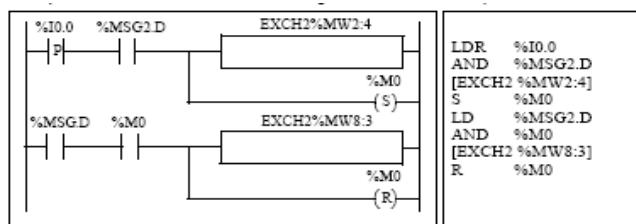
14.1.16.6 COMMUNICATIONS DONE OUTPUT (%MSGX.D)

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.

14.1.16.7 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:



```

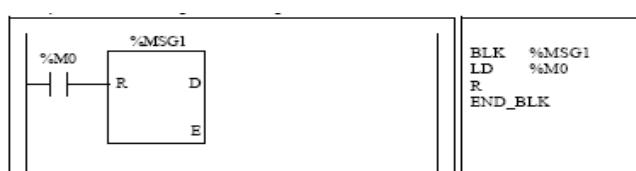
LDR  %I0.0
AND  %MSG2.D
[EXCH2 %MW2:4]
S    %M0
LD   %MSG2.D
AND  %M0
[EXCH2 %MW8:3]
R    %M0

```

14.1.16.8 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:



```

BLK  %MSG1
LD   %M0
R
END_BLK

```

14.1.16.9 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.

14.2 CLOCK FUNCTIONS

14.2.1 AT A GLANCE

14.2.1.1 AIM OF THIS SECTION

This section describes the time management functions for Twido controllers.

14.2.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Clock Functions 286

Schedule Blocks 287

Time/Date Stamping 288

Setting the Date and Time 289

14.2.2 CLOCK FUNCTIONS

14.2.2.1 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 the TwidoSuite Program Configure Data task. 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.

14.2.2.2 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 TwidoSuite by using the Configure RTC option from the TwidoSuite Monitoring Utility accessible via the TwidoSuite Application Launcher.

14.2.3 SCHEDULE BLOCKS

14.2.3.1 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. 564. The RTC option is required for using schedule blocks.

14.2.3.2 PARAMETERS

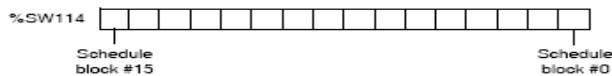
The following table lists parameters for a schedule block:

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.

14.2.3.3 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.

14.2.3.4 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).

14.2.3.5 EXAMPLE

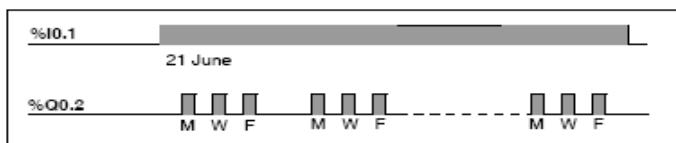
The following table shows the parameters for a summer month spray program example:

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.



14.2.3.6 TIME DATING BY PROGRAM

Date and time are both available in system words %SW50 to %SW53 (see System Words (%SW), p. 571). 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.

14.2.4 TIME/DATE STAMPING

14.2.4.1 INTRODUCTION

System words %SW49 to %SW53 contain the current date and time in BCD format (see Review of BCD Code, p. 393, 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. 571).

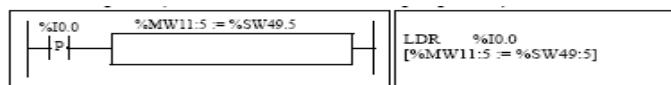
Note: Date and time can also be set by using the optional Operator Display (see Time of Day Clock, p. 285).

14.2.4.2 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).

14.2.4.3 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.

14.2.4.4 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

14.2.4.5 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. 571).

14.2.5 SETTING THE DATE AND TIME

14.2.5.1 INTRODUCTION

You can update the date and time settings by using one of the following methods:

- ▲ TwidoSuite

Use the Set Time dialog box. This dialog box is available from the TwidoSuite Monitoring Utility accessible via the TwidoSuite Application Launcher.

- ▲ System Words

Use system words %SW49 to %SW53 or system word %SW59.

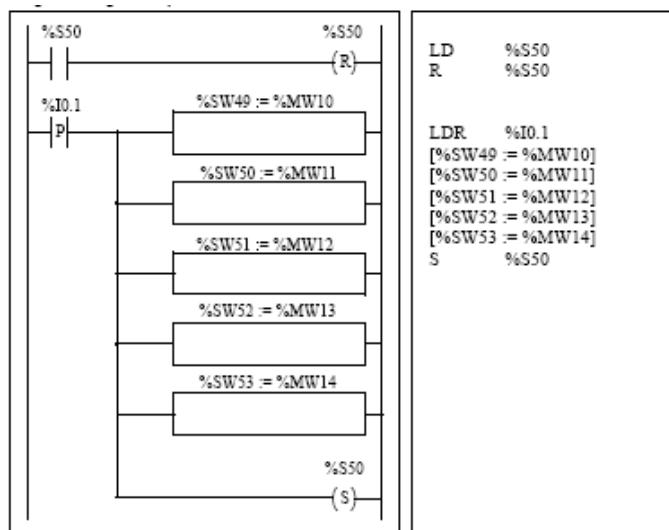
The date and time settings can only be updated when the RTC option cartridge (TWDXCPRTC) is installed on the controller. Note that the TWDLCA•40DRF series of compact controllers have RTC onboard.

14.2.5.2 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. 393) 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

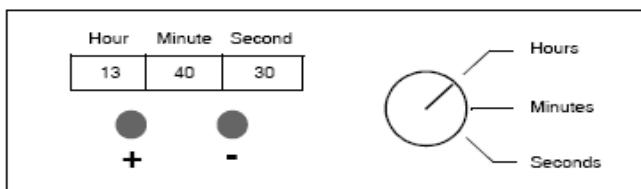
14.2.5.3 USING %SW59

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. 571). %SW59 increments or decrements each of the date and time components on a rising edge.

14.2.5.4 APPLICATION EXAMPLE

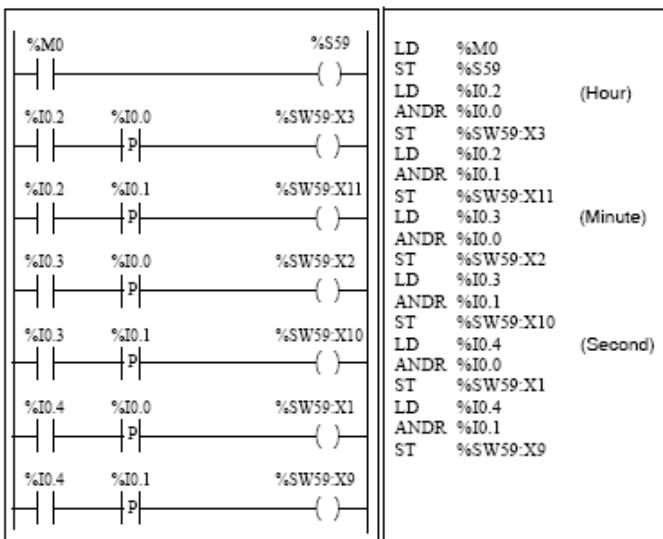
The following front panel is created to modify the hour, minutes, and seconds of the internal clock.



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.



```

LD  %M0
ST  %S59
LD  %I0.2      (Hour)
ANDR %I0.0
ST  %SW59:X3
LD  %I0.2
ANDR %I0.1
ST  %SW59:X11 (Minute)
LD  %I0.3
ANDR %I0.0
ST  %SW59:X2
LD  %I0.3
ANDR %I0.1
ST  %SW59:X10 (Second)
LD  %I0.4
ANDR %I0.0
ST  %SW59:X1
LD  %I0.4
ANDR %I0.1
ST  %SW59:X9

```

14.3 TWIDO PID QUICK START GUIDE

14.3.1 AT A GLANCE

14.3.1.1 OVERVIEW

This section contains information for getting started with the PID control and Auto-Tuning functions available on Twido controllers.

14.3.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Purpose of Document 292

Step 1 - Configuration of Analog Channels Used for Control 293

Step 2 - Prerequisites for PID Configuration 294

Step 3 - Configuring the PID 296

Step 4 - Initialization of Control Set-Up 298

Step 5 - Control Set-Up AT + PID 301

Step 6 - Debugging Adjustments 305

14.3.2 PURPOSE OF DOCUMENT

14.3.2.1 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.

14.3.2.2 THIS DOCUMENT CONTAINS:

This document explains the following steps:

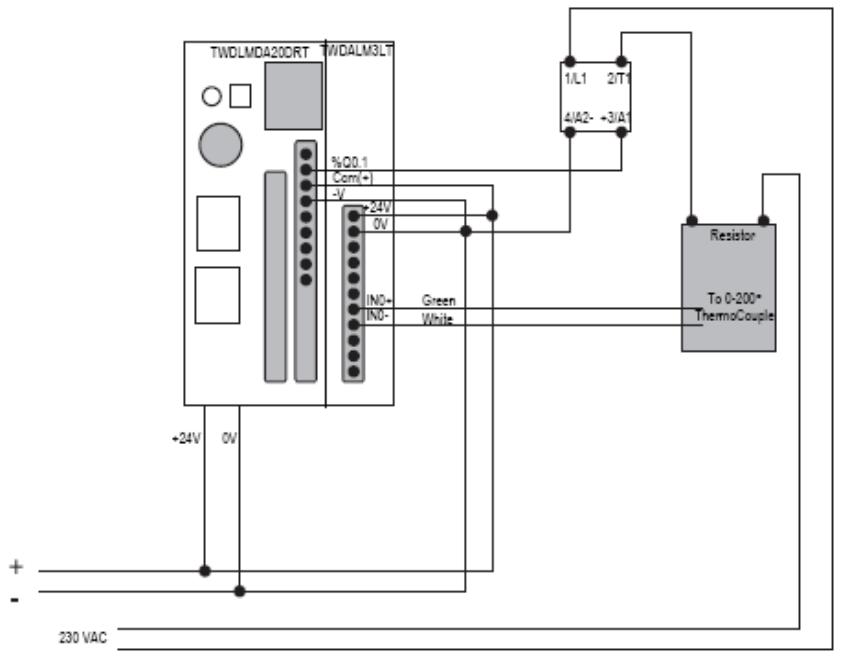
Step	Description
1	Configuration of analog channels used for control
2	Prerequisites for PID configuration
3	PID configuration
4	Initialization of control setup.
5	AT + PID control setup
6	Debugging and adjustments

14.3.2.3 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 Step3 – Configuring the PID, p. 464)

The diagram below shows the experimental setup used in the example:



14.3.3 STEP 1 - CONFIGURATION OF ANALOG CHANNELS USED FOR CONTROL

14.3.3.1 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.

14.3.3.2 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.

14.3.3.3 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.

14.3.3.4 HOW TO CONFIGURE ANALOG INPUT CHANNELS

The following table describes the procedure for configuring the analog channels of the expansion module:

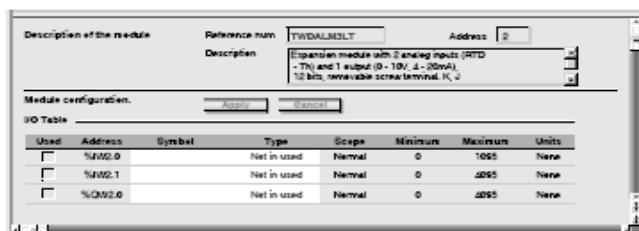
Step	Action
1	Select Describe step from the TwidoSuite interface. See .
2	Display the product catalog and select a module to add to the system description. For example, TWDALM3LT for measuring temperature using a PT100 or Thermocouple.
3	Add the module to the system description (see (See Positioning a Module, TwidoSuite Programming Software, Online Help.)
4	Use the Configuration Editor to set parameters of analog I/O modules that you added as expansion modules when you described the system.
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..

14.3.3.5 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:



14.3.4 STEP 2 - PREREQUISITES FOR PID CONFIGURATION

14.3.4.1 INTRODUCTION

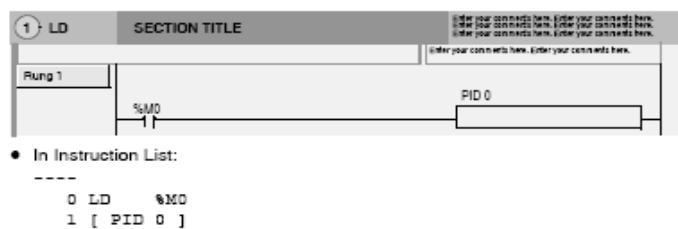
Before configuring the PID, ensure that the following phases have been performed:

Phase	Description
1	PID enabled in the program.
2	Scan period configured

14.3.4.2 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:

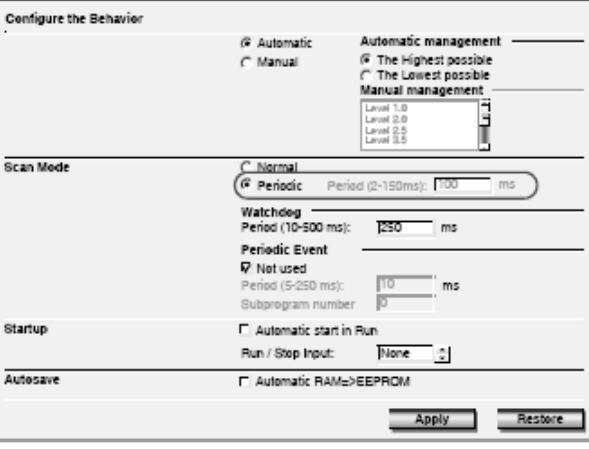
▲ In Ladder:



Note: Ensure that you use correct syntax: Check that there is a space between "PID" and the PID number (e.g. PID<space>0).

14.3.4.3 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	Use the Program → Configure → Configure the Behavior task to configure the controller Scan Mode settings.
2	Check the Periodic box.
3	Set the cycle time as shown in the screen below: 

14.3.5 STEP3 – CONFIGURING THE PID

14.3.5.1 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.

14.3.5.2 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).

14.3.5.3 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 K_p, T_i, T_d (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 K_p, T_i, T_d (PID tab) and the type of PID action (Output tab). At the end of the sequence the PID stops and waits. The gain values K_p, T_i, T_d (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.

14.3.5.4 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	Select the Program → Configure → Configure the Data task on the TwidoSuite interface. Result: The default software configuration window appears.
2	Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame.
3	Select the desired PID# from the PID table.

Step	Action
4	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. <p>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.</p>

14.3.5.5 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.

14.3.5.6 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 Operating Modes, p. 464). 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.

14.3.5.7 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).

14.3.5.8 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.

14.3.5.9 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. 498 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.

14.3.5.10 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	<p>In the Output tab, enter the selection from the Action drop-down list. This selection depends on the configured system:</p> <ul style="list-style-type: none"> • Direct action: the controller output decreases as the variation value (setpoint - measurement) increases (cold controller). • Inverse action: Direct action: the controller output decreases as the variation value (setpoint - measurement) increases (hot controller). <p>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.</p>
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	<p>Set the Manual mode operating mode. The drop-down list offers several choices:</p> <ul style="list-style-type: none"> • 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). <p>In the example: Here we select %M2 to activate the choice, and %MW18 to adjust the value of the manual setpoint.</p>
4	<p>Adjust the Discrete 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.</p> <p>Important: When using the PWM function, enter a memory address (%MWxx) in this field.</p>
5	<p>Set the PWM output if required by the system:</p> <ol style="list-style-type: none"> 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 TWDLMDA200RT 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.

14.3.6 STEP 4 - INITIALIZATION OF CONTROL SET-UP

14.3.6.1 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.

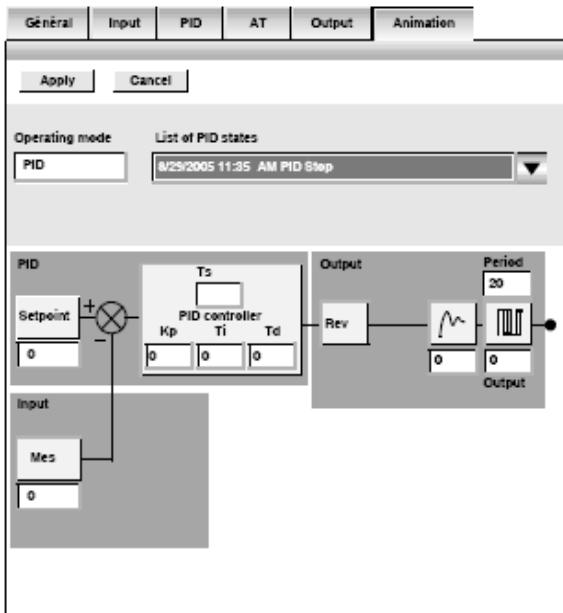
Note: Before switching the controller to RUN mode, check that the machine's operating conditions allow this for the rest of the application.

14.3.6.2 PROCEDURE

The steps below must be followed to initialize control set-up:

Step	Action
1	<p>Create an animation table containing the main objects needed for diagnostics.</p> <p>In the example:</p> <ul style="list-style-type: none"> ● %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	<p>Check the consistency of the value measured in the %IW1.0 field.</p> <p>In the example:</p> <ol style="list-style-type: none"> 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). <p>Note: This test is quite important, as the operation of the controller depends essentially on the accuracy of the measurement.</p> <ol style="list-style-type: none"> 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): <ul style="list-style-type: none"> • 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	<p>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.</p> <p>To set the controller to Manual mode:</p> <ol style="list-style-type: none"> 1. Switch the controller to RUN mode. 2. Enter the memory addresses with the following values in the animation table: <ul style="list-style-type: none"> • %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. <p>This setpoint value can be selected several times, on condition that the system be left to return to its initial state.</p> <p>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.</p>
4	<p>Check that the controller is RUN mode.</p> <p>(%MO: controller validation = 1, to be entered in the animation table.)</p>
5	<p>Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame.</p> <p>Select the desired PID# from the PID table.</p>

Step	Action
6	<p>Activate the Animation tab for the required PID number and check that the animation matches the screen below:</p>  <p>Note: The screens of the PID controller are only refreshed if the controller is enabled (and API set to RUN).</p>
7	<p>Activate the Trace tab for the required PID number, then:</p> <ol style="list-style-type: none"> Set the time elapse drop-down list to 15 min to see a trace of the measurement signal's progress. 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	<p>Set the time elapse scroll list to 15min to see a trace of the measurement signal's progress.</p> <p>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).</p>

Step	Action
9	<p>If we see that the actuator is not controlled, check the output circuit:</p> <ul style="list-style-type: none"> For an analog output, check the output voltage or current from the analog card. For a PWM output, check: <ul style="list-style-type: none"> the LED of the output concerned is lit (%Q0.1, in this example), the wiring of the supplies and OV circuit for the TWDLMDA20DRT base outputs, the actuator power supply.
10	<p>Close the PID display screen and stop the manual mode by entering the following values in the animation table:</p> <ul style="list-style-type: none"> %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.

14.3.7 STEP 5 - CONTROL SET-UP AT + PID

14.3.7.1 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.

14.3.7.2 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	<p>Enter or check the memory addresses with the following values in the animation table:</p> <ul style="list-style-type: none"> • %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 in AT), • %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 (%KWD 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	<p>Set the Time of the Twido controller scanning period so that the Sampling period (T_s) value of the PID controller is an exact multiple.</p> <p>Note: For further details on how to determine the sampling period, see <i>Auto-Tuning Requirements</i>, p. 515 and <i>Methods for Determining the Sampling Period (T_s)</i>, p. 516.</p>
4	Check that the controller is RUN mode.
5	<p>Enter the memory bit %M0.</p> <p>%M0: controller validation = 1 in the animation table.</p>

Step	Action
6	Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame. Select the desired PID# from the PID table.
7	<p>Activate the Animation tab for the required PID number and check that the animation matches the screen below:</p> <p>Note: The screens of the PID controller are only refreshed if the controller is enabled (and API set to RUN).</p>
8	<p>Click on Trace button and wait for the system to start AT.</p> <p>Trace</p> <p>Note: The waiting time may last for 10-20 minutes before the AT procedure changes.</p>

14.3.7.3 STORAGE OF CALCULATED KP, TI AND TD COEFFICIENTS

Once the Auto-Tuning sequence is complete, the memory words assigned to the K_p, T_i and T_d 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.

14.3.7.4 REPETITION OF AT

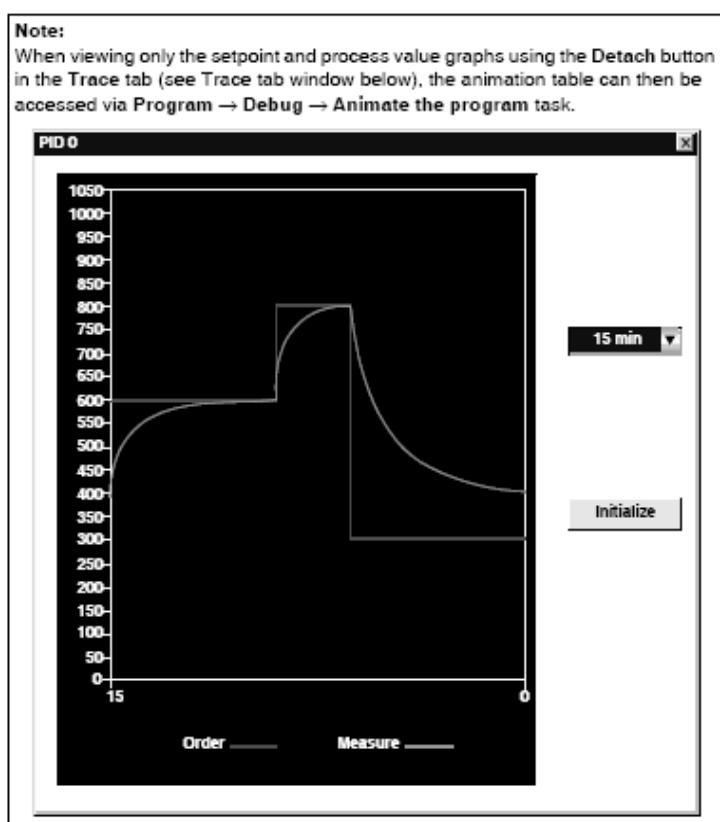
The Auto-Tuning sequence is repeated on every switch to RUN or cold start (%S0).

You should therefore test the diagnostics words using the program what to do in the event of a restart.

14.3.8 STEP 6 - DEBUGGING ADJUSTMENTS

14.3.8.1 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.



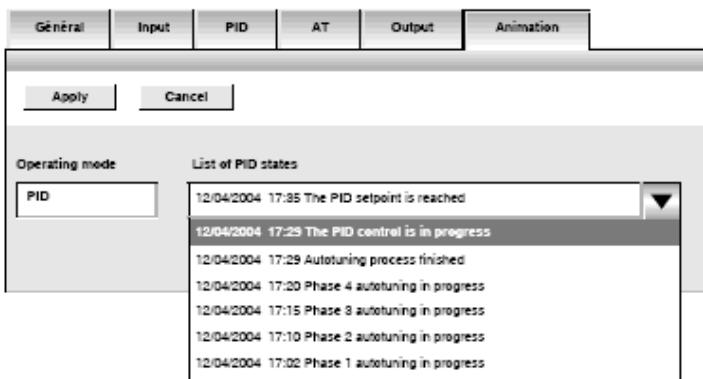
14.3.8.2 RETURNING TO PID SCREENS

To return to the PID controller screens without losing the graph trace history, proceed as follows:

Step	Action
1	Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame. Select the desired PID# from the PID table.
2	Click the Animation tab.

14.3.8.3 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 TwidoSuite are in online mode.

14.4 PID FUNCTION

14.4.1 AT A GLANCE

14.4.1.1 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. 457.

14.4.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Overview 306

Principal of the Regulation Loop 308

Development Methodology of a Regulation Application 309

Compatibilities and Performances	310
Detailed characteristics of the PID function	311
How to access the PID configuration	313
PID Screen Elements of PID function	314
General tab of PID function	317
Input tab of the PID	319
PID tab of PID function	321
AT tab of PID function	322
Output tab of the PID	325
How to access PID debugging	327
Animation tab of PID function	328
Trace screen of PID function	329
PID States and Errors Codes	330
PID Tuning With Auto-Tuning (AT)	332
PID parameter adjustment method	337
Role and influence of PID parameters	339
Appendix 1: PID Theory Fundamentals	341
Appendix 2: First-Order With Time Delay Model	342

14.4.2 OVERVIEW

14.4.2.1 GENERAL

The PID regulation function is a TwidoSuite programming language function.

It allows programming of PID regulation loops on controllers compatible with TwidoSuite 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:

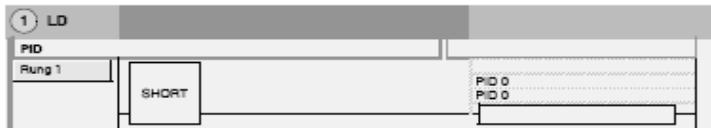
- ▲ Configuration

✗ 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.

The correct syntax when writing a PID instruction is: PID<space>n, when n is the PID number.

Example of a program line in Ladder Language:



Note: In any given Twido automation application, the maximum number of configurable PID functions is 14.

14.4.2.2 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.

14.4.3 PRINCIPAL OF THE REGULATION LOOP

14.4.3.1 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 TwidoSuite 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:

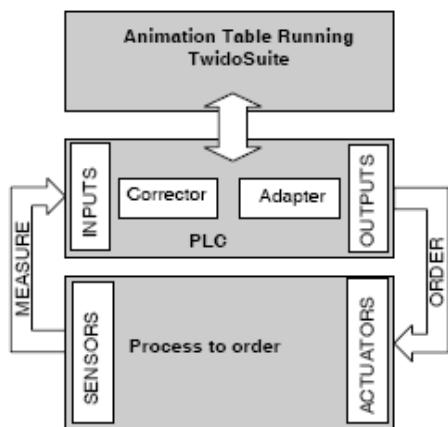
- ✗ The measurement sampled by the input module

- ▲ 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.

14.4.3.2 ILLUSTRATION

The following diagram schematizes the principal of a regulation loop.

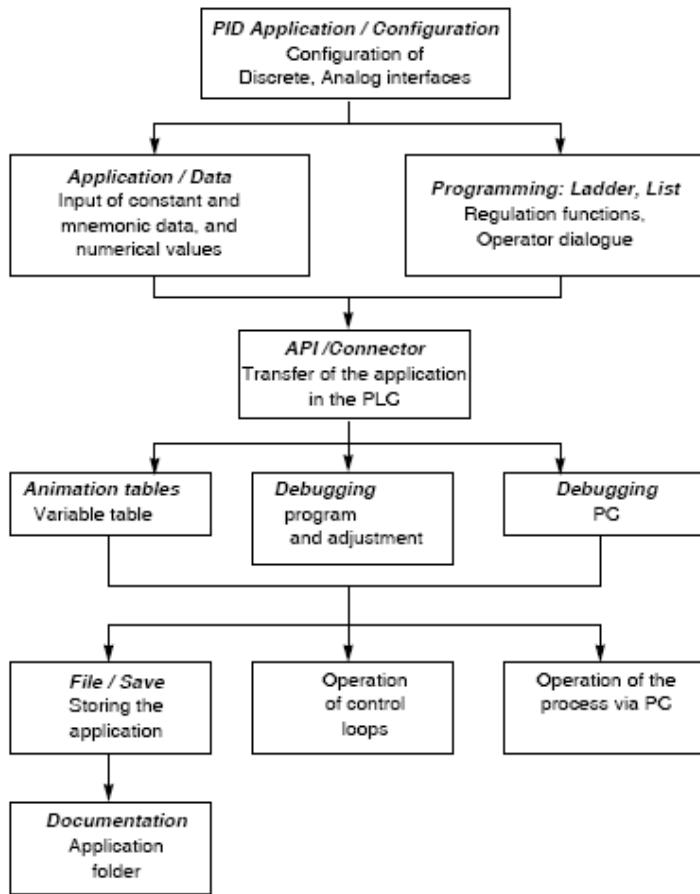


14.4.4 DEVELOPMENT METHODOLOGY OF A REGULATION APPLICATION

14.4.4.1 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.



14.4.5 COMPATIBILITIES AND PERFORMANCES

14.4.5.1 AT A GLANCE

The Twido PID function is a function that is available for controllers compatible with TwidoSuite 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.

14.4.5.2 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 the TwidoSuite software.

14.4.5.3 PERFORMANCE

The PID regulation loops have the following performances:

Description	Time
Loop execution time	0.4 ms

14.4.6 DETAILED CHARACTERISTICS OF THE PID FUNCTION

14.4.6.1 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 discrete 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).

14.4.6.2 DETAILS OF AVAILABLE FUNCTIONS

The following table indicates the different functions available and their scale:

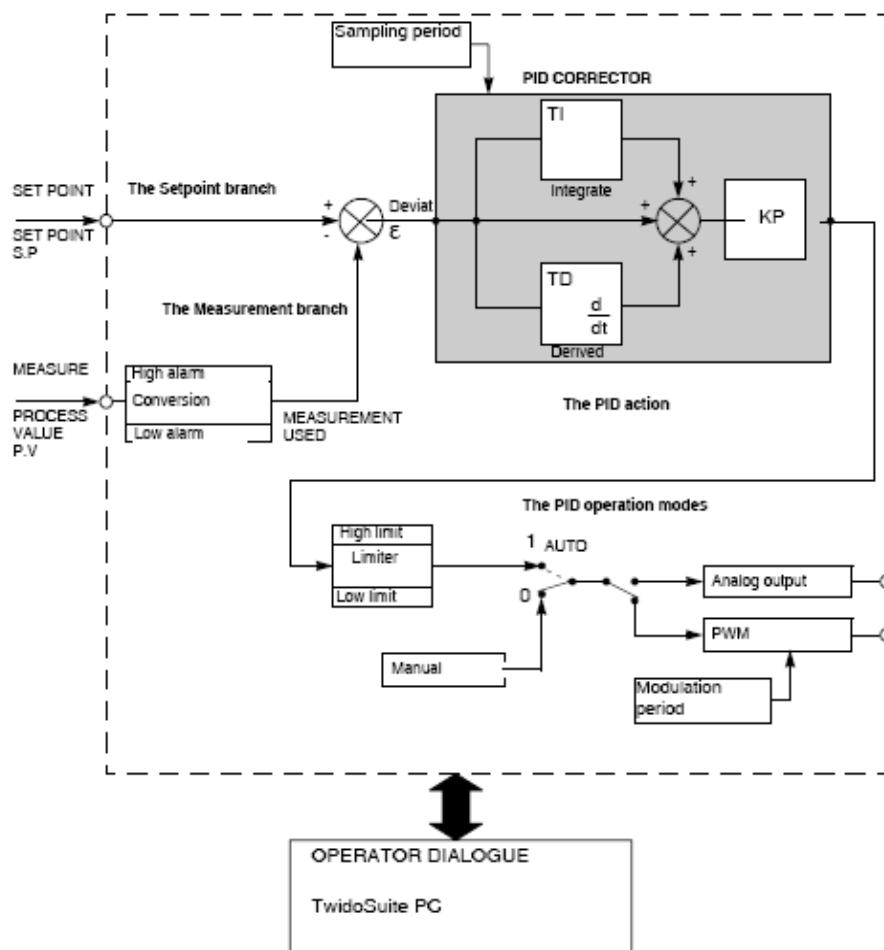
Function	Scale and comment
PWM output	Using a timebase of 0.1 seconds, its value is between 1 and 500. This corresponds to a modulation period of between 0.1 and 50 seconds.
Analog output	Value between 0 and +10000
High level alarm on process variable	This alarm is set after conversion. It is set to 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.

Function	Scale and comment
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.

14.4.6.3 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.

14.4.7 HOW TO ACCESS THE PID CONFIGURATION

14.4.7.1 AT A GLANCE

The following paragraphs describe how to access the PID configuration screens on Twido controllers.

14.4.7.2 PROCEDURE

The following table describes the procedure for accessing the PID configuration screens:

Step	Action
1	Check that you are in offline mode.
2	Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame.
3	Select the desired PID# from the PID table. (See PID Selection Table of the PID Function, p. 488) Result: The PID configuration window opens and displays the General (See General tab of PID function, p. 491) tab by default.

14.4.8 PID SCREEN ELEMENTS OF PID FUNCTION

14.4.8.1 AT A GLANCE

The PID configuration window permits to:

- ▲ Configure each TWIDO PID (in offline mode),
- ▲ Debug each TWIDO PID (in online mode).

This section describes the PID Screen Elements, including:

- ▲ Access to the PID Configuration Screen,
- ▲ PID Selection Table of the PID Function,
- ▲ PID tabs of the PID Function,
- ▲ PID Trace.

14.4.8.2 ACCESS TO THE PID CONFIGURATION SCREEN

To access the PID configuration window:

If...	Then ...	Outcome
You are in online mode.	Select Program → Debug → Monitor software configuration → Advanced objects → PID.	You will go to the Animation tab and will have access to the debugging and adjustment parameters.
You are in offline mode.	Select Program → Configure → Configure the data → Advanced objects → PID.	You will go to the General tab by default and will have access to the configuration parameters.

14.4.8.3 PID SELECTION TABLE OF THE PID FUNCTION

The table below is used to select the PID you wish to Configure/Debug.

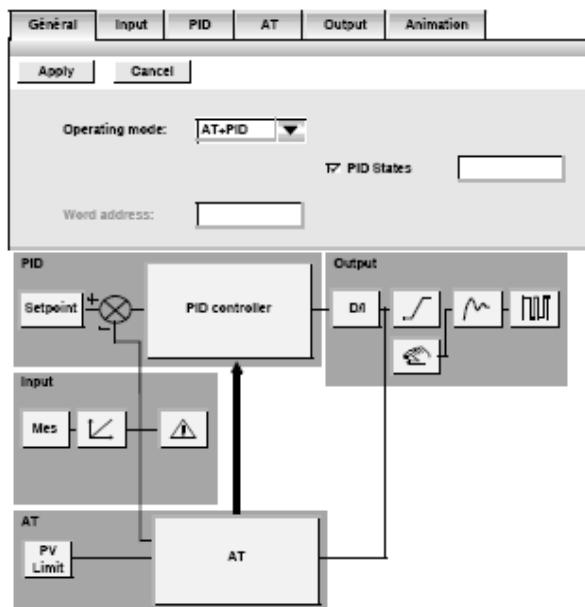
All		
Used	Address	Configured
<input type="checkbox"/>	PID 0	<input type="checkbox"/>
<input type="checkbox"/>	PID 1	<input type="checkbox"/>
<input type="checkbox"/>	PID 2	<input type="checkbox"/>
<input type="checkbox"/>	PID 3	<input type="checkbox"/>
<input type="checkbox"/>	PID 4	<input type="checkbox"/>
<input type="checkbox"/>	PID 5	<input type="checkbox"/>
<input type="checkbox"/>	PID 6	<input type="checkbox"/>
<input type="checkbox"/>	PID 7	<input type="checkbox"/>
<input type="checkbox"/>	PID 8	<input type="checkbox"/>
<input type="checkbox"/>	PID 9	<input type="checkbox"/>
<input type="checkbox"/>	PID 10	<input type="checkbox"/>
<input type="checkbox"/>	PID 11	<input type="checkbox"/>
<input type="checkbox"/>	PID 12	<input type="checkbox"/>
<input type="checkbox"/>	PID 13	<input type="checkbox"/>

The table below describes the settings that you may define.

Field	Description
Address	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.
Used	Read only, this box is checked if the PID with corresponding number is used in the application program
Sort options (combo box)	Select the corresponding combo box option whether you wish to display All, or only Used or Unused PID in the PID selection table.
Note:	You must first complete the current PID configuration before switching to another PID or performing any another software task.

14.4.8.4 PID TABS OF THE PID FUNCTION

The PID tabs allow you to to configure the PID parameters. The screen below shows the tabs of The PID.



The table below describes the tabs of the PID.

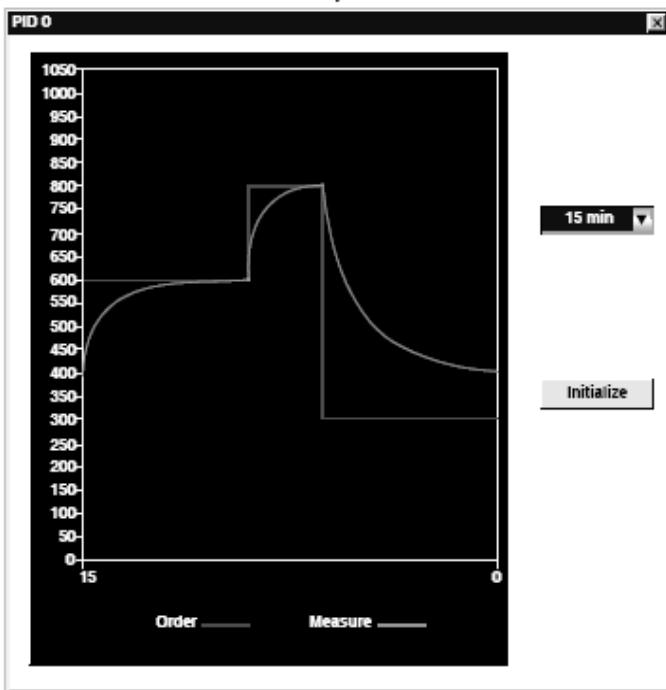
Field	Description
General Tab	Specify the PID General parameters, see <i>General tab of PID function, p. 491</i>
Input Tab	Specify the PID Input parameters, see <i>Input tab of the PID, p. 494</i>
PID Tab	Specify the PID internal parameters, see <i>PID tab of PID function, p. 496</i>
AT Tab	Specify the AT parameters, see <i>AT tab of PID function, p. 498</i>
Output Tab	Specify the PID Output parameters, see <i>Output tab of the PID, p. 503</i>
Animation Tab	View/Debug the PID, see <i>Animation tab of PID function, p. 507</i>

Note: In some cases, the grayed-out tabs and fields may not be accessible for any of the two reasons listed below:

- ▲ 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.

14.4.8.5 PIDTRACE OF THE PID FUNCTION

The PID trace button Trace allows you to view the PID control.



This tab allows you to view PID operation and to make adjustments to the way it behaves, see Trace screen of PID function, p. 509.

14.4.9 GENERAL TAB OF PID FUNCTION

14.4.9.1 AT A GLANCE

Select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame.

Select the desired PID# from the PID table.

The PID configuration window allows you to:

- ▲ configure each TWIDO PID (in online mode),
- ▲ debug each TWIDO PID (in offline mode),

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.

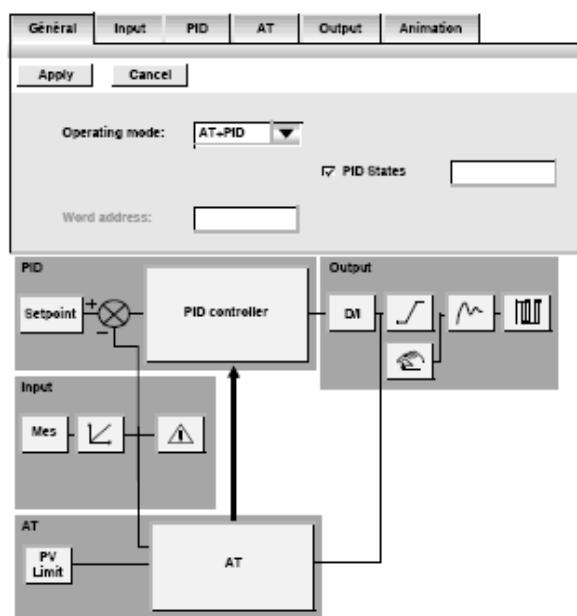
Note: In some cases, the grayed-out tabs and fields may not be accessible for any of the two reasons listed below: The "PID only" operating mode is selected, which prevents access to the AT tab parameters that are no longer needed.

- ✗ 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.

14.4.9.2 GENERAL TAB OF THE PID FUNCTION

The screen below is used to enter the general PID parameters.



14.4.9.3 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • %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. 511.)
Diagram	The diagram allows you to view the different possibilities available for configuring your PID.

14.4.10 INPUT TAB OF THE PID

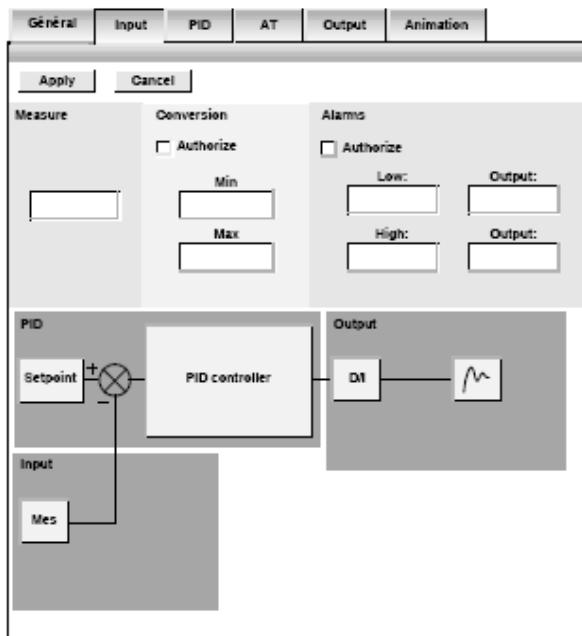
14.4.10.1 AT A GLANCE

The tab is used to enter the PID input parameters.

Note: It is accessible in offline mode.

INPUT TAB OF THE 11.4.10.2 ON

The screen below is used to enter the PID input parameters.



14.4.10.3 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.

14.4.11 PID TAB OF PID FUNCTION

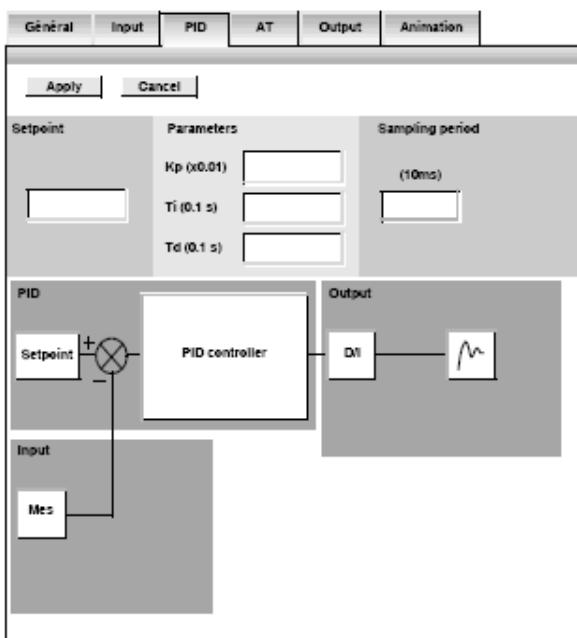
14.4.11.1 AT A GLANCE

The tab is used to enter the internal PID parameters.

Note: It is accessible in offline mode.

14.4.11.2 PID TAB OF THE PID FUNCTION

The screen below is used to enter the internal PID parameters.



14.4.11.3 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 (%KWO 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 (%KWO 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 (%KWO 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 (%KWO 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 (%KWO 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

14.4.12 AT TAB OF PID FUNCTION

14.4.12.1 AT A GLANCE

The setting of correct PID parameters may be tedious, time-consuming and errorprone. 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.

14.4.12.2 AT REQUIREMENTS

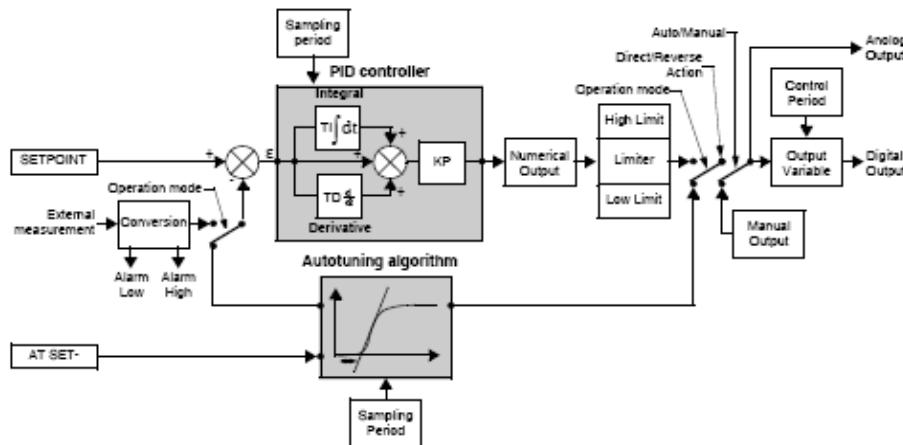
PID Auto-tuning is particularly suited for temperature control processes.

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.)

14.4.12.3 AT OPERATING PRINCIPLE

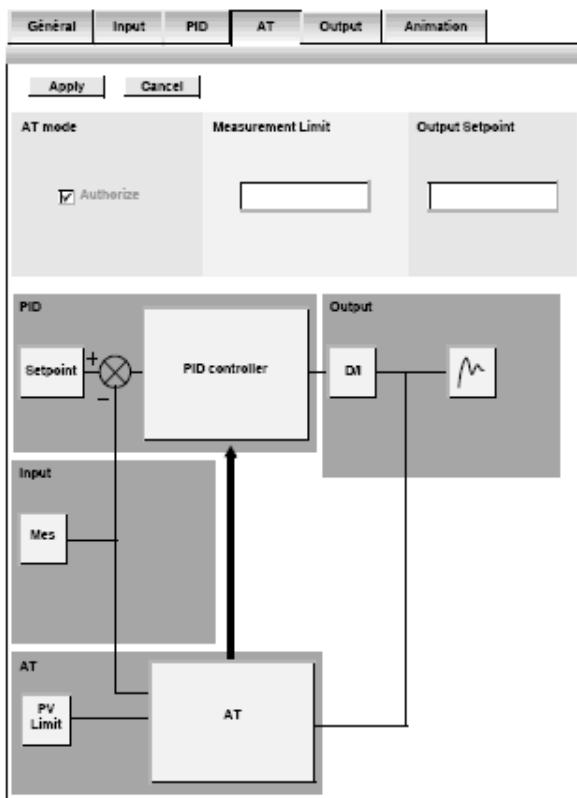
The following diagram describes the operating principle of the AT function and how it interacts with the PID loops:



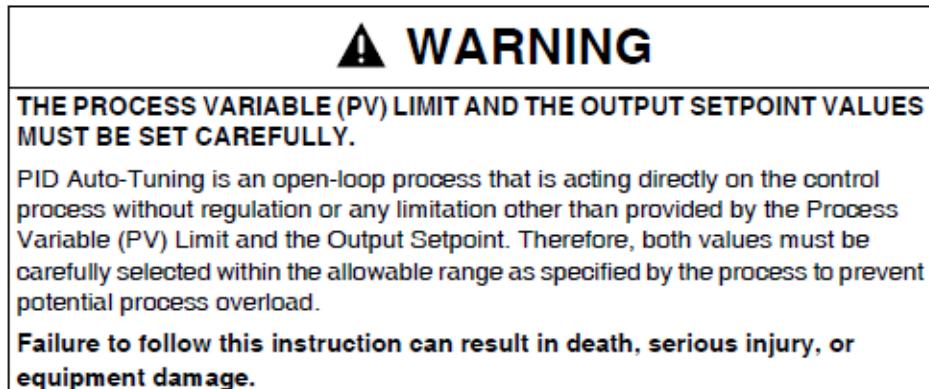
14.4.12.4 AT TAB OF THE PID FUNCTION

The screen below is used to enable/disable the AT function and enter the AT parameters.

Note: It is accessible in offline mode only.



14.4.12.5 DESCRIPTION



The table below describes the settings that you may define.

Field	Description
Authorize	<p>Check this box if you wish to enable the AT mode.</p> <p>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:</p> <ul style="list-style-type: none"> • If you set the Operating mode to PID+AT or AT from the General tab (see <i>General tab of PID function, p. 49</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. <p>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.</p>
Process Variable (PV) Limit	<p>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.</p> <p>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.</p> <p>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.</p>
AT Output setpoint	<p>Specify the AT output value here. This is the value of the step-change that is applied to the process.</p> <p>This value can be an internal word (%MW0 to %MW2999), an internal constant (%KW0 to %KW255) or a direct value.</p> <p>This value must therefore be between 0 and 10000.</p> <p>Note: The AT Output Setpoint must always be larger than the output last applied to the process.</p>

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.

14.4.12.6 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 TwidoSuite online mode only.

14.4.13 OUTPUT TAB OF THE PID

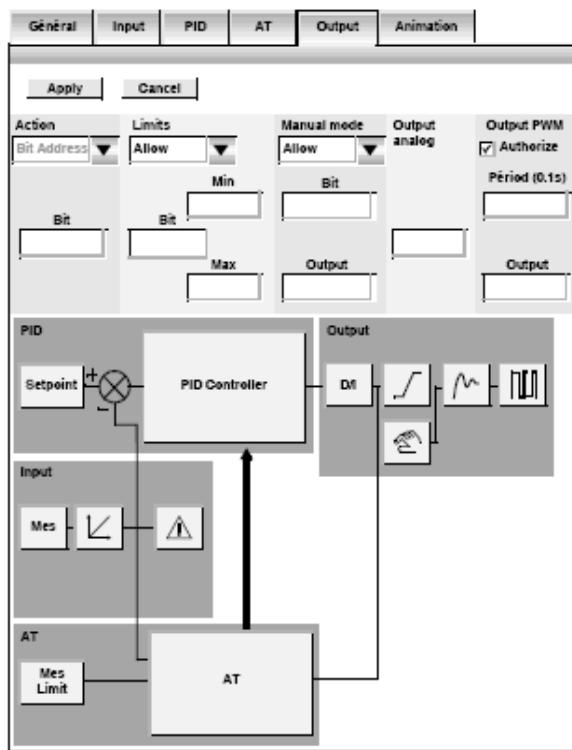
14.4.13.1 AT A GLANCE

The tab is used to enter the PID output parameters.

Note: It is accessible in offline mode.

14.4.13.2 OUTPUT TAB OF THE PID FUNCTION

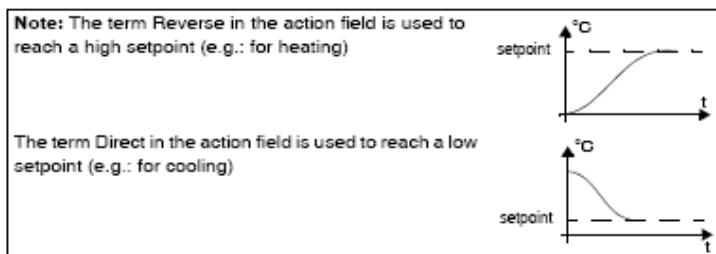
The screen below is used to enter the internal PID parameters.



14.4.13.3 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 (%MO 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 (%MO 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 (%MO 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 %OW-type (%OWx.0).
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 (%MO 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.



14.4.14 HOW TO ACCESS PID DEBUGGING

14.4.14.1 AT A GLANCE

The following paragraphs describe how to access the PID debugging screens on TWIDO controllers.

14.4.14.2 PROCEDURE

The following table describes the procedure for accessing the PID debugging screens:

Step	Action
1	Check that you are in online mode.
2	In the monitor software configuration screen, select Advanced Objects from the Object Category (See Object Category Frame, TwidoSuite Programming Software, Online Help) frame and choose PID from the Objects Type (See Object Type Frame, TwidoSuite Programming Software, Online Help) frame.
3	Select the desired PID# from the PID table. Note: You can also double-click the PID graphic element in the ladder rung to access the PID configuration window.
4	Click the Animation tab. Result: The PID configuration window opens and displays the Animation (See Animation tab of PID function, p. 507) tab by default.

14.4.15 ANIMATION TAB OF PID FUNCTION

14.4.15.1 AT A GLANCE

The tab is used to debug the PID.

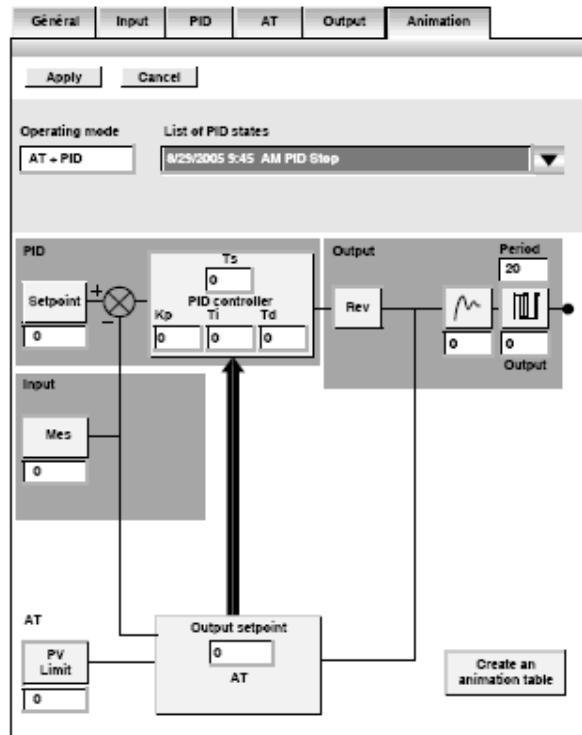
The diagram depends on the type of PID control that you have created. Only configured elements are shown.

The display is dynamic. Active links are shown in red and inactive links are shown in black.

Note: It is accessible in online mode.

14.4.15.2 ANIMATION TAB OF PID FUNCTION

The screen below is used to view and debug the PID.



14.4.15.3 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.

14.4.16 TRACE SCREEN OF PID FUNCTION

14.4.16.1 AT A GLANCE

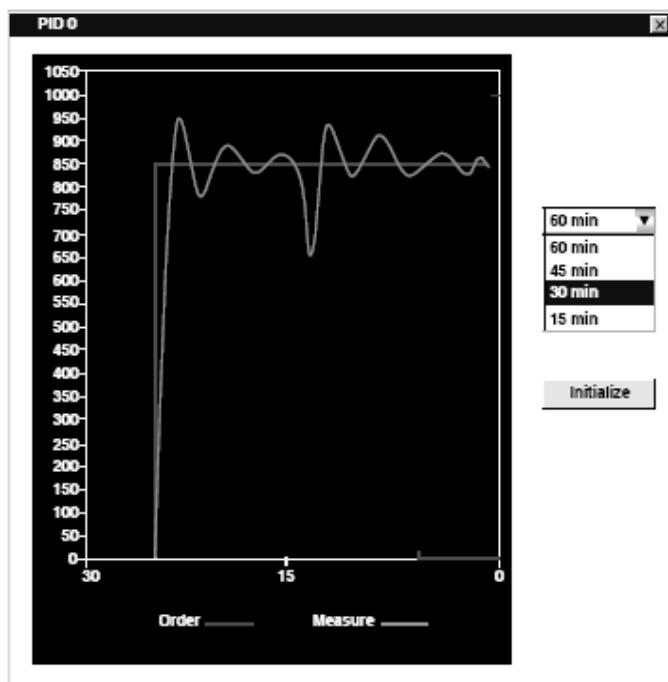
This screen allows you to view PID operation and to make adjustments to the way it behaves.

The graphs begin to be traced as soon as the debug window is displayed.

Note: It is accessible in online mode.

14.4.16.2 ANIMATION TAB OF PID FUNCTION

The screen below is used to view the PID control.



14.4.16.3 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.

14.4.17 PID STATES AND ERRORS CODES

14.4.17.1 AT A GLANCE

In addition to the List of PID States available from the Animation dialog box (see Animation tab of PID function, p. 507) 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. 491.

14.4.17.2 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.

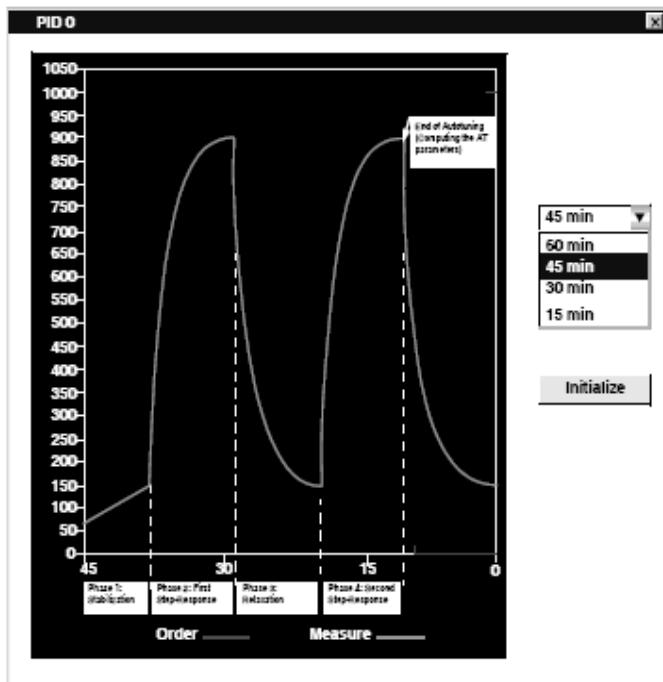
14.4.17.3 PID STATE MEMORY WORD

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

14.4.17.4 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.
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.

14.4.17.5 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

14.4.17.6 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 discrete 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

14.4.18 PID TUNING WITH AUTO-TUNING (AT)

14.4.18.1 OVERVIEW OF PID TUNING

The PID control function relies on the following three user-defined parameters: Kp, Ti and Td. PID tuning aims at determining these process parameters accurately to provide optimum control of the process.

14.4.18.2 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.

14.4.18.3 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.)

14.4.18.4 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 K_p , T_i and T_d (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 K_p , T_i and T_d 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.

14.4.18.5 METHODS FOR DETERMINING THE SAMPLING PERIOD (TS)

As will be explained in the two following sections (see Appendix 1: PID Theory Fundamentals, p. 529 and Appendix 2: First-Order With Time Delay Model, p. 531), the sampling period (T_s) 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 (T_s) 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.

14.4.18.6 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)

14.4.18.7 USING THE PROCESS RESPONSE CURVE METHOD

To determine the sampling period (T_s) 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.
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... 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: <ol style="list-style-type: none"> 1. Compute the process variable output at 63% rise ($S_{(63\%)}$) by using the following formula: $S_{(63\%)} = S_{(\text{initial})} + (S_{(\text{ending})} - S_{(\text{initial})}) \times 63\%$ 2. Find out graphically the time abscissa ($t_{(63\%)}$) that corresponds to $S_{(63\%)}$. 3. Find out graphically the initial time ($t_{(\text{initial})}$) that corresponds the start of the process response rise. 4. Compute the time constant (τ) of the control process by using the following relationship: $\tau = t_{(63\%)} - t_{(\text{initial})}$
9	Compute the sampling period (T_s) based on the value of (τ) that you have just determined in the previous step, using the following rule: $T_s = \tau / 75$ Note: The base unit for the sampling period is 10ms. Therefore, you should round up/down the value of T_s to the nearest 10ms.
10	Select Program > Configure the Behavior to set the Scan Mode parameters and proceed as follows: <ol style="list-style-type: none"> 1. Set the Scan mode of the Twido PLC to Periodic. 2. Set the Scan Period so that the sampling period (T_s) is an exact multiple of the scan period, using the following rule: Scan Period = T_s / 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].

14.4.18.8 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	<p>The following information is obtained directly from the graphical analysis of the response curve, as shown in the figure below:</p> <p>where</p> <ul style="list-style-type: none"> • $S_{[0]} = \text{initial value of process variable} = 260$ • $S_{[e]} = \text{ending value of process variable} = 660$ • $S_{[63\%]} = \text{process variable at } 63\% \text{ rise} = S_{[0]} + (S_{[e]} - S_{[0]}) \times 63\%$ $= 260 + (660 - 260) \times 63\%$ $= 512$ • $t = \text{time constant}$ $= \text{time elapsed from the start of the rise till } S_{[63\%]} \text{ is reached}$ $= 9 \text{ min } 30 \text{ s} = 570 \text{ s}$
7	The sampling period (T_s) is determined using the following relationship: $T_s = \pi/75$ $= 570/75 = 7.6 \text{ s (7600 ms)}$
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: $2 \text{ ms} \leq \text{Scan Period} \leq 150 \text{ ms.}$)

14.4.18.9 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 K_p , T_i and T_d 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: <ul style="list-style-type: none">• Auto-tuning completes successfully: You may continue to Step 9.• Auto-tuning fails: This means the current guess for the sampling period (T_s) is not correct. Try a new T_s 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 T_s guess:<ul style="list-style-type: none">• AT ends with the error message "The computed time constant is negative!": This means the sampling period T_s is too large. You should decrease the value of T_s to provide as new guess.• AT ends with the error message "Sampling error!": this means the sampling period T_s is too small. You should increase the value of T_s to provide as new guess.
10	You may now view the PID control parameters (K_p , T_i and T_d) 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 K_p , T_i and T_d control parameters.

14.4.18.10 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).

14.4.18.11 LIMITATIONS ON USING THE AUTOTUNING AND THE PID CONTROL

The auto-tuning is best suited for processes whose time constant (τ) and delaytime (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.

14.4.18.12 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: <ul style="list-style-type: none">• 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. 515.
Autotuning error: error calculating K _p	The AT algorithm has failed (no convergence). <ul style="list-style-type: none">• Disturbances on the process while autotuning have caused a distortion of the process static gain evaluation.• The process variable transient response is not big enough for the autotuning to determine the static gain.• A combination of the above possible causes may effect on the process.	<p>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. Try to modify<ul style="list-style-type: none">• the output setpoint,• the sampling period.Make sure there is no process disturbance while autotuning is in progress.</p>
Autotuning error: time constant over delay ratio > 20	$\tau/\delta > 20$	PID regulation is no longer guaranteed. For more details, please check out <i>PID Tuning With Auto-Tuning (AT)</i> , p. 515.
Autotuning error: time constant over delay ratio < 2	$\tau/\delta < 2$	PID regulation is no longer guaranteed. For more details, please check out <i>PID Tuning With Auto-Tuning (AT)</i> , p. 515.
Autotuning error: the limit for K _p has been exceeded	Computed value of static gain (K _p) is greater than 10000.	Measurement sensitivity of some application variables may be too low. The applications measurement range must be rescaled within the [0-10000] interval.
Autotuning error: the limit for T _i has been exceeded	Computed value of integral time constant (T _i) is greater than 20000.	Computational limit is reached.
Autotuning error: the limit for T _d has been exceeded	Computed value of derivative time constant (T _d) is greater than 10000.	Computational limit is reached.

14.4.19 PID PARAMETER ADJUSTMENT METHOD

14.4.19.1 INTRODUCTION

Numerous methods to adjust the PID parameters exist, we suggest Ziegler and

Nichols which have two variants:

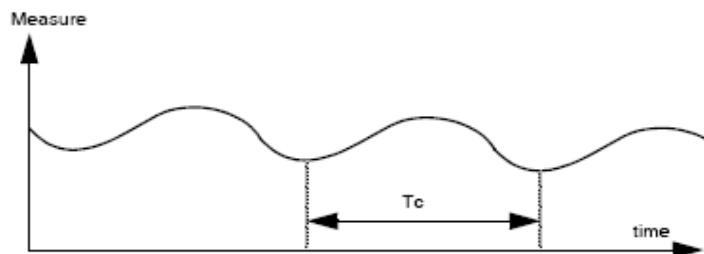
- ↗ closed loop adjustment,
- ↗ open loop adjustment.

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 (KP > 0),
- ↗ on the other hand, if this causes a PV reduction, make the PID direct (KP < 0).

14.4.19.2 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:

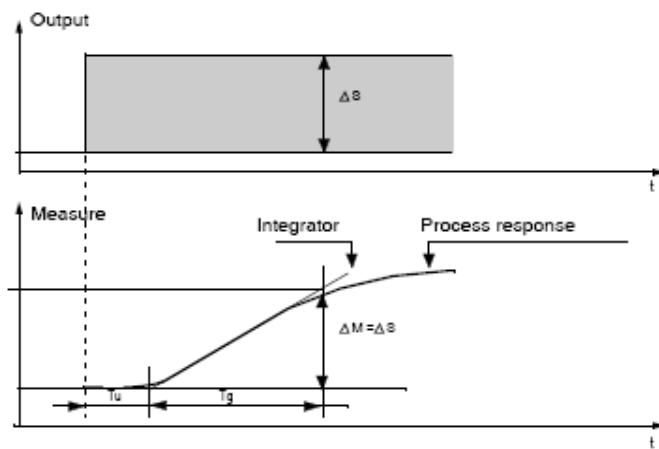
-	K_p	T_i	T_d
PID	$K_{pc}/1.7$	$T_c/2$	$T_c/8$
PI	$K_{pc}/2.22$	$0.83 \times T_c$	-

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.

14.4.19.3 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 Tg/Tu	2 x Tu	0,5 x Tu
PI	-0,9 Tg/Tu	3,3 x Tu	-

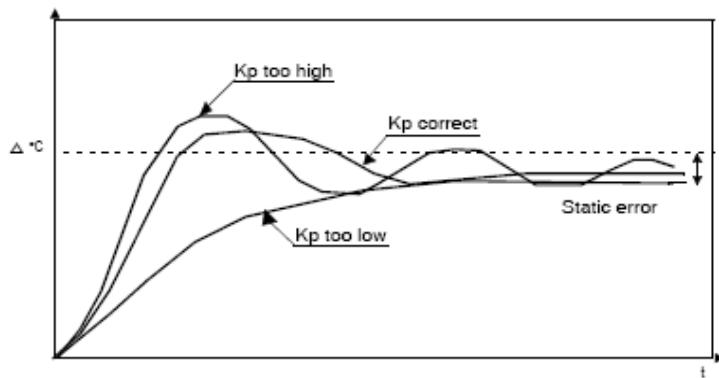
where K_P = proportional production, T_I = integration time and T_D = diversion time.

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.

14.4.20 ROLE AND INFLUENCE OF PID PARAMETERS

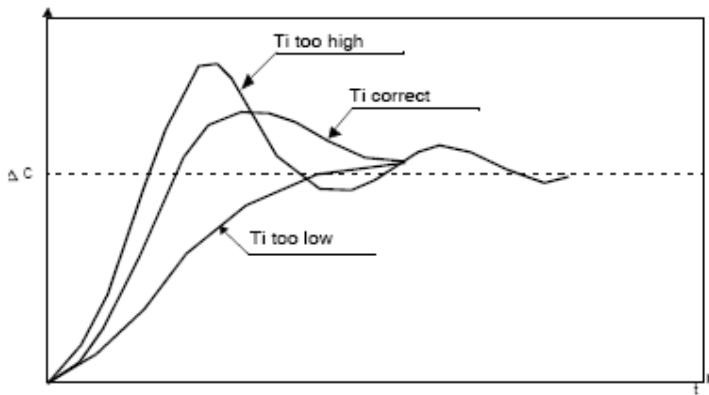
14.4.20.1 INFLUENCE OF PROPORTIONAL ACTION

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:



14.4.20.2 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:

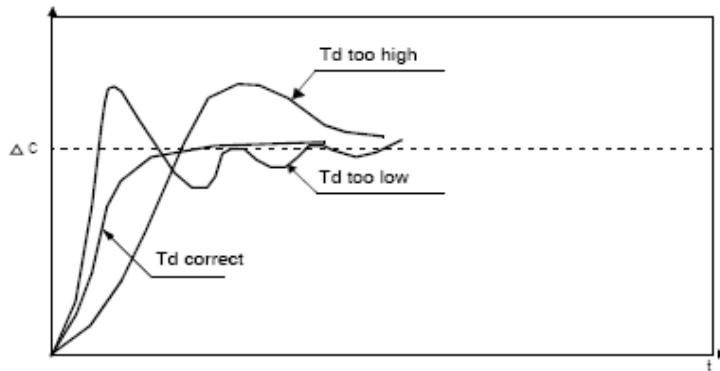


Note: A low Ti means a high level of integral action.

where K_p = proportional gain, T_i = integration time and T_d = derivative time.

14.4.20.3 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:



14.4.20.4 LIMITS OF THE PID CONTROL LOOP

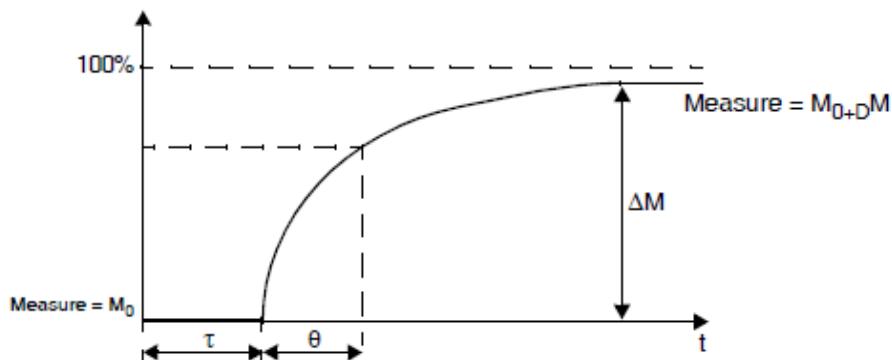
If the process is assimilated to a pure delay first order with a transfer function:

$$(H(p)) = K \frac{(e^{(-\tau)p})}{(1 + \theta p)}$$

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 τ) 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.

14.4.21 APPENDIX 1: PID THEORY FUNDAMENTALS

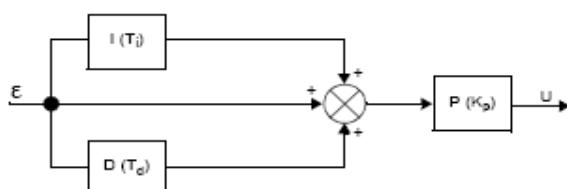
14.4.21.1 INTRODUCTION

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).

14.4.21.2 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.)

14.4.21.3 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. 496. 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. 531.)

14.4.22 APPENDIX 2: FIRST-ORDER WITH TIME DELAY MODEL

14.4.22.1 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.

14.4.22.2 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

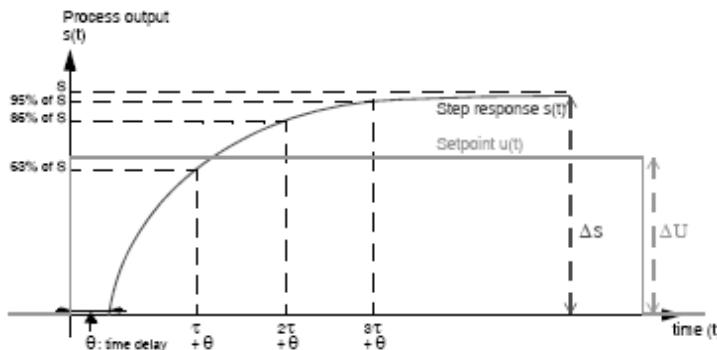
- ↗ k = the static gain,
- ↗ τ = the time constant,
- ↗ θ = the delay-time,
- ↗ U = the process input (this is the output of the PID controller),
- ↗ S = the process output.

14.4.22.3 THE PROCESS TIME CONSTANT T

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. 515.)

14.5 FLOATING POINT INSTRUCTIONS

14.5.1 AT A GLANCE

14.5.1.1 AIM OF THIS SECTION

This section describes advanced floating point (See Floating point and double word objects, p. 32) instructions in TwidoSuite language.

The Comparison and Assignment instructions are described in the Numerical Processing, p. 378

14.5.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Arithmetic instructions on floating point 344

Trigonometric Instructions 346

Conversion instructions 347

Integer Conversion Instructions <-> Floating 348

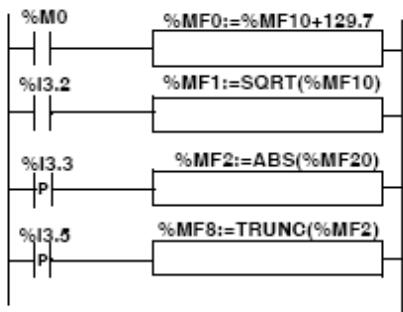
14.5.2 ARITHMETIC INSTRUCTIONS ON FLOATING POINT

14.5.2.1 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		

14.5.2.2 STRUCTURE

Ladder Language**Instruction List Language**

```

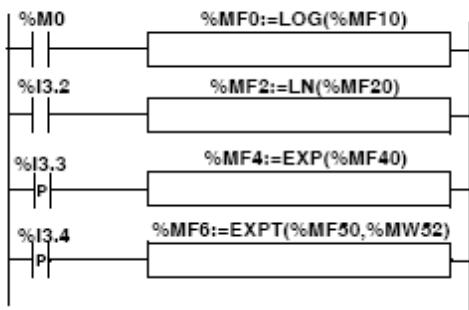
LD %M0
[%MF0:=%MF10+129.7]

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)]

```

14.5.2.3 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	$\%MFi$	$\%MFi, \%KFi$	[\cdot]
TRUNC	$\%MFi$	$\%MFi, \%KFi$	[\cdot]
EXPT	$\%MFi$	$\%MFi, \%KFi$	$\%MWi, \%KWi$, immediate value

14.5.2.4 RULES OF USE

- ▲ Operations on floating point and integer values can not be directly mixed. Conversion operations (See Integer Conversion Instructions <-> Floating, p. 541) 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. 386), the word $\%SW17$ (See System Words (%SW), p. 571) 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.

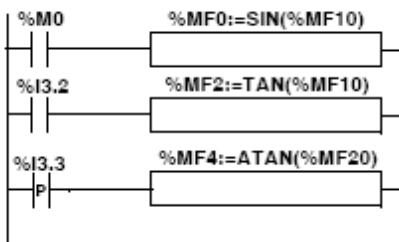
14.5.3 TRIGONOMETRIC INSTRUCTIONS

14.5.3.1 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 $\frac{\pi}{2}$)

14.5.3.2 STRUCTURE

Ladder language**Instruction List Language**

```
LD %M0
[%MFO:=SIN(%MF10)]
```

```
LD %I3.2
[%MF2:=TAN(%MF10)]
```

```
LDR %I3.3
[%MF4:=ATAN(%MF20)]
```

Structured text language

```
IF %M0 THEN
    %MFO:=SIN(%MF10);
END_IF;
IF %I3.2 THEN
    %MF2:=TAN(%MF10);
END_IF;
IF %I3.3 THEN
    %MF4:=ATAN(%MF20);
END_IF;
```

14.5.3.3 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

14.5.3.4 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. 571) 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.

14.5.4 CONVERSION INSTRUCTIONS

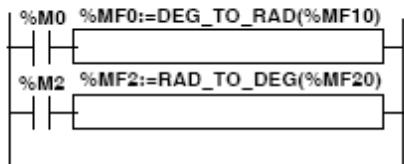
14.5.4.1 GENERAL

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

14.5.4.2 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;

```

14.5.4.3 SYNTAX

Operators, operands and syntax of conversion instructions

Operators	Syntax	Operand 1 (Op1)	Operand 2 (Op2)
DEG_TO_RAD	Op1:=Operator(Op2)	%MFi	
RAD_TO_DEG			%MFi, %KFi

14.5.4.4 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.

14.5.5 INTEGER CONVERSION INSTRUCTIONS <-> FLOATING

14.5.5.1 GENERAL

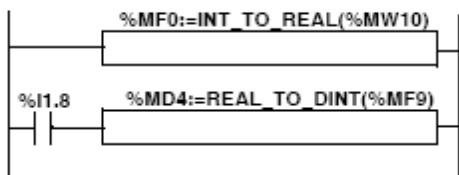
Four conversion instructions are offered.

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)

14.5.5.2 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;
    
```

14.5.5.3 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	%MFi	%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.

14.5.5.4 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

14.6 INSTRUCTIONS ON OBJECT TABLES

14.6.1 AT A GLANCE

14.6.1.1 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. 382).

14.6.1.2 WHAT'S IN THIS SECTION?

This section contains the following topics:

Topic Page

Table summing functions 351

Table comparison functions 352

Table search functions 353

Table search functions for maxi and mini values 355

Number of occurrences of a value in a table 355

Table rotate shift function 356

Table sort function 357

Floating point table interpolation function 358

Mean function of the values of a floating point table 361

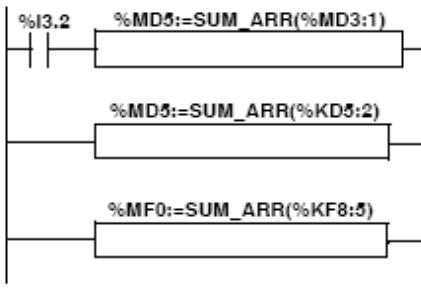
14.6.2 TABLE SUMMING FUNCTIONS

14.6.2.1 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

14.6.2.2 STRUCTURE

Ladder language**Instruction List Language**

```
LD %I3.2
[%MD5:=SUM_ARR(%MD3:1)]
[%MD5:=SUM_ARR(%KD5:2)]
%MF0:=SUM_ARR(%KF8:5)
```

14.6.2.3 SYNTAX

Syntax of table summing instruction:

<code>Res:=SUM_ARR(Tab)</code>

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.

14.6.2.4 EXAMPLE

`%MD4:=SUM(%MD30:4)`

where %MD30=10, %MD32=20, %MD34=30, %MD36=40

`%MD4:=10+20+30+40`

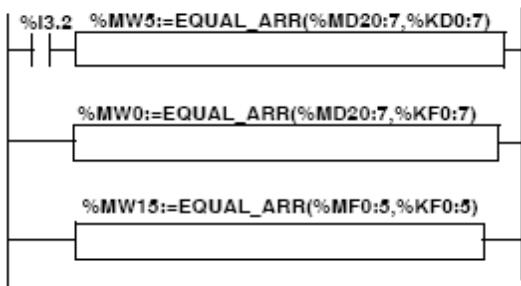
14.6.3 TABLE COMPARISON FUNCTIONS**14.6.3.1 GENERAL**

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.

14.6.3.2 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)
%MW15:=EQUAL_ARR(%MF0:5,%KF0:5)
```

14.6.3.3 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.

14.6.3.4 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	%MD32=20	%KD2=20	=
2	%MD34=30	%KD4=60	Different
3	%MD36=40	%KD6=40	=

The value of the word %MW5 is 2 (different first rank)

14.6.4 TABLE SEARCH FUNCTIONS

14.6.4.1 GENERAL

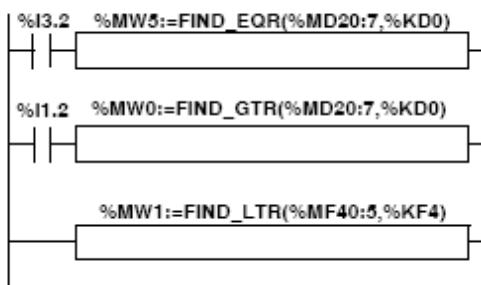
There are 3 search functions:

- ↗ 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.

14.6.4.2 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,%KF4)
    
```

14.6.4.3 SYNTAX

Syntax of table search instructions:

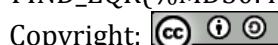
Function	Syntax
FIND_EQR	Res:=Function(Tab,Val)
FIND_GTR	
FIND_LTR	

Parameters of floating word and double word table search instructions:

Type	Result (Res)	Table (Tab)	Value (val)
Floating word tables	%MW <i>i</i>	%MF <i>i</i> :L,%KF <i>i</i> :L	%MF <i>i</i> ,%KF <i>i</i>
Double word tables	%MW <i>i</i>	%MD <i>i</i> :L,%KD <i>i</i> :L	%MD <i>i</i> ,%KD <i>i</i>

14.6.4.4 EXAMPLE

%MW5:=FIND_EQR(%MD30:4,%KD0)



Copyright: 2012 by Géza HUSI, Péter SZEMES, István BARTHA

Search for the position of the first double word =%KD0=30 in the table:

Rank	Word Table	Result
0	%MD30=10	-
1	%MD32=20	-
2	%MD34=30	Value (val), rank
3	%MD36=40	-

14.6.5 TABLE SEARCH FUNCTIONS FOR MAXI AND MINI VALUES

14.6.5.1 GENERAL

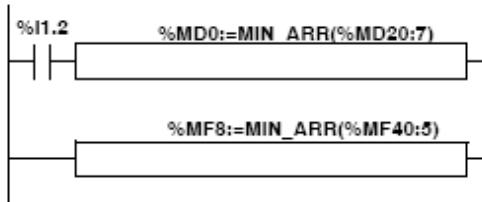
There are 2 search functions:

- ▲ 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.

14.6.5.2 STRUCTURE

Ladder language



Instruction List Language

```

LD %I1.2
[%MD0:=MIN_ARR(%MD20:7)]
[%MF8:=MIN_ARR(%MF40:5)]
    
```

14.6.5.3 SYNTAX

Syntax of table search instructions for max and min values:

Function	Syntax
MAX_ARR	Res:=Function(Table)
MIN_ARR	

Parameters of table search instructions for max and min values:

Type	Result (Res)	Table (Tab)
Double word tables	%MDi	%MDi:L,%KDj:L
Floating word tables	%MFi	%MFi:L,%KFj:L

14.6.6 NUMBER OF OCCURRENCES OF A VALUE IN A TABLE

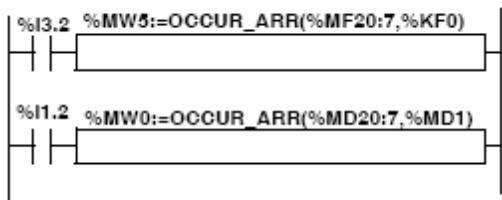
14.6.6.1 GENERAL

This search function:

- ▲ OCCUR_ARR: searches in a double word or floating word table for a number of elements equal to a given value

14.6.6.2 STRUCTURE

Ladder language



Instruction List Language

```

LD %I3.2
[%MW5:=OCCUR_ARR(%MF20:7,%KF0)]
LD %I1.2
[%MW0:=OCCUR_ARR(%MD20:7,%MD1)]
    
```

14.6.6.3 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,%KDii:L	%MDi,%KDii
Floating word tables	%MFi	%MFi:L,%KFii:L	%MFi,%KFii

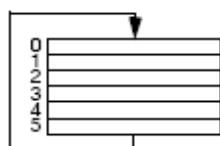
14.6.7 TABLE ROTATE SHIFT FUNCTION

14.6.7.1 GENERAL

There are 2 shift functions:

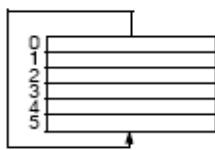
- ▲ ROL_ARR: performs a rotate shift of n positions from top to bottom of the elements in a floating word table

Illustration of the ROL_ARR functions



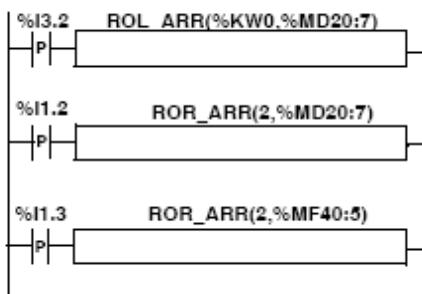
- ▲ ROR_ARR: performs a rotate shift of n positions from bottom to top of the elements in a floating word table

Illustration of the ROR_ARR functions



14.6.7.2 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)]
```

14.6.7.3 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.

14.6.8 TABLE SORT FUNCTION

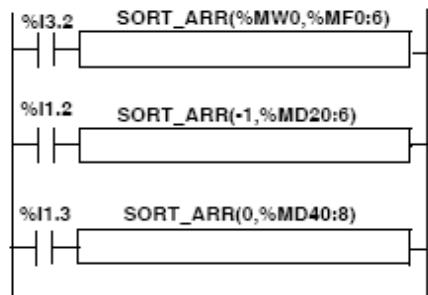
14.6.8.1 GENERAL

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.

14.6.8.2 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)]

```

14.6.8.3 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

14.6.9 FLOATING POINT TABLE INTERPOLATION FUNCTION

14.6.9.1 OVERVIEW

The LKUP function is used to interpolate a set of X versus Y floating point data for a given X value.

14.6.9.2 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 $i = 1 \dots (m - 1)$;

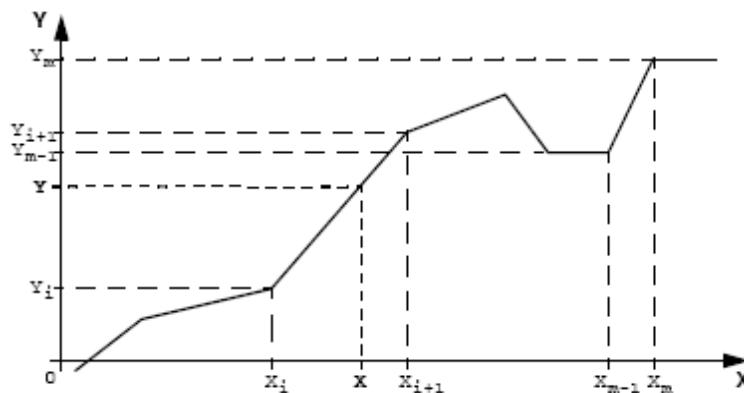
assuming X_i values are ranked in ascending order: $X_1 \leq X_2 \leq \dots X_m \dots \leq X_{m-1} \leq X_m$.

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 $i = 1 \dots (m - 1)$.

14.6.9.3 GRAPHICAL REPRESENTATION OF THE LINEAR INTERPOLATION RULE



14.6.9.4 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 (Xi, Yi) variable array
[Op1 := LKUP(Op2, Op3)]	%MWi	%MFO	Integer value, %MWi or %KWi

14.6.9.5 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).

14.6.9.6 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:

$$X_1 \leq Op2 \leq X_m$$

14.6.9.7 DEFINITION OF OP3

Op3 sets the size (Op3 / 2) of the floating-point array where the (Xi,Yi) data pairs are stored.

Xi and Yi 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 (Xi,Yi), the upper index (u) of the floating point array (%MFu) is set by using the following relationships:

- (equation 3:) $Op3 = 2 \cdot m$;
- (equation 4:) $u = 2 \cdot (Op3 - 1)$.

The floating point array Op3 (%MFi) has a structure similar to that of the following example (where Op3=8):

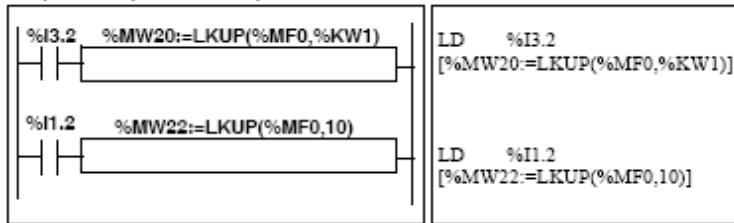
(X)		(X1)		(X2)		(X8)	
%MF0		%MF4		%MF8		%MF12	
	%MF2		%MF6		%MF10		%MF14
	(Y)		(Y1)		(Y2)		(Y8)
							(Op3=8)

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
- ▲ $Op3 \geq 6$ (for there must be at least 2 data points to allow linear interpolation).

14.6.9.8 STRUCTURE

Interpolation operations are performed as follows:



14.6.9.9 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 %MF_u, where u=18 is given by equation 4, above.

There are 4 pairs of data points stored in Op3 array [%MF4..%MF18]:

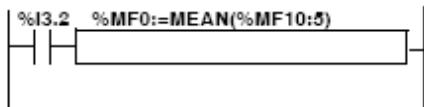
- ▲ %MF4 contains X1,%MF6 contains Y1.
- ▲ %MF8 contains X2,%MF10 contains Y2.
- ▲ %MF12 contains X3,%MF14 contains Y3.
- ▲ %MF16 contains X4,%MF18 contains Y4.

14.6.10 MEAN FUNCTION OF THE VALUES OF A FLOATING POINT TABLE

14.6.10.1 GENERAL

The MEAN function is used to calculate the mean average from a given number of values in a floating point table.

14.6.10.2 STRUCTURE

Ladder Language**Instruction List Language**

```
LD %I3.2
[%MF0:=MEAN(%MF10:5)]
```

14.6.10.3 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

15 SYSTEM BITS AND SYSTEM WORDS

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) 364

System Words (%SW) 370

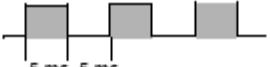
15.1.1 SYSTEM BITS (%S)

15.1.1.1 INTRODUCTION

The following section provides detailed information about the function of system bits and how they are controlled.

15.1.1.2 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: <ul style="list-style-type: none">● 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: <ul style="list-style-type: none">● 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  5 ms 5 ms	-	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: <ul style="list-style-type: none">● 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): <ul style="list-style-type: none">● At state 1, outputs are forced to 0 when the controller is in RUN mode,● At state 0, outputs are updated normally.	0	U
%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

System Bit	Function	Description	Init state	Control state
%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 system 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: <ul style="list-style-type: none">● 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: <ul style="list-style-type: none">● 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 %SWO). This bit is reset to 0 by the user.	0	S->U
%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

System Bit	Function	Description	Init state	Control
%S21	GRAFCET initialization	<p>Normally set to 0, it is set to 1 by:</p> <ul style="list-style-type: none"> • 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. <p>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.</p>	0	U->S
%S22	GRAFCET reset	<p>Normally set to 0, it can only be set to 1 by the program in pre-processing.</p> <p>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.</p>	0	U->S
%S23	Preset and freeze GRAFCET	<p>Normally set to 0, it can only be set to 1 by the program in the pre-processing program module.</p> <p>Set to 1, it validates the pre-positioning of GRAFCET.</p> <p>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.</p>	0	U->S
%S24	Operations Display	<p>Normally at 0, this bit can be set to 1 by the user.</p> <ul style="list-style-type: none"> • 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
%S25	Choosing a display mode on the operator display	<p>You can choose between two display modes on the 2-line operator display: data mode and normal mode.</p> <ul style="list-style-type: none"> • 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. <p>When %S25=1, the operator keyboard is disabled. Note: Firmware version must be V3.0 or higher.</p>	0	U

System Bit	Function	Description	Init state	Control
%S26	Choosing a signed or unsigned value on the operator display	You can choose between two value types: signed or unsigned. <ul style="list-style-type: none">• 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. <p>%S26 can only be used if %S25=1. Note: Firmware version must be V3.0 or higher.</p>	0	U
%S31	Event mask	Normally at 1. <ul style="list-style-type: none">• Set to 0, events cannot be executed and are queued.• Set to 1, events can be executed. <p>This bit can be set to its initial state 1 by the user and the system (on cold re-start).</p>	1	U->S
%S38	Permission for events to be placed in the events queue	Normally at 1. <ul style="list-style-type: none">• 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, <p>This bit can be set to its initial state 1 by the user and the system (on cold re-start).</p>	1	U->S
%S39	Saturation of the events queue	Normally at 0. <ul style="list-style-type: none">• Set to 0, all events are reported,• Set to 1, at least one event is lost. <p>This bit can be set to 0 by the user and the system (on cold re-start).</p>	0	U->S
%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. <ul style="list-style-type: none">• Set to 0, the date and time can be read,• Set to 1, the date and time can be updated. <p>The controllers internal RTC is updated on a falling edge of %S50.</p>	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. <ul style="list-style-type: none">• Set to 0, the date and time are consistent,• Set to 1, the date and time must be initialized by the user. <p>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.</p>	0	U->S

System Bit	Function	Description	Init state	Control
%S26	Choosing a signed or unsigned value on the operator display	You can choose between two value types: signed or unsigned. <ul style="list-style-type: none">• 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. <p>%S26 can only be used if %S25=1. Note: Firmware version must be V3.0 or higher.</p>	0	U
%S31	Event mask	Normally at 1. <ul style="list-style-type: none">• Set to 0, events cannot be executed and are queued.• Set to 1, events can be executed. <p>This bit can be set to its initial state 1 by the user and the system (on cold re-start).</p>	1	U->S
%S38	Permission for events to be placed in the events queue	Normally at 1. <ul style="list-style-type: none">• 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, <p>This bit can be set to its initial state 1 by the user and the system (on cold re-start).</p>	1	U->S
%S39	Saturation of the events queue	Normally at 0. <ul style="list-style-type: none">• Set to 0, all events are reported,• Set to 1, at least one event is lost. <p>This bit can be set to 0 by the user and the system (on cold re-start).</p>	0	U->S
%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. <ul style="list-style-type: none">• Set to 0, the date and time can be read,• Set to 1, the date and time can be updated. <p>The controllers internal RTC is updated on a falling edge of %S50.</p>	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. <ul style="list-style-type: none">• Set to 0, the date and time are consistent,• Set to 1, the date and time must be initialized by the user. <p>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.</p>	0	U->S

System Bit	Function	Description	Init state	Control
%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. <ul style="list-style-type: none">● 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. <ul style="list-style-type: none">● 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: <ul style="list-style-type: none">● 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
%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: <ul style="list-style-type: none">● 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. <ul style="list-style-type: none">● 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. <ul style="list-style-type: none">● Set to 0, save %MW is not OK.● Set to 1, save %MW is OK.	0	S
%S100	TwidoSuite communications cable connection	Shows whether the TwidoSuite communication cable is connected. <ul style="list-style-type: none">● Set to 1, TwidoSuite communications cable is either not attached or TwidoSuite is connected.● Set to 0, TwidoSuite Remote Link cable is connected.	-	S

System Bit	Function	Description	Init state	Control
%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. <ul style="list-style-type: none">• 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. Note: in online mode, the address of port 2 cannot be changed using system bit %S101 and system word %SW102.	0	U
%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. <ul style="list-style-type: none">• Set to 0, the protocol used is the one configured in TwidoSuite,• 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. <ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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 base controller. %S118 determines the nature of the fault. Resets to 1 when the fault disappears.	1	S

15.1.1.3 TABLE ABBREVIATIONS DESCRIBED

Abbreviation table:

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

15.1.2 SYSTEM WORDS (%SW)

15.1.2.1 INTRODUCTION

The following section provides detailed information about the function of the system words and how they are controlled.

15.1.2.2 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 loosing 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: <ul style="list-style-type: none">• 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	<ul style="list-style-type: none"> • Bit [0]: Backup/restore in progress: <ul style="list-style-type: none"> • Set to 1 if backup/restore in progress, • Set to 0 if backup/restore complete or disabled. • Bit [1]: Controller's configuration OK: <ul style="list-style-type: none"> • Set to 1 if configuration ok. • Bit [3..2] EEPROM status bits: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • Set to 1 if RAM application different to EEPROM. • Bit [5]: RAM application different to cartridge: <ul style="list-style-type: none"> • Set to 1 if RAM application different to cartridge. • Bit [6] not used (status 0) • Bit [7]: Controller reserved: <ul style="list-style-type: none"> • Set to 1 if reserved. • Bit [8]: Application in Write mode: <ul style="list-style-type: none"> • Set to 1 if application is protected. • Bit [9] not used (status 0) • Bit [10]: Second serial port installed: <ul style="list-style-type: none"> • Set to 1 if installed. • Bit [11]: Second serial port type: (0 = EIA RS-232, 1 = EIA RS-485): <ul style="list-style-type: none"> • Set to 0 = EIA RS-232 • Set to 1 = EIA RS-485 • Bit [12]: application valid in internal memory: <ul style="list-style-type: none"> • Set to 1 if application valid. • Bit [13] Valid application in cartridge: <ul style="list-style-type: none"> • Set to 1 if application valid. • Bit [14] Valid application in RAM: <ul style="list-style-type: none"> • Set to 1 if application valid. • Bit [15]: ready for execution: <ul style="list-style-type: none"> • 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	<p>For example, if %SW14=0232:</p> <ul style="list-style-type: none"> • 8 MSB=02 in hexadecimal, then xx=2 in decimal • 8 LSB=32 in hexadecimal, then yy=50 in decimal <p>As a result, Commercial version is V2.50. Note: Firmware version must be 2.5 or higher.</p>	S

System Words	Function	Description	Control
%SW15	Firmware patch, Pzz	For example, if %SW15=0005: <ul style="list-style-type: none"> • 8 MSB is not used • 8 LSB=05 in hexadecimal, then zz=5 in decimal <p>As a result, Firmware patch is P05. Note: Firmware version must be 2.5 or higher.</p>	S
%SW16	Firmware version, Vxx.yy	For example, if %SW16=0232: <ul style="list-style-type: none"> • 8 MSB=02 in hexadecimal, then xx=2 in decimal • 8 LSB=32 in hexadecimal, then yy=50 in decimal <p>As a result, Firmware version is V2.50. Note: Firmware version must be 2.5 or higher.</p>	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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • %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>Programming and diagnostics for the CANopen fieldbus</i> , p. 234.	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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • $T_{ON} \geq 1 \text{ ms}$ • 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		
		bit 0	bit 8		
		bit 1	bit 9		
		bit 2	bit 10		
		bit 3	bit 11		
		bit 4	bit 12		
		bit 5	bit 13		
		bit 6	bit 14		
		bit 7	bit 15		
%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	<p>EXCH3 error code is implemented on Ethernet-capable TWDLCAE40DRF Twido controllers only</p> <p>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.)</p> <p>The following are Ethernet-specific error codes:</p> <ul style="list-style-type: none"> 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	<p>Contains the following information:</p> <ul style="list-style-type: none"> • Controller type bits [0 -11] • 8B0 = TWDLCA*A10DRF • 8B1 = TWDLCA*A16DRF • 8B2 = TWDLMDA20DUK/DTK • 8B3 = TWDLCA*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	<p>If %S25=1, then data display mode is enabled. The operator keyboard is disabled.</p> <p>%SW68 and %SW69 can be displayed on the 2-line operator display:</p> <ul style="list-style-type: none"> • %SW68 value on the first line, • %SW69 value on the second line. <p>Note: Firmware version must be V3.0 or higher.</p>	U

System Words	Function	Description	Control
%SW73 and %SW74	AS-Interface System State	<ul style="list-style-type: none"> • 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	<p>For standard analog module, %SW8x is described as follows:</p> <p>Bit [0] All analog channels in normal state Bit [1] Module in initialization state Bit [2] Power supply default Bit [3] Configuration default Bit [4] Conversion in running for input channel 0 Bit [5] Conversion in running for input channel 1 Bit [6] Invalid parameter for input channel 0 Bit [7] Invalid parameter for input channel 1 Bit [8 & 9] Not used Bit [10] Overflow value for input channel 0 Bit [11] Overflow value for input channel 1 Bit [12] Underflow value for input channel 0 Bit [13] Underflow value for input channel 1 Bit [14] Not used Bit [15] Invalid parameter for output channel</p>	S
%SW80 cont'd	Base I/O Status cont'd	<p>For TWDAMI4LT analog module, %SW8x is described as follows:</p> <p>Bit [0 & 1] Channel 0 state 0 0: Analog channel in normal state 0 1: Invalid parameter for input channel 1 0: Unavailable input value (module in initialization state, conversion in running). 1 1: Invalid value for input channel (overflow or underflow value) Bit [2 & 3] Channel 1 state (same description as bit [0 & 1]) Bit [4 & 5] Channel 2 state (same description as bit [0 & 1]) Bit [6 & 7] Channel 3 state (same description as bit [0 & 1]) Bit [8 to 15] Not used</p>	S

System Words	Function	Description	Control
%SW80 cont'd	Base I/O Status cont'd	For TWDAMI8HT analog module, %SW8x is described as follows: Bit [0 & 1] Channel 0 state 0 0: Analog channel in normal state 0 1: Invalid parameter for input channel 1 0: Unavailable input value (module in initialization state, conversion in running). 1 1: Invalid value for input channel (overflow or underflow value) Bit [2 & 3] Channel 1 state (same description as bit [0 & 1]) Bit [4 & 5] Channel 2 state (same description as bit [0 & 1]) Bit [6 & 7] Channel 3 state (same description as bit [0 & 1]) Bit [8 & 9] Channel 4 state (same description as bit [0 & 1]) Bit [10 & 11] Channel 5 state (same description as bit [0 & 1]) Bit [12 & 13] Channel 6 state (same description as bit [0 & 1]) Bit [14 & 15] Channel 7 state (same description as bit [0 & 1])	S
%SW81		<ul style="list-style-type: none"> • Expansion I/O Module 1 Status: Same definitions as %SW80 • CANopen Master Module Status at Expansion Address 1: <ul style="list-style-type: none"> • 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
%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. <i>Note:</i> Firmware version must be V2.5 or higher.	S

System Words	Function	Description	Control
%SW96	Command and/or diagnostics for save/restore function of application program and %MW.	<ul style="list-style-type: none"> Bit [0]: Indicates that the %MW memory words must be saved to EEPROM: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Set to 1 if an error appeared, Set to 0 if a new backup request is asked for. Bit [8]: Set to 1 if the controller contains a valid application in RAM. Bit [8]: Indicates that the number of %MWs specified in %SW97 is greater than the number of %MWs configured in the application: <ul style="list-style-type: none"> 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 TwidoSuite. <ul style="list-style-type: none"> Set to 1 if an error is detected, Bit [10]: Difference between internal RAM and internal EEPROM (1 = yes). <ul style="list-style-type: none"> Set to 1 if there is a difference. Bit [14]: Indicates if an EEPROM write fault has occurred: <ul style="list-style-type: none"> Set to 1 if an error is detected, 	S and U
%SW97	Command or diagnostics for save/restore function	<p>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:</p> <p>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.</p>	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. Note: in online mode, the address of port 2 cannot be changed using system bit %S101 and system word %SW102.	S													
%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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td> </tr> <tr> <td style="text-align: center;">End of the character string</td> <td style="text-align: center;">Data bit</td> <td style="text-align: center;">Stop bit</td> <td style="text-align: center;">Parity</td> <td style="text-align: center;">RTS / CTS</td> <td style="text-align: center;">Baud rate</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table> <ul style="list-style-type: none"> ● Baud rate: <ul style="list-style-type: none"> ● 0: 1200 bauds, ● 1: 2400 bauds, ● 2: 4800 bauds, ● 3: 9600 bauds, ● 4: 19200 bauds, ● 5: 38400 bauds. ● RTS/CTS: <ul style="list-style-type: none"> ● 0: disabled, ● 1: enabled. ● Parity: <ul style="list-style-type: none"> ● 00: none, ● 10: odd, ● 11: even. ● Stop bit: <ul style="list-style-type: none"> ● 0: 1 stop bit, ● 1: 2 stop bits. ● Data bits: <ul style="list-style-type: none"> ● 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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td> </tr> <tr> <td style="text-align: center;">Timeout frame in ms</td> <td style="text-align: center;">Timeout response in multiple of 100 ms</td> </tr> <tr> <td></td> <td></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															

System Words	Function	Description	Control
%SW111	Remote link status	Indication: Bit 0 corresponds to remote controller 1, bit 1 to remote controller 2, etc. Bit [0] to [6]: <ul style="list-style-type: none">• Set to 0 = remote controller 1-7 absent• Set to 1 = remote controller 1-7 present Bit [8] to bit [14]: <ul style="list-style-type: none">• Set to 0 = remote I/O detected on remote controller 1-7• Set to 1 = extension controller detected on remote controller 1-7	S
%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]: <ul style="list-style-type: none">• Set to 0 = remote controller 1-7 not configured• Set to 1 = remote controller 1-7 configured Bit [8] to bit [14]: <ul style="list-style-type: none">• 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 base 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

System Words	Function	Description	Control
%SW121 %SW122	ASCII frame size	When bit %S103 (Comm 1) or %S104 (Comm 2) is set to 1, the ASCII protocol is used. You can change the ASCII frame size of port 1 or port 2. The ASCII frame size of port 1 is %SW121, and that of port 2 is %SW122. The value is used only on EXCH instruction start. Then, if some bytes are already received, you can't stop the reception until the last byte.	U

15.1.2.3 TABLE ABBREVIATIONS DESCRIBED

Abbreviation table:

Abbreviation	Description
S	Controlled by the system
U	Controlled by the user