

# Modicon M340 with Unity Pro CANopen User manual

November 2007 eng



---

# Table of Contents



---

	<b>Safety Information</b>	<b>7</b>
	<b>About the Book</b>	<b>9</b>
<b>Part I</b>	<b>Overview of CANopen Communication</b>	<b>11</b>
	At a Glance	11
<b>Chapter 1</b>	<b>Overview of CANopen Communication</b>	<b>13</b>
	At a Glance	13
	Principles	14
	CAN At a Glance	15
	General Architecture of the CANopen Field Bus	18
	Topology	20
	Length Limitations of the CANopen Network	23
	Conformity Class	25
<b>Part II</b>	<b>CANopen Hardware Implementation</b>	<b>27</b>
	At a Glance	27
<b>Chapter 2</b>	<b>Hardware Implementation of BMX P34 Processors</b>	<b>29</b>
	At a Glance	29
	Description of Processors: BMX P34 2010/2030	30
	Installation	32
	Visual Diagnostics of CANopen Processors	33
<b>Chapter 3</b>	<b>Presentation of CANopen devices</b>	<b>37</b>
	At a Glance	37
	CANopen Devices	38
	CANopen motion command devices	39
	CANopen Input/Output devices	45
	Other Devices	49
<b>Part III</b>	<b>Software Implementation of CANopen Communication</b>	<b>57</b>
	At a Glance	57

<b>Chapter 4</b>	<b>Generalities</b>	<b>59</b>
	At a Glance	59
	Implementation Principle	60
	Implementation Method	62
	Performances	63
<b>Chapter 5</b>	<b>Configuration of Communication on the CANopen Bus</b>	<b>65</b>
	At a Glance	65
5.1	General Points	66
	Generalities	66
5.2	Bus Configuration	67
	At a Glance	67
	How to Access the CANopen Bus Configuration Screen	68
	CANopen Bus Editor	69
	How to Add a Device on the Bus	70
	How to Delete/Move/Duplicate a Bus Device	72
	View CANopen Bus in the Project Browser	74
5.3	Device Configuration	75
	At a Glance	75
	Slave Functions	76
	Configuration Using Unity	79
	Configuration Using an External Tool: Configuration Software	85
	Manual Configuration	89
5.4	Master Configuration	91
	At a Glance	91
	How to Access the CANopen Master Configuration Screen	92
	CANopen Master Configuration Screen	94
	Description of Master Configuration Screen	96
<b>Chapter 6</b>	<b>Programming</b>	<b>99</b>
	At a Glance	99
	Exchanges Using PDOs	100
	Exchanges Using SDOs	105
	Communication functions exemple	108
	Modbus request exemple	115
<b>Chapter 7</b>	<b>Debugging Communication on the CANopen Bus</b>	<b>117</b>
	At a Glance	117
	How to Access the Debug Screens of Remote Devices	118
	Debugging Screen of the CANopen Master	119
	Slave Debug Screens	121
<b>Chapter 8</b>	<b>Diagnostics</b>	<b>125</b>
	At a Glance	125
	How to perform a diagnostic	126
	Master Diagnostics	127

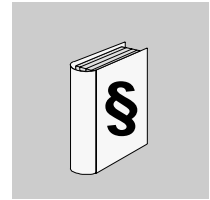
	Slave Diagnostics .....	128
<b>Chapter 9</b>	<b>Language Objects.....</b>	<b>131</b>
	At a Glance .....	131
9.1	Language objects and IODDT for CANopen communication .....	132
	At a Glance .....	132
	Introduction to the Language Objects for CANopen Communication .....	133
	Implicit Exchange Language Objects Associated with the Application-Specific Function .....	134
	Explicit Exchange Language Objects Associated with the Application-Specific Function .....	135
	Management of Exchanges and Reports with Explicit Objects .....	137
9.2	Language Objects and Generic IODDT Applicable to All Communication Protocols .....	139
	At a Glance .....	139
	Details of IODDT Implicit Exchange Objects of Type T_COM_STS_GEN ....	140
	Details of IODDT Explicit Exchange Objects of Type T_COM_STS_GEN ....	141
9.3	Language Object of the CANopen Specific IODDT .....	143
	At a Glance .....	143
	Details of T_COM_CO_BMX Type Implicit Exchange Objects of the IODDT. .	144
	Details of T_COM_CO_BMX Type Explicit Exchange Objects of the IODDT. .	156
	Language Objects Associated with Configuration. ....	158
9.4	Emergency objects .....	159
	Emergency Objects .....	159
9.5	The IODDT Type T_GEN_MOD Applicable to All Modules .....	163
	Details of the Language Objects of the IODDT of Type T_GEN_MOD. ....	163
<b>Part IV</b>	<b>Quick start : example of CANopen implementation ..</b>	<b>165</b>
	At a glance .....	165
<b>Chapter 10</b>	<b>Description of the application .....</b>	<b>167</b>
	Overview of the application .....	167
<b>Chapter 11</b>	<b>Installing the application using Unity Pro .....</b>	<b>171</b>
	At a glance .....	171
11.1	Presentation of the solution used .....	173
	At a glance .....	173
	Technological choices used .....	174
	The different steps in the process using Unity Pro .....	175
11.2	Developping the application .....	176
	At a glance .....	176
	Creating the project .....	177
	Configuration of the CANopen Bus .....	178
	Configuration of the CANopen Master .....	182
	Configuration of the equipments .....	184
	Declaration of variables .....	188

---

	Creating the program in SFC for managing the move sequence. . . . .	192
	Creating a Program in LD for Application Execution . . . . .	197
	Creating a Program in LD for the operator screen animation . . . . .	199
	Creating a program in ST for the Lexium configuration . . . . .	200
	Creating an Animation Table . . . . .	203
	Creating the Operator Screen . . . . .	205
<b>Chapter 12</b>	<b>Starting the Application . . . . .</b>	<b>209</b>
	Execution of Application in Standard Mode . . . . .	209
<b>Appendices . . . . .</b>		<b>217</b>
	At a glance. . . . .	217
<b>Appendix A</b>	<b>CANopen Master local object dictionary entry . . . . .</b>	<b>219</b>
	At a glance. . . . .	219
	Object Dictionary entries according Profile DS301 . . . . .	220
	Object Dictionary entries according Profile DS302 . . . . .	225
	Midrange Manufacturer Specific Object Dictionary Entries . . . . .	227
<b>Appendix B</b>	<b>Relation between PDOs and STB variables . . . . .</b>	<b>235</b>
	STB island configuration . . . . .	235
<b>Appendix C</b>	<b>Actions and transitions . . . . .</b>	<b>239</b>
	At a glance. . . . .	239
	Transitions . . . . .	240
	Actions . . . . .	241
<b>Glossary . . . . .</b>		<b>243</b>
<b>Index . . . . .</b>		<b>247</b>

---

## Safety Information



---

### Important Information

#### NOTICE

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



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



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

#### **DANGER**

DANGER indicates an imminently hazardous situation, which, if not avoided, **will result** in death or serious injury.

#### **WARNING**

WARNING indicates a potentially hazardous situation, which, if not avoided, **can result** in death, serious injury, or equipment damage.

#### **CAUTION**

CAUTION indicates a potentially hazardous situation, which, if not avoided, **can result** in injury or equipment damage.

**PLEASE NOTE**

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

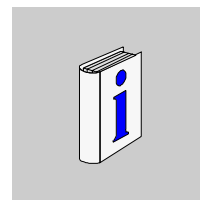
© 2007 Schneider Electric. All Rights Reserved.

---



---

## About the Book



---

### At a Glance

**Document Scope** This manual describes the implementation of a CANopen network on PLCs of the Modicon M340 range.

**Validity Note** The data and illustrations found in this documentation are not binding. We reserve the right to modify our products in line with our policy of continuous product development.

The information in this document is subject to change without notice and should not be construed as a commitment by Schneider Electric.

---

### Product Related Warnings

#### **WARNING**

##### **UNINTENDED EQUIPMENT OPERATION**

The application of this product requires expertise in the design and programming of control systems. Only persons with such expertise should be allowed to program, install, alter, and apply this product. Follow all local and national safety codes and standards.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

Schneider Electric assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product.

For reasons of safety and to ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When controllers are used for applications with technical safety requirements, please follow the relevant instructions.

Failure to observe this product related warning can result in injury or equipment damage.

---

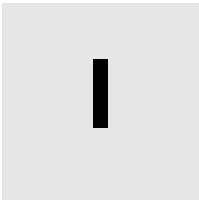
**User Comments**

We welcome your comments about this document. You can reach us by e-mail at [techpub@schneider-electric.com](mailto:techpub@schneider-electric.com)

---

---

# Overview of CANopen Communication



---

## At a Glance

<b>Aim of this Part</b>	This part introduces communication on a CANopen field bus.		
<b>What's in this Part?</b>	This part contains the following chapters:		
	<b>Chapter</b>	<b>Chapter Name</b>	<b>Page</b>
	1	Overview of CANopen Communication	13

---



---

# Overview of CANopen Communication

1

---

## At a Glance

**Aim of this Chapter**

This chapter describes the main technical characteristics for CANopen communication.

**What's in this Chapter?**

This chapter contains the following topics:

Topic	Page
Principles	14
CAN At a Glance	15
General Architecture of the CANopen Field Bus	18
Topology	20
Length Limitations of the CANopen Network	23
Conformity Class	25

---

## Principles

---

### Introduction

Originally developed for onboard automobile systems, the CAN communication bus is now used in many fields, including:

- Transport,
- Mobile devices,
- Medical equipment,
- Construction,
- Industrial control.

The strong points of the CAN system are:

- The bus allocation system,
  - Error detection,
  - Reliability of data exchanges.
- 

### Master/Slave Structure

The CAN bus has a master/slave bus management structure.

The master manages:

- The initialization of the slaves,
  - The communication errors,
  - The statuses of the slaves.
- 

### Baud Rate

The baud rate depends on the length of bus (see *Length Limitations of the CANopen Network*, p. 23) and the topology.

---

### Point to Point Communication

Communication on the bus functions point to point.

At any time, each device can send a request to the bus, to which the devices concerned respond.

The priority of the requests circulating on the bus is determined by an identifier for each message.

---

### Design Principles of the Bus

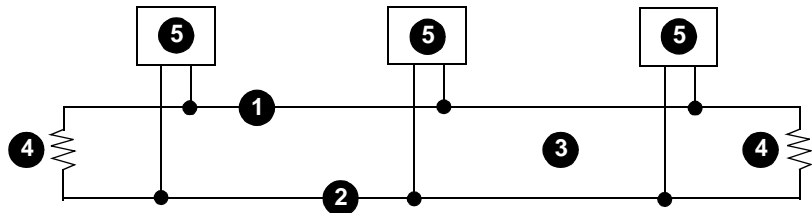
The CANOpen bus can evolve in modifying, for example the cable length, by connecting to additional devices or tap cases.

The following rules must be respected during the design of the CANOpen bus:

- determine the distance between the nodes furthest from the bus,
  - verify the length of each segment and the number of nodes connected to it,
  - verify the length and the density of taps,
  - verify that all segments have a line termination at each extremity.
-

# CAN At a Glance

<b>At a Glance</b>	CANopen is a standard Field Bus protocol for industrial monitoring systems. It is particularly adapted to Real Time PLCs, because it's an efficient, low-cost solution designed for embedded industrial applications.
<b>CANopen protocol</b>	The CANopen protocol was designed from a subset of CAL. By profile definition, it's even more specifically adapted to the use in standard industrial components. CANopen is a standard of the CiA (CAN in Automation) Association and that quickly became known as soon as it was put on the market. In Europe CANopen is now recognized as the standard reference for industrial systems based on the CAN concept.
<b>Physical layer</b>	<p>CAN uses a Bus line with two wires controlled in a differential manner (common return). A CAN signal is the difference between the tension levels of CAN-high and CAN-low. (See following figure.)</p> <p>The following figure shows the components of the physical layer of a CAN bus with two wires :</p>



Description

No.	Description
1	CAN-high wire
2	CAN-low wire
3	Difference in the potential of CAN-high/CAN-low signals
4	Resistance block of 120 Ω
5	Cell

The Bus wires can be parallel routed, twisted or reinforced according to the electromagnetic compatibility requirements. A structure with only one line reduces the reflection.

## **CANopen profiles**

### **Communication Profile**

The CANopen profile family is based on a 'communication profile' that specifies principal communication mechanisms and their description (DS301).

### **Device Profile**

The most important device types used in the industrial robotics technique are described in "Device profiles". Their functionalities are also defined there.

Examples of standard devices described are:

- the input/output digital and analog distributors (DS401),
- (DS402) Motors
- Command devices (DSP403),
- Loop controllers (DSP404)
- PLCs (DS405),
- Coding devices (DS406).

---

## **Configuration of devices via the CAN bus**

The possibility of configuring device using the CAN bus is the basic element of the independence desired by the manufacturers (by the profile family).

---

## **General characteristics of CAN open profiles**

CANopen is a group of profiles for CAN systems, which have the following specifications:

- Open bus system,
- Real time data exchange without protocol overload,
- Open bus system,
- modular conception with the possibility of modifying the size,
- Interconnection and interchangeability of devices,
- supported by numerous international manufacturers,
- Standardized configuration of networks,
- access to all the device parameters,
- Synchronization and circulation of data with cyclic process and/or commanded by events (possibility of reaction time for short systems).

---

## **Certifying CANopen products**

All manufacturers offering CANopen products certified on the market are members of the Association. As an active member of this Association (CiA), Schneider Electric Industries SAS develops its products in conformity with standards recommendations.

---



**CAN Standards**      CANopen specifications are defined by the CiA association and are partially accessible on the site [www.can-cia.com](http://www.can-cia.com). The Source code for master and slaves are available from various suppliers.

**Note:** To find out more about CANopen specifications and standard mechanisms, visit the CiA home page(<http://www.can-cia.de>).

---

**Communication on the CANopen network**      The communication profile is based on CAL services and protocols. It allows the user two types of exchange: SDO and PDO: On switch-on, the device goes into initialization phase, at the end of which it enters pre-operational state. At this stage, only communication by SDO is allowed. After receiving a start-up order, the device goes into an operational state. PDO exchanges are then started and communication by SDO is still possible.

---

**The PDO**      PDO are objects that are the communication interface with process data and allow Real Time data exchange. All PDO in a CANopen device describe implicit exchanges between this device and its communication partners on the network. PDO exchange is authorized as soon as the device is in "Operational" mode.

---

**The SDO**      SDO allow access to device data by explicit requests. The SDO service is available when the device is in an "Operational" or "Pre-operational" state.

---

## General Architecture of the CANopen Field Bus

---

### At a Glance

A CANopen architecture includes:

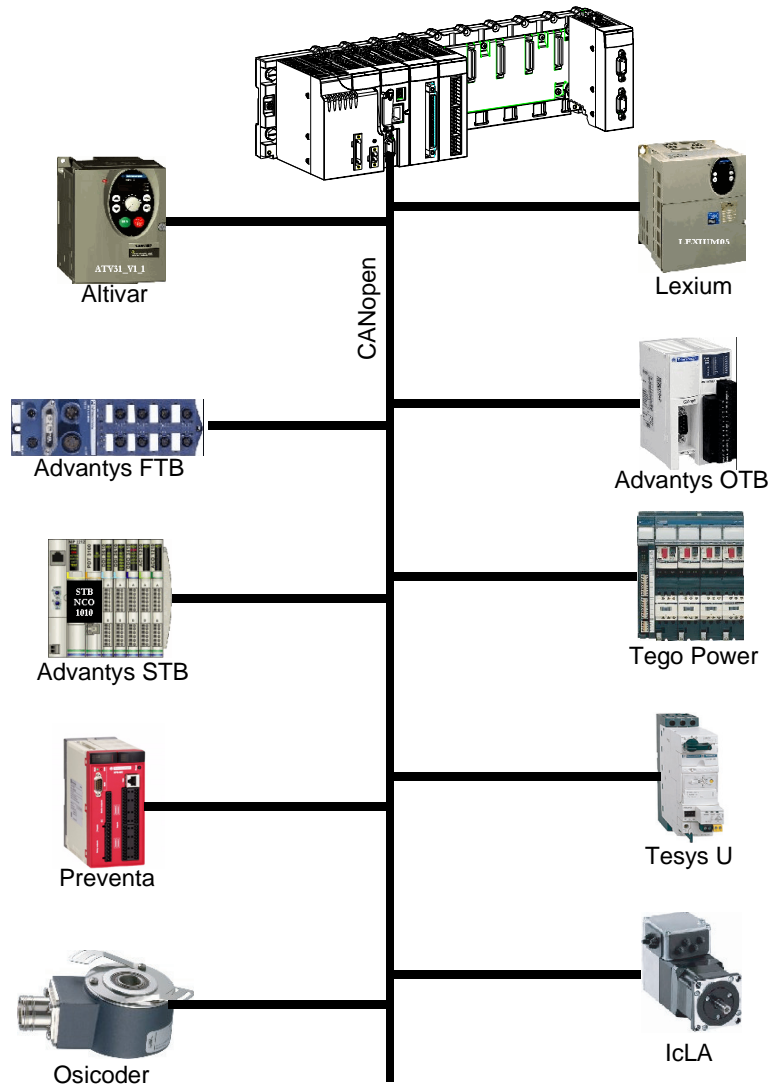
- A Bus Master
- Slave devices

**Note:** The address of the CANopen master is node number 127.

---

## Architecture Example

The following figure gives an example of CANopen architecture:



## Topology

---

### Introduction

A CANopen field bus always has a master: the BMX P34 2010/2030 processor.

The bus editor enables you to declare the network devices and to associate them to a unique address.

There are 2 types of devices:

- **compact elements:** composed of a single module.
- **modular elements:** composed of a communicator and one or several modules.

Modular devices can for example be STB islands (see *Configuration Using an External Tool: Configuration Software, p. 85*) or OTB devices.

---

## CANopen Topology

The devices can be connected to the bus.

- **Drop:** using nodes connected to a single-port or multi-port shunt box.
- **Chaining:** with single or double connectors.

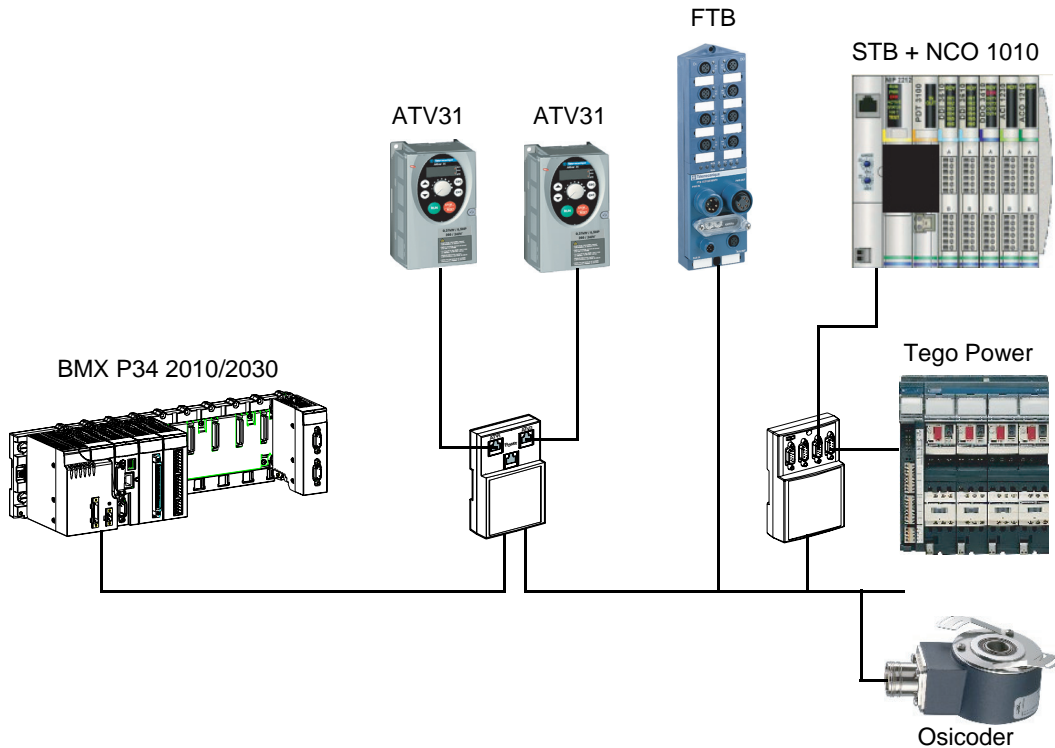
Whatever the chosen topology type, length limitations (see *Length Limitations of the CANopen Network*, p. 23) must be taken into account.

These limitations concern:

- the bus totality, that is, the maximum distance between 2 nodes,
- Segment length,
- Tap length.

All segments must have a line termination at each extremity.

The following illustration shows an example shunt topology:



**Line Terminator**

It is imperative to put a line termination in the proximity of each bus extremity in order to minimize reflections at the end of the line.

Each line termination must be connected between lines `CAN_H` and `CAN_L`.

These terminations are resistant to  $120\Omega$ , 1/4 W 5% resistors.

<b>Note:</b> In some cases the line termination is included in CANopen equipment.
---

---

**Number of  
Devices in a  
Segment**

It is theoretically possible to connect up to 63 devices on the same segment. Despite this, the topology limitations described above mean that in practice the limit is often inferior. To increase the number of devices on the bus whilst conserving the same flow rate, it is possible to switch the different segments with a "bridge".

In all cases, an M340 CANopen master cannot handle more than 63 slave devices.

---

# Length Limitations of the CANopen Network

**Introduction** The CANopen network allows you to connect up to 63 devices and a master to the bus.

Bus lengths, segments and taps are limited and detailed in the tables below.

**Bus Length** The data flow rate chosen for the bus determines the maximum length of the network in its totality:

Baud rate	Maximum length
1 Mbit/s	4 m
500 Kbit/s	100 m
250 Kbit/s	250 m
125 Kbit/s	500 m
50 Kbit/s	1000 m
20 Kbit/s	2500 m

**Note:** The bus length must also take into account the use of repeaters that add a propagation delay for information on the bus. Because repeaters add a propagation delay in the bus, this delay reduces the maximum network length of the bus. A propagation delay of 5ns is equal to a length reduction of 1m. A repeater with 150ns delay, for example, would reduce the bus length by 30m.

**Segment Lengths** Independently of the data flow rate, the number of connections and the type of cable used limit the length of a segment without a repeater.

	Resistance	Node_16	Node_32	Node_64
Large section cable AWG 18	33 Ω/km	575 m	530 m	460 m
AWG cable:22	70 Ω/km	270 m	250 m	215 m
Small section cable AWG 24	93 Ω/km 88 Ω/km	205 m 215 m	185 m 200 m	160 m 170 m
AWG cable:26	157 Ω/km	120 m	110 m	95 m

**Drop Lengths**

Length limitations concerning stubs have to be taken into account and are fixed by the following parameters:

<b>Baud rate</b>	1 Mbit/s	500 Kbit/s	250 Kbit/s	125 Kbit/s	50 Kbit/s	20 Kbit/s
<b>L max (1)</b>	0.3 m	5 m	5 m	5 m	60 m	150 m
<b><math>\Sigma L</math> max local star (2)</b>	0,6 m	10 m	10 m	10 m	120 m	300 m
<b>Minimum Interval</b> 0,6x $\Sigma L$ local (3)	-	6 m	6 m	6 m	72 m	180 m
<b><math>\Sigma L</math> max on all bus</b>	1,5 m	30 m	60 m	120 m	300 m	750 m

(1)  $L_{\max}$ : Maximum length for one stub.

(2)  $\Sigma L_{\max}$  local star: Maximum cumulative length of stubs in the same point when using a multi-port TAP creating a local star.

(3) Minimum interval: Minimum distance between two TAP. Value for a maximum length of derivation in the same point. Could be computed case by case for each derivation: minimum interval between two derivations is 60% of the cumulative length of derivations at the same point.

(4)  $\Sigma L_{\max}$  on all bus: Maximum cumulative length of stubs on all the bus.

For more details, consult the document "CANopen, Hardware Implementation Manual".



# Conformity Class

## At a Glance

The CANopen communication port conforms to the Schneider M20 class.

		Class M20
Layer configuration	Slave identification	1-63
	Binary flow (Kbit/s)	50, 125, 250, 500, 1000
Supported Devices		63
NMT	NMT Master	NMT Master conforms to DS301
	Boot Procedure	DSP 302 compliant
SDO	SDO Client	1
	SDO Server	1
	SDO Data transfer	Sent, segmented transfer
PDO	COB-ID	Read/write
	PDO TT	0, 1-240, 254, 255
	PDO Inhibit Time	TPDOs (Read/write)
	PDO Event Timer	TPDOs (Read/write)
SYNC	SYNC	Production
EMCY		Consumer/producer
HEALTH	Heartbeat	63 consumers 1 producer
	Node guarding	yes
Parameters	Save parameters.	yes

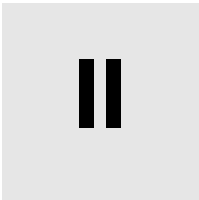
**Note:** The number of supported PDOs are as follows:

- Receiving 256 (RxPDO)
- 256 Transmitted (TxPDO)



---

# CANopen Hardware Implementation



---

## At a Glance

**Subject of this Part**

This part describes the various hardware configuration possibilities of a CANopen bus architecture.

**What's in this Part?**

This part contains the following chapters:

Chapter	Chapter Name	Page
2	Hardware Implementation of BMX P34 Processors	29
3	Presentation of CANopen devices	37



---

# Hardware Implementation of BMX P34 Processors



---

## At a Glance

### Aim of this Chapter

This chapter presents BMX P34 processors equipped with a CANopen port as well as their implementation.

### What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Description of Processors: BMX P34 2010/2030	30
Installation	32
Visual Diagnostics of CANopen Processors	33

## Description of Processors: BMX P34 2010/2030

### At a Glance

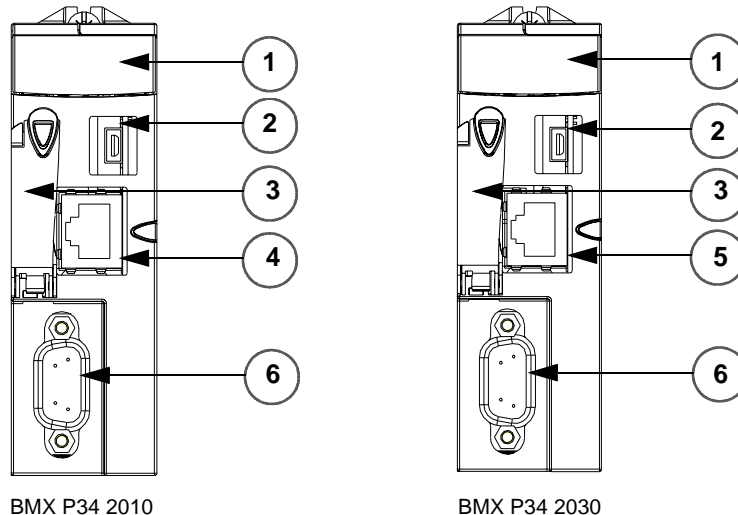
Each PLC station is equipped with a BMX P34 ••• processor.

There are two processors in the Modicon M340 range that have a CANopen port:

- The BMX P34 2010, which also has a USB port and serial port,
- The BMX P34 2030, which also has a USB port and Ethernet port.

BMX P34 ••• processors have a simple design, and include a memory card slot.

The following figures present the front sides of the BMX P34 2010/2030:



Number	Designation
1	Display panel
2	USB Port.
3	SD-Card slot
4	SerialPort
5	Ethernet Port
6	CANopen Port

These processors are bus masters; they cannot function as slaves. They are linked by SUB-D 9 connector points and allow the connection of slave devices which support the CANopen protocol.

**Note:** There is only one BMX P34 ••• master by bus.

---

## Installation

### At a Glance

BMX P34 2010/2030 processors equipped with a CANopen port are mounted on BMX XBP \*\*\*\* racks fed by BMX CPS \*\*\*\* modules.

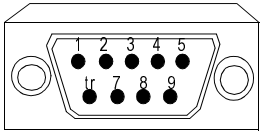
**Note:** After an extract/insert of the processor while running, the bus is no longer operational. In order to restart the bus, the power supply must be re-initialized.

### CANopen Connectors

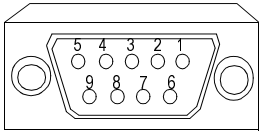
The CANopen processor port is equipped with a SUB-D9 connection.

The following figure represents the CANopen connector for modules (male) and cables (female).

Male connector



Female connector



Pin	Signal	Description
1	-	Reserved
2	CAN_L	CAN_L bus line (Low)
3	CAN_GND	CAN mass
4	-	Reserved
5	Reserved	CAN optional protection
6	GND	Optional mass
7	CAN_H	CAN_H bus line (High)
8	-	Reserved
9	Reserved	CAN External Power Supply. (Dedicated to the optocouplers power and transmitters-receivers.) Optional

**Note:** CAN\_SHLD and CAN\_V+ are not installed on the Modicon M340 range processors. These are reserved connections.



# Visual Diagnostics of CANopen Processors

## At a Glance

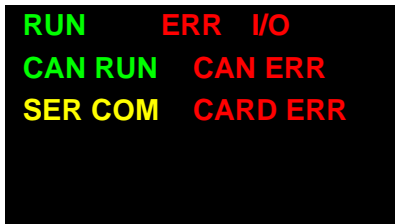
BMX P34 **••••** processors form the Modicon M340 range are equipped with several Module Status visualization LEDs.

BMX P34 2010/2030 processors equipped with a CANopen port have 2 LEDs on their facade that indicate the bus status:

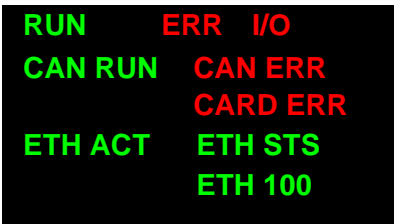
- a green CAN RUN LED,
- a red CAN ERR LED.

In normal operation, the CAN ERR LED is off and the CAN RUN LED is on.

The following figures show the LEDs on the facade of modules:



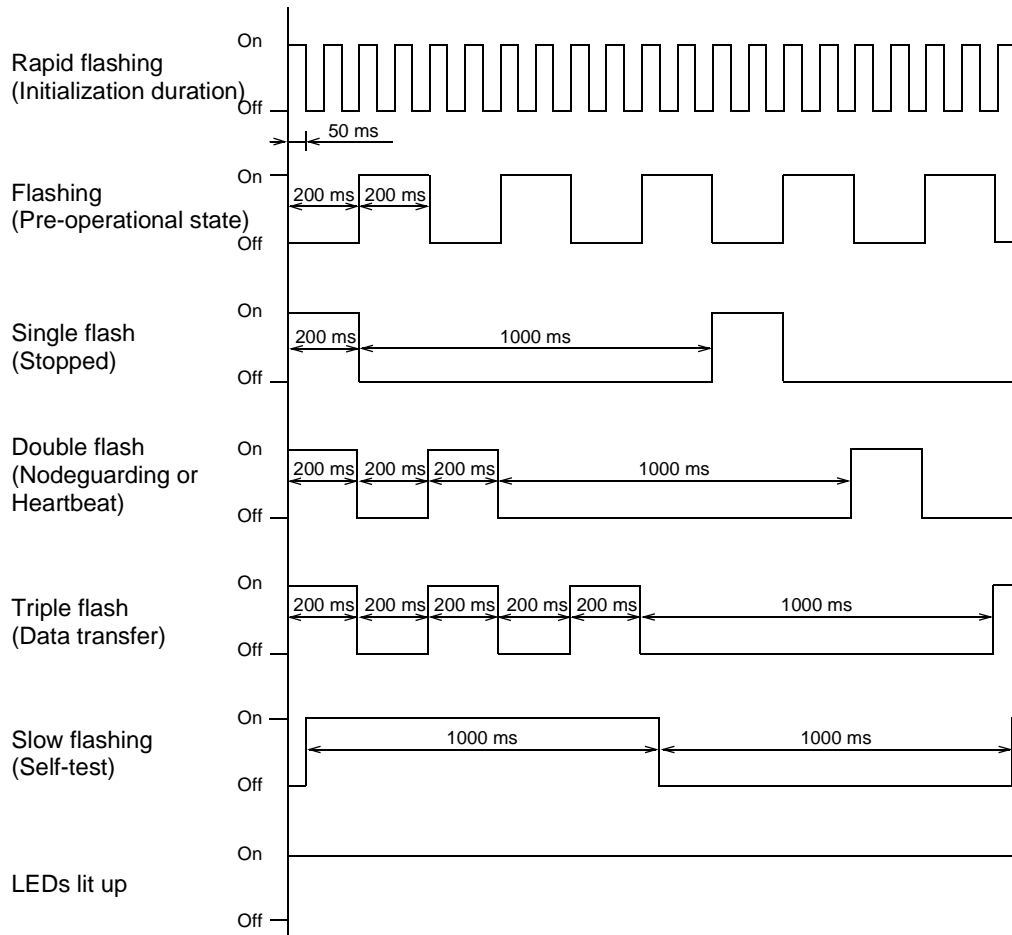
Visualization screen of BMX P34 2010



Visualization screen of BMX P34 2030







**LED Status**

The following trend diagram represents the possible status of LEDs:



**Description**

The following table describes the role of CAN RUN and CAN ERR LEDs:

Display LED	On 	Rapid flashing 	Flash 	Flashing 	Off 	Slow flashing 
CAN RUN (green)	The master is operational.	Initialization in process.	<b>Simple:</b> The master is stopped. <b>Triple :</b> Loading of CANopen firmware in process.	The master is pre-operational.	-	Starting CANopen master self-test.
CAN ERR (red)	Bus stopped. The CAN controller has status "BUS OFF".	Initialization in process.	<b>Simple :</b> at least one of the error counters has attained or over or exceeded the alert level. <b>Double :</b> Monitoring fault (Nodeguarding or Heartbeat)	Invalid configuration.	No error.	The CANopen component cannot start.



---

# Presentation of CANopen devices



---

## At a Glance

**Subject of this Section**

This section presents the different CANopen devices.

**What's in this Chapter?**

This chapter contains the following topics:

Topic	Page
CANopen Devices	38
CANopen motion command devices	39
CANopen Input/Output devices	45
Other Devices	49

## CANopen Devices

---

### At a Glance

The devices that you can connect to a CANopen bus and that can be configured in Unity Pro V3.1 are grouped according to their functions:

- motion command devices,
- input/output devices,
- other devices.

<b>Note:</b> Only devices from the hardware catalog can be used with Unity Pro.
---

---

### Motion Command Devices

Motion command devices enable you to control motors.

These devices are:

- Altivar,
- Lexium,
- IcLA,
- Osicoder,
- Telsys T,
- SD328A Stepper Drive.

---

### Input/Output Devices

Input/Output modules function as remote modules. These devices are:

- Tego Power devices,
- Advantys FTB,
- Advantys OTB,
- Advantys FTM,
- Preventa devices.

---

### Other Devices

These are:

- Advantys islands STB,
- Tesys U,
- Festo Valve Terminal,
- Parker Moduflex.

The STB islands also allow the monitoring of inputs/outputs.

---

## CANopen motion command devices

---

### At a glance

Motion command devices enable you to control motors.

These devices are:

- Altivar,
  - Lexium,
  - IcLA,
  - Osicoder,
  - Tesys T,
  - SD328A Stepper Drive.
-

### Altivar devices

An Altivar device enable to control the speed a motor by flux vector control.  
The following figure gives an example of an Altivar device:



**Note:** The recommended minimum version of the firmware is V1.1 for ATV31, ATV61 and ATV71.

**Note:** ATV31 V1.7 is not supported. However, it can be used by configuring it with ATV31 1.2 profile. In this case, only the ATV31 V1.2 functions will be available

**Note:** ATV71 : if you have to disconnect it from the CANopen bus, power off the device, else, when reconnecting it on the bus, it will provoke a Bus Fatal error. This problem is fixed with the ATV71 firmware version V1.2 and upper.

**Note:** ATV61 : if you have to disconnect it from the CANopen bus, power off the device, else, when reconnecting it on the bus, it will provoke a Bus Fatal error. This problem is fixed with the ATV61 firmware version V1.4 and upper.



**Lexium devices**

The range of Lexium 05 servo drives that are compatible with BSH servo motors constitutes a compact and dynamic combination for machines across a wide power (0,4...6 kW) and power supply voltage range.

The compact design of the Lexium 05 servo drive and the integrated components (line filter, braking resistor and safety function) reduces the space required in the switch cabinet to a minimum. It integrates the Power Removal safety function which prevents accidental starting of the motor.

Another advantage of the servodrive Lexium 05 is the versatile application options:

- as torque or speed controller via the analogue inputs,
- as electronic gearbox via the RS422 interface,
- as positioning or speed controller via the field bus interface.

The servodrive is available in four voltage types:

- 115 VAC single-phase,
- 230 VAC single-phase and 3-phase,
- 400/480 VAC 3-phase.

The following figure gives an example of a Lexium device:



**Note:** The recommended minimum version of the firmware for Lexium05 device is V1.120

**Note:** The recommended minimum version of the firmware for Lexium 15 LP is V1.45

**Note:** The recommended minimum version of the firmware for Lexium 15 MH is V6.64

### IcLA devices

IcLA devices are intelligent compact drives. They integrate everything required for motion tasks: positioning controller, power electronics and servo, EC or stepper motor.

The following figure gives an example of an IcLA device:



#### **WARNING**

##### **NON-SUPPORTED ICLA IFA VERSION.**

Operating is guaranteed from the minimum firmware version V1.105.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

#### **WARNING**

##### **NON-SUPPORTED ICLA IFE VERSION.**

Operating is guaranteed from the minimum firmware version V1.104.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

#### **WARNING**

##### **NON-SUPPORTED ICLA IFS VERSION.**

Operating is guaranteed from the minimum firmware version V1.107.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

**Osicoder  
devices**

The Osicoder device is an angular position sensor.

Mechanically coupled to a driving spindle of a machine, the shaft of the encoder rotates a disc that comprises a succession of opaque and transparent sectors. Light from leds passes through the transparent sectors of the disc as they appear and is detected by photosensitive diodes. The photosensitive diodes, in turn, generate an electrical signal which is amplified and converted into a digital signal before being transmitted to a processing system or an electronic variable speed drive. The electrical output of the encoder therefore represents, in digital form, the angular position of the input shaft.

The following figure gives an example of an Osicoder device:



<b>Note:</b> The minimum version of the firmware for Osicoder devices is V1.0.
--

---

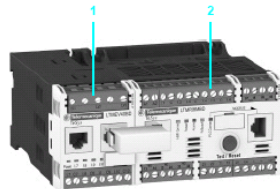
### **Tesys T Motor Management System**

Tesys T is a motor management system that provides protection, metering and monitoring functions for single-phase and 3-phase, constant speed, a.c. motors up to 810 A.

Its use in motor control panels makes it possible to:

- Increase the operational availability of installations,
- improve flexibility from project design through to implementation,
- increase productivity by making available all information needed to run the system.

The following figure gives an example of a Tesys T device:



1 LTM EV40BD extension module  
2 LTM R08MBD controller

---

### **SD328A Stepper Drive**

The SD328A is a universally applicable stepper drive.

It offers a very compact and powerful drive system in combination with selected stepper motors by Berger Lahr.

The device has an output for direct connection of an optional holding brake.

The following figure gives an example of a SD328A Stepper Drive device:



## CANopen Input/Output devices

---

### At a glance

The Input/Output modules function as remote modules.

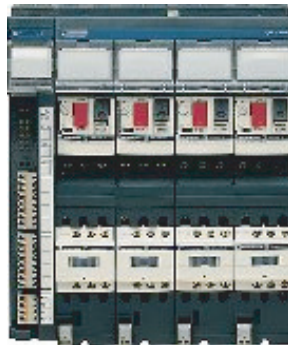
These devices are:

- Tego Power devices,
- Advantys FTB,
- Advantys OTB,
- Advantys FTM,
- Preventa devices.

### Tego Power devices

Tego Power is a modular system which standardizes and simplifies the implementation of motor starters with its pre-wired control and power circuits. In addition, this system enables the motor starter to be customized at a later date, reduces maintenance time and optimizes panel space by reducing the number of terminals and intermediate interfaces and also the amount of ducting.

The following figure gives an example of a Tego Power device:



<p><b>Note:</b> The minimum version for TegoPower APP_1CCO0 and TegoPower APP_1CCO2 is V1.0</p>
---

---

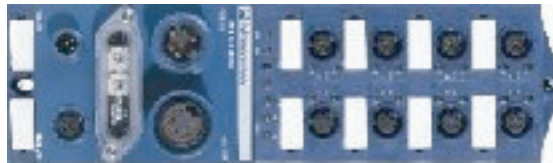
### **Advantys FTB devices**

The Advantys FTB dispatcher is composed of several input/outputs that allow captors and activators to be connected.

**Note:** The minimum firmware version for FTB is V1.7

**Note:** For FTB 1CN16CM0, operating is guaranteed from the minimum firmware version V1.5.

The following figure gives an example of an Advantys FTB device:



**Advantys OTB devices**

An Advantys OTB device enables you to constitute discrete input/output islands (max.132 channels in boundaries) or analog (max. 48 channels) IP20 and to connect them close to the active captors.

The following figure gives an example of an Advantys OTB device:



<b>Note:</b> The minimum firmware version for OTB is V2.0
---

<b>⚠ WARNING</b>
------------------

<b>NON-SUPPORTED OTB VERSION.</b>
-----------------------------------

Operating is guaranteed from the minimum firmware version V2.0.
---

<b>Failure to follow these instructions can result in death, serious injury, or equipment damage.</b>
---

### **Advantys FTM CANopen**

The Advantys FTM modular system enables you to connect a variable number of input/output splitter boxes, using a single communication interface (field bus module).

These splitter boxes are connected to the module using a hybrid cable which includes the internal bus and power supply (internal, sensor and actuator).

The input/output splitter boxes are independent of the field bus type, thus reducing the number of splitter box references. Once installed, the system is ready to begin operation.

The following figure gives an example of an Advantys FTM CANopen device:



---

### **Preventa devices**

Preventa devices are electronic safety controllers for monitoring safety functions.

The following figure gives an example of a Preventa device:





## Other Devices

---

### At a Glance

These devices are:

- STB Island,
  - Tesys U,
  - Festo Valve Terminal,
  - Parker Moduflex.
-

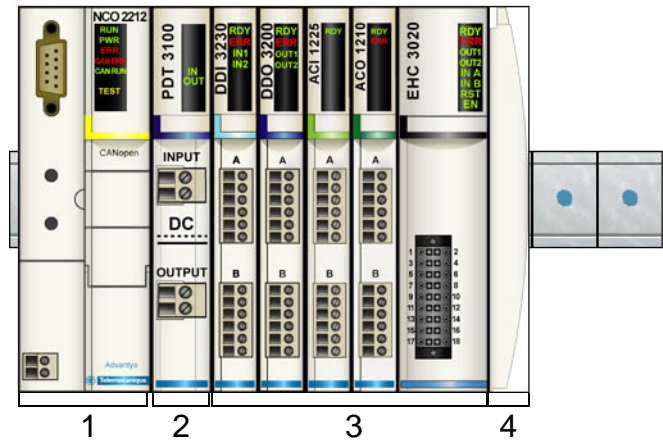
**STB Island**

An Advantys STB island is composed of several input/output modules.

The modular elements of the island are connected by a CANopen local bus using a network interface module NIM.

STB modules can only be used in an STB island.

The following figure gives an example of an island:



Description:

Number	Designation
1	Network Interface Module.
2	Power supply Distribution Module.
3	Distributed input/output modules. These modules can be: <ul style="list-style-type: none"><li>● digital input/output modules,</li><li>● analog input/output modules,</li><li>● special purposes.</li></ul>
4	Termination plate of island bus.

**Tesys U Devices**

TeSys U-Line motor starters provide motor control for choices ranging from a basic motor starter with solid-state thermal overload protection to a sophisticated motor controller which communicates on networks and includes programmable motor protection.

This device performs the following functions:

- Protection and control of 1-phase or 3-phase motors:
  - isolation breaking function,
  - electronic short-circuit protection,
  - electronic overload protection,
  - power switching.
- Control of the application:
  - alarming (warning protection function alarms, e.g. overload pending),
  - status monitoring (running, ready, fault...),
  - application monitoring (running time, number of faults, motor current values),
  - fault logging (last 5 faults saved, together with motor parameter values).

The following figure gives an example of a Tesys U device:



**Festo valve terminal****CPV Direct:**

CPV valves are series manifold valves, in addition to the valve function they contain all of the pneumatic ducts for supply, exhaust and the working lines.

The supply ducts are a central component of the valve slices and allow a direct flow of air through the valve slices. This helps achieve maximum flow rates. All valves have a pneumatic pilot control for optimising performance.

The fieldbus node is directly integrated in the electrical interface of the valve terminal and therefore takes up only a minimal amount of space.

The optional string extension allows an additional valve terminal and I/O modules to be connected to the Fieldbus Direct fieldbus node.

The CPV valve terminal is available in three sizes:

- CPV10
- CPV14
- CPV18

The following figure gives an example of a Festo valve terminal device:

**CPX Terminal:**

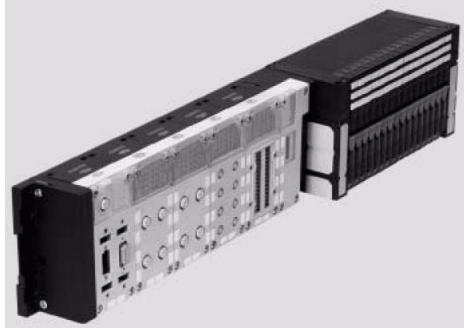
The electrical terminal CPX is a modular peripheral system for valve terminals. The system is specifically designed so that the valve terminal can be adapted to suit different applications.

Variable connection options for the valve terminal pneumatic components (MPA/CPA/VTSA)

Flexible electrical connection technology for sensors and actuators

The CPX terminal can also be used without valves as a remote I/O system.

The following figure gives an example of a CPX terminal device:



**Parker Moduflex**

Parker Moduflex Valve System provides flexible pneumatic automation.

Depending on application, you can assemble short or long islands (up to 16 outputs). IP 65-67 water and dust protection allows the valve to be installed near the cylinders for shorter response time and lower air consumption. The Parker Moduflex Valve System CANopen module (P2M2HBVC11600) can be used as an enhanced CANopen device in an Modicon M340 configuration.

The firmware version of the P2M2HBVC11600 must be V 1.4 or later.

For detailed descriptions of P2M2HBVC11600 wiring, LED patterns, set-up procedures, and functionality, refer to user documentation provided by Parker.

**"S" Series Stand-Alone Valves:**

For isolated cylinders on a machine, it is preferable to locate the valve close by. Therefore a stand-alone module is ideal, response time and air consumption are then reduced to a minimum. Peripheral modules can be installed directly into the valve.

The following figure gives an example of a "S" Series Single Solenoid device:



The following figure gives an example of a "S" Series Single Air Pilot device:

**"T" Series Valve Island Modules**

For small groups of cylinders requiring short localized valve islands.

Modules with different functions and flow passages may be combined in the same island manifold, giving total flexibility to adapt to all machine requirements.

The following figure gives an example of a "T" Series Valve Island Module device:







---

# Software Implementation of CANopen Communication



---

## At a Glance

### Subject of this Part

This part describes the various possibilities for software configuration, programming and diagnostics in a CANopen application.

### What's in this Part?

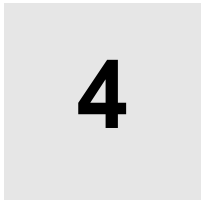
This part contains the following chapters:

Chapter	Chapter Name	Page
4	Generalities	59
5	Configuration of Communication on the CANopen Bus	65
6	Programming	99
7	Debugging Communication on the CANopen Bus	117
8	Diagnostics	125
9	Language Objects	131



---

# Generalities



---

## At a Glance

### Subject of this Chapter

This chapter describes CANopen software implementation principles on the Modicon M340 bus.

### What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Implementation Principle	60
Implementation Method	62
Performances	63

## Implementation Principle

---

### At a Glance

In order to implement a CANopen bus, it is necessary to define the physical context of the application in which the bus is integrated (rack, supply, processor, modules), and then ensure that the necessary software is implemented.

The software is implemented in two ways with Unity Pro:

- in offline mode
  - in online mode
-

## Implementation Principle

The following table shows the different implementation phases:

Mode	Phase	Description
Offline	Configuration	Entry of configuration parameters.
Offline or online	Symbolization	Symbolization of the variables associated with the CANopen port of the <b>BMX P34</b> processor.
	Programming	Programming the specific functions: <ul style="list-style-type: none"> <li>• bit objects or associated words,</li> <li>• Specific instructions.</li> </ul>
Online	Transfer	Transferring the application to the PLC.
	Debugging Diagnostics	Different resources are available for debugging the application, controlling inputs/outputs and diagnosing faults: <ul style="list-style-type: none"> <li>• Language objects or IODDTs,</li> <li>• The Unity Pro debugging screen,</li> <li>• Signaling by LED.</li> </ul>
Offline or online	Documentation	Printing the various information relating to the configuration of the CANopen port.

**Note:** The above order is given for your information. Unity Pro software enables you to use editors in the desired order of interactive manner.

## DANGER

### UNEXPECTED APPLICATION BEHAVIOR

Use diagnosis system information and monitor the response time of the communication. In case of disturbed communication, the response time can be too high.

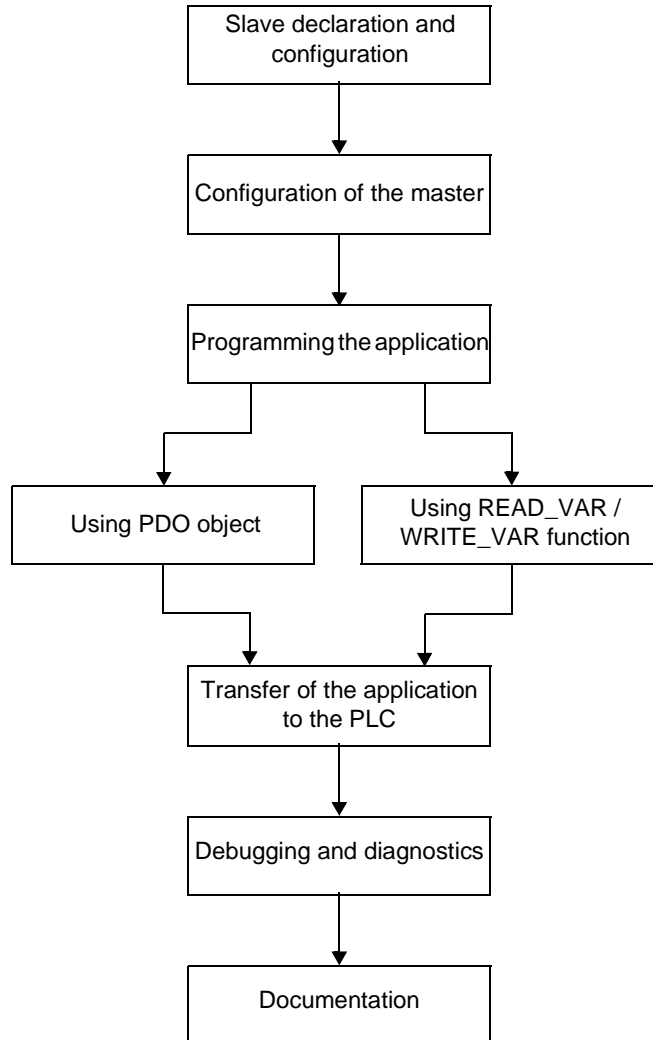
**Failure to follow these instructions will result in death or serious injury.**

## Implementation Method

---

### Overview

The following flowchart shows the CANopen port implementation method for BMX P34 •••• processors:



---

## Performances

---

### Introduction

Various CANopen performances are detailed below.

---

### Impact on Task Cycle Time

The time given to each task cycle is as follows:

Task	Minimum
CANopen inputs	10 $\mu$ s / PDO
CANopen outputs	120 $\mu$ s / PDO
Diagnostics	100 $\mu$ s

---

### Communication by SDO

The average duration of READ\_VAR and WRITE\_VAR functions is as follows:

Function	Minimum
READ_VAR or WRITE_VAR	2 * Task cycle in ms * Number of SDOs

---

Example : for a task cycle of 15ms and a number of 10 SDOs, the SDO exchange time is :  $2 * 15 * 10 = 300$  ms.

**Note:** Only one SDO is exchanged at the same time on the bus. It is necessary to await the end of the preceding exchange to begin a new exchange. The end of exchange polling is carried out at each task cycle, so there is one SDO exchange for each task cycle.

---

### Bus Start

The CANopen bus start time depends on the number of devices.

The minimum time to start a CANopen bus is 7 seconds.

The time to configure one device is about 0.8 second.

The start time of a CANopen bus with 64 devices is about 1 minute.

---

**Disconnection/  
Reconnection of  
a Device****Disconnection:**

The time to detect the disconnection of a device depends on the error control:

Error control	Description
Guardtime	The time to detect the disconnection is <b>Guardtime * life time factor</b>
Heartbeat	The time to detect the disconnection is <b>Heartbeat producer time + (Heartbeat producer time /2)</b>

**Reconnection:**

Each second, the master polls on the device to check the reconnection of the device. The time to reconnect the device is about 1 second if the device is not alone on the bus.

If the device is alone on the bus, the disconnection of the device set the master in the same case as the disconnection of the complete bus. After this state, the master restarts the bus and the reconnection time of the device is about 7 seconds.

---



---

# Configuration of Communication on the CANopen Bus



---

## At a Glance

### Aim of this Chapter

This chapter presents the configuration of the CANopen field bus and of the bus master and slaves.

### What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
5.1	General Points	66
5.2	Bus Configuration	67
5.3	Device Configuration	75
5.4	Master Configuration	91

## 5.1 General Points

---

### Generalities

---

#### Introduction

Configuration of a CANopen architecture is entirely integrated to Unity Pro.

When the channel of the CANopen master has been configured, a node is automatically created in the project browser. It is then possible to launch Bus Editor from this node in order to define the topology of the bus and configure the CANopen elements.

<p><b>Note:</b> You cannot modify the configuration of the CANopen bus in connected mode.</p>
---

---

# 5.2                    Bus Configuration

## At a Glance

**Subject of this Section**                    This section presents the configuration of the CANopen bus.

**What's in this Section?**                    This section contains the following topics:

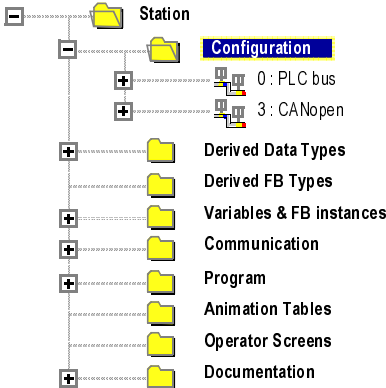
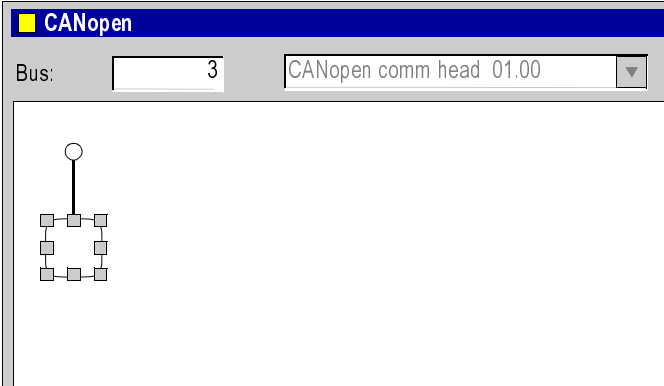
Topic	Page
How to Access the CANopen Bus Configuration Screen	68
CANopen Bus Editor	69
How to Add a Device on the Bus	70
How to Delete/Move/Duplicate a Bus Device	72
View CANopen Bus in the Project Browser	74

## How to Access the CANopen Bus Configuration Screen

This describes how to access the configuration screen of the CANopen bus for a Modicon M340 PLC with a built-in CANopen link.

**Procedure**

To access the CANopen field bus, perform the following actions:

Step	Action
1	<p>From the project navigator, deploy the <b>Configuration</b> directory.</p> <p><b>Result:</b> the following screen appears:</p> 
2	<p>To open the CANopen bus screen, select one of the following methods:</p> <ul style="list-style-type: none"><li>• double-click on the CANopen directory,</li><li>• select the CANopen sub-directory and select <b>Open</b> in the contextual menu.</li></ul> <p><b>Result:</b> the <b>CANopen</b> window appears:</p> 

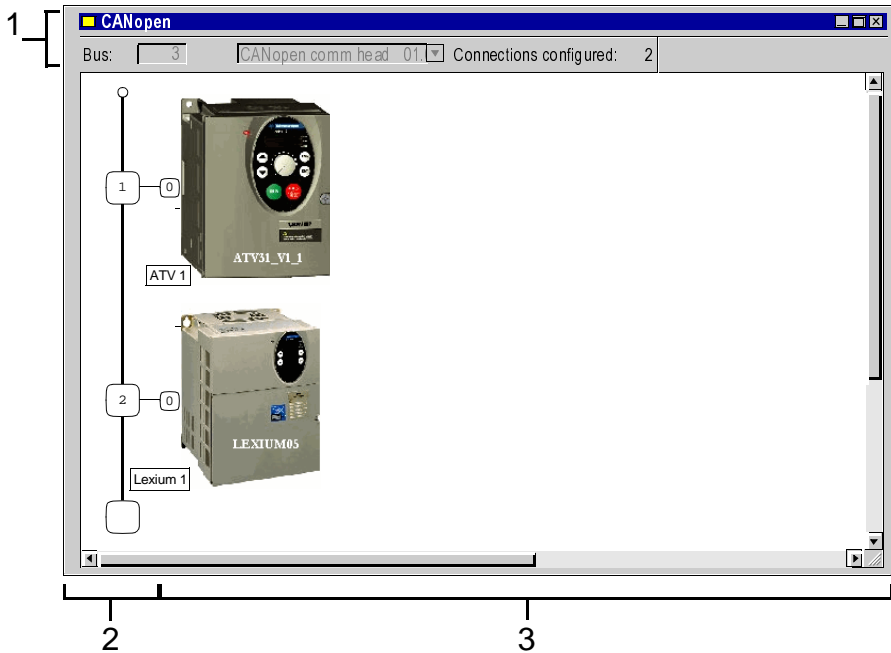
## CANopen Bus Editor

### At a Glance

This screen is used to declare devices which are connected to the bus.

### Illustration

The CANopen bus editor looks like this:



### Elements and Functions

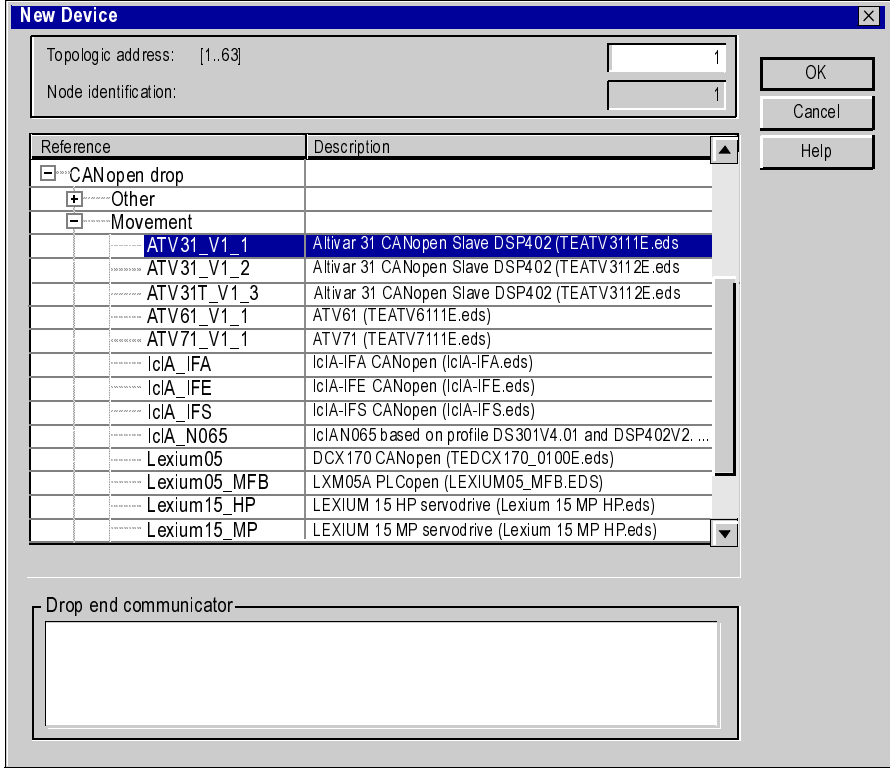
This table describes the different areas that make up the configuration screen:

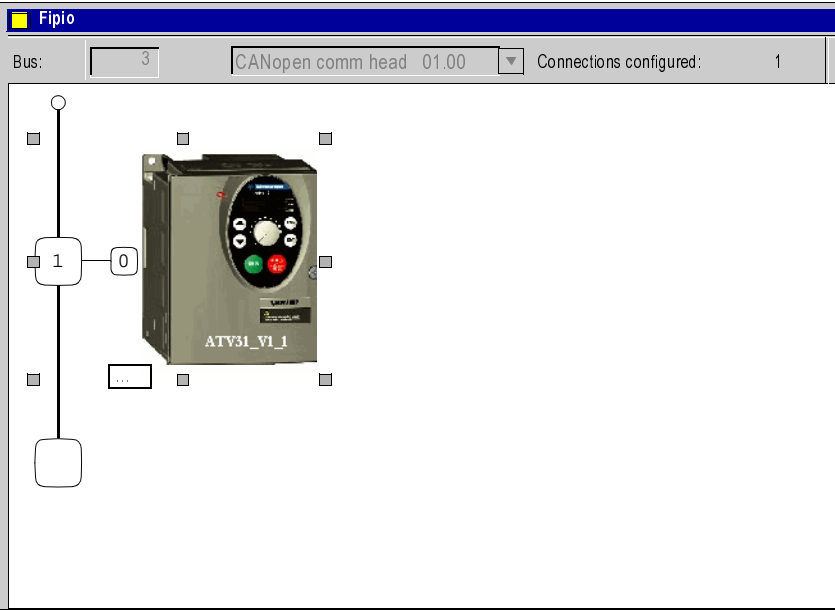
Number	Element	Function
1	Bus	Bus number.
	Connections configured	Indicates the number of connection points configured.
2	Logical address area	This area includes the addresses of the devices connected to the bus.
3	Module area	This area includes the devices that are connected to the bus.

Available connection points are indicated by an empty white square.

## How to Add a Device on the Bus

**Procedure** This operation is used to add, via the software, a device connected to the CANopen bus:

Step	Action
1	Access the CANopen (see <i>How to Access the CANopen Bus Configuration Screen, p. 68</i> ) configuration screen.
2	<p>Double-click on the place where the module should be connected.  <b>Result:</b> the <b>New Device</b> screen appears.</p> 
3	Enter the number of the connection point corresponding to the address. By default, the Unity Pro software offers the first free consecutive address.
4	In the <b>Communicator</b> field, select the element type enabling communication on the CANopen bus. For modules with built-in communicators, this window does not appear.

Step	Action
5	<p>Validate with <b>Ok</b>. <b>Result:</b> the module is declared.</p>  <p>The screenshot shows a software window titled "Fipio" with a blue header bar. Below the header, there is a status bar with the text "Bus: 3", "CANopen comm head 01.00", and "Connections configured: 1". The main area of the window displays a network diagram. On the left, a vertical line represents the CANopen bus. A square node labeled "1" is connected to this bus. To the right of node "1" is a square node labeled "0". In the center of the diagram is a detailed image of an industrial motor drive unit, labeled "ATV31_V1_1". The unit has a circular control panel with several buttons and a small display. The entire diagram is enclosed in a light gray border.</p>

## How to Delete/Move/Duplicate a Bus Device

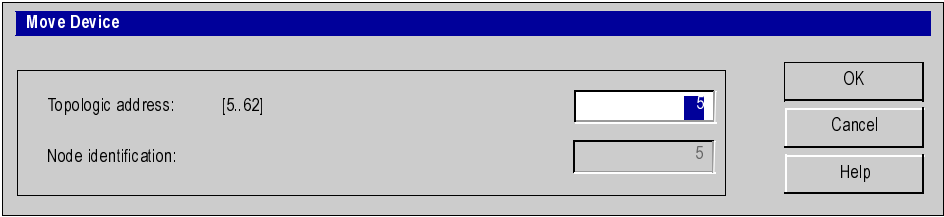
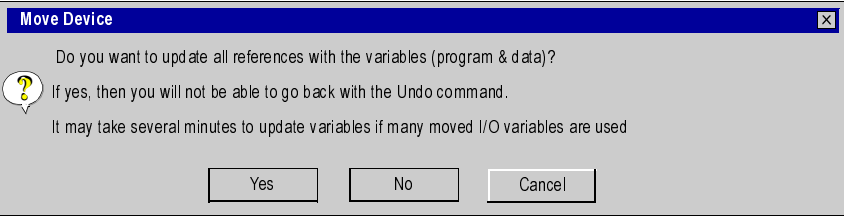
### Procedure for Deleting a Device

This operation is used to delete, via the software, a device connected to the CANopen bus:

Step	Action
1	Access the CANopen configuration screen.
2	Right-click on the connection point of the device to be deleted, then click on <b>Delete the drop</b> .

### Procedure for Moving a Device

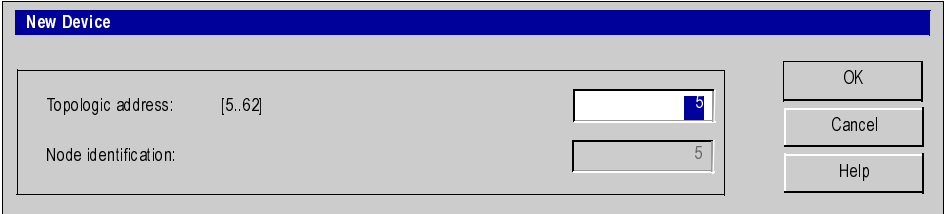
Moving a device does not involve a physical move on the bus, but rather a change in the device address logic. A movement thus triggers modification of the address of inputs/outputs objects in the program and movement of the variables associated with these objects.

Step	Action
1	Access the CANopen configuration screen.
2	Select the connection point to be moved (a frame surrounds the selected connection point).
3	<p>Drag and drop the connection point to be moved to an empty connection point.  <b>Result:</b> the <b>Move Device</b> screen appears:</p> 
4	Enter the number of the destination connection point.
5	<p>Confirm the new connection point by pressing <b>OK</b>.  <b>Result:</b> the <b>Move Device</b> screen appears:</p> 
6	Confirm the modification by pressing <b>Yes</b> to modify the addresses of the inputs/outputs objects in the program and move the variables associated with these objects.



**Procedure for  
Duplicating a  
Device**

This feature is similar to the function for moving a device:

Step	Action
1	Access the CANopen configuration screen.
2	Right-click on the device to be copied, then click on <b>Copy</b> .
3	Right-click on the connection point desired, then click on <b>Paste</b> . <b>Result:</b> the <b>New Device</b> screen appears: 
4	Enter the number of the destination connection point.
5	Confirm the new connection point by pressing <b>OK</b> .

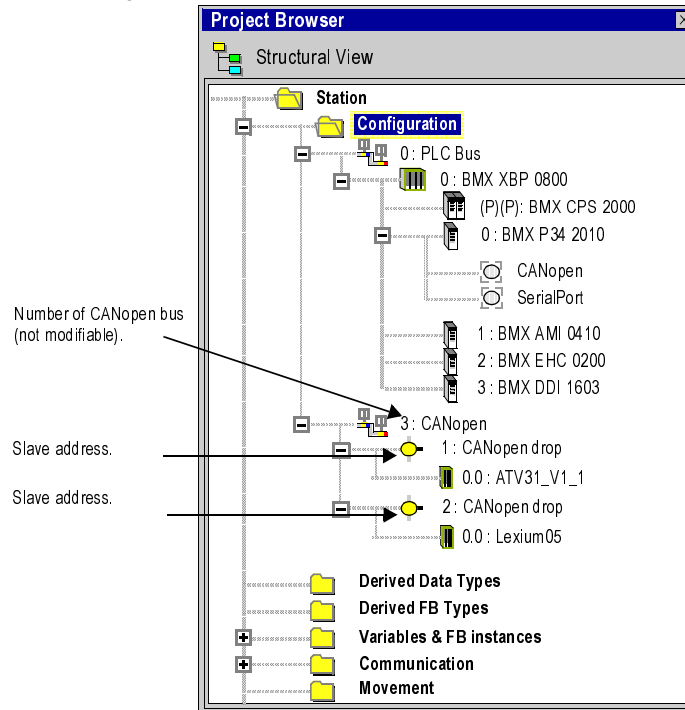
## View CANopen Bus in the Project Browser

### At a Glance

The CANopen bus is shown in the configuration directory in the project browser. The number of the bus is calculated automatically by Unity Pro.

**Note:** The value of the bus number cannot be modified.

The following illustration shows the CANopen bus and slaves in the project browser:



# 5.3 Device Configuration

## At a Glance

**Subject of this Section** This section presents the configuration of the initial parameters of the CANopen devices.

There are three ways of configuring the initial parameters:

- Configuration using Unity,
- Configuration using an external tool,
- Manual Configuration.

**Note:** Before configuring a device, it is strongly recommended to select the function, when available.

**What's in this Section?** This section contains the following topics:

Topic	Page
Slave Functions	76
Configuration Using Unity	79
Configuration Using an External Tool: Configuration Software	85
Manual Configuration	89

## Slave Functions

### At a Glance

So as to facilitate their configuration, certain CANopen devices are represented through functions.

Each function defines premapped PDOs, as well as certain debugging variables which can be mapped (**PDO** tab of the slave configuration screen).

**Note:** The function should be selected before configuring the slave.

### Available Functions

The available functions are as follows:

Function	Description	Devices involved
Basic	This function allows a simple control of the speed.	Altivar
Standard	This function allows control of the speed and/or torque. All the parameters that can be mapped are mapped in the supplemental PDOs for: <ul style="list-style-type: none"><li>● an adjustment of the operating parameters (length of acceleration,),</li><li>● additional surveillance (current value,...),</li><li>● additional control (PID, outputs command,...).</li></ul>	
Advanced	This function allows control of the speed and/or torque. Certain parameters can be configured and can also be mapped in the PDOs to allow: <ul style="list-style-type: none"><li>● an adjustment of the operating parameters (length of acceleration,),</li><li>● additional surveillance (current value,...),</li><li>● additional control (PID, outputs command,...).</li></ul>	

Function	Description	Devices involved
Simple	<p>Use this profile if the island does not contain high resolution analog I/O module or the TeSys U STB modules.</p> <p>This profile contains:</p> <ul style="list-style-type: none"> <li>● NIM diagnostic information (index 4000-index 4006),</li> <li>● discrete input information (index 6000),</li> <li>● 16 bit discrete information (index 6100),</li> <li>● discrete output information (index 6200),</li> <li>● 16 bit discrete output information (index 6300),</li> <li>● low resolution analog input information (index 6401),</li> <li>● low resolution analog output information (index 6411).</li> </ul>	STB NCO1010 & NCO2212
Extended	<p>Use this profile if the island contains high resolution analog I/O module or the TeSys U STB modules.</p> <ul style="list-style-type: none"> <li>● NIM diagnostic information (index 4000-index 4006),</li> <li>● discrete input information (index 6000),</li> <li>● 16 bit discrete information (index 6100),</li> <li>● discrete output information (index 6200),</li> <li>● 16 bit discrete output information (index 6300),</li> <li>● low resolution analog input information (index 6401),</li> <li>● low resolution analog output information (index 6411),</li> <li>● high resolution analog input information (index 2200-221F),</li> <li>● high resolution analog output information (index 3200-321F),</li> <li>● TeSys U input information (index 2600-261F),</li> <li>● TeSys U output information (index 3600-361F).</li> </ul>	
Advanced	<p>Use this profile if the island contains high resolution analog I/O module or HMI or the TeSys U STB modules.</p> <p>This profile contains:</p> <ul style="list-style-type: none"> <li>● NIM diagnostic information (index 4000-index 4006),</li> <li>● discrete input information (index 6000),</li> <li>● 16 bit discrete information (index 6100),</li> <li>● discrete output information (index 6200),</li> <li>● 16 bit discrete output information (index 6300),</li> <li>● low resolution analog input information (index 6401),</li> <li>● low resolution analog output information (index 6411),</li> <li>● high resolution analog input information (index 2200-221F),</li> <li>● high resolution analog output information (index 3200-321F),</li> <li>● TeSys U input information (index 2600-261F),</li> <li>● TeSys U output information (index 3600-361F),</li> <li>● 3rd party CANopen devices (index 2000-201F),</li> <li>● RTP information (index 4100 &amp; index 4101).</li> </ul>	STB NCO2212
Controlling	This function is especially created for CANopen communications with the built-in controller card and all the application cards (pump control,...).	Altivar 61/ 71

Function	Description	Devices involved
Basic	The basic level is designed to configure the valve terminal without CP extension.	Festo CPV
CP_Extension	This level is designed to configure I/Os including the CP extension.	
Basic_DIO_only	The basic level is designed to configure the CPX with pneumatic valves and Digital I/O only.	Festo CPX
Generic_DIO_AIO	The generic DS401 level is designed to configure CPX valves and I/Os, including Analogue I/O modules.	
Advanced	The advanced level is designed to configure the maximum I/Os and the complete parameters set.	
Default	This feature is the default function for certain devices. It may not be modified.	All the slaves except ATV and Lexium

**Note:** Some devices can only handle one function. In this case, the function appears grayed out and cannot be modified.

Function

Default

---

## Configuration Using Unity

---

### At a Glance

Some equipment can be configured directly from Unity:

- FTB
  - Osicoder
  - OTB
  - Preventa
  - STB NCO 1010
  - Tego Power
  - Festo Valve Terminal
  - Parker Moduflex
- 

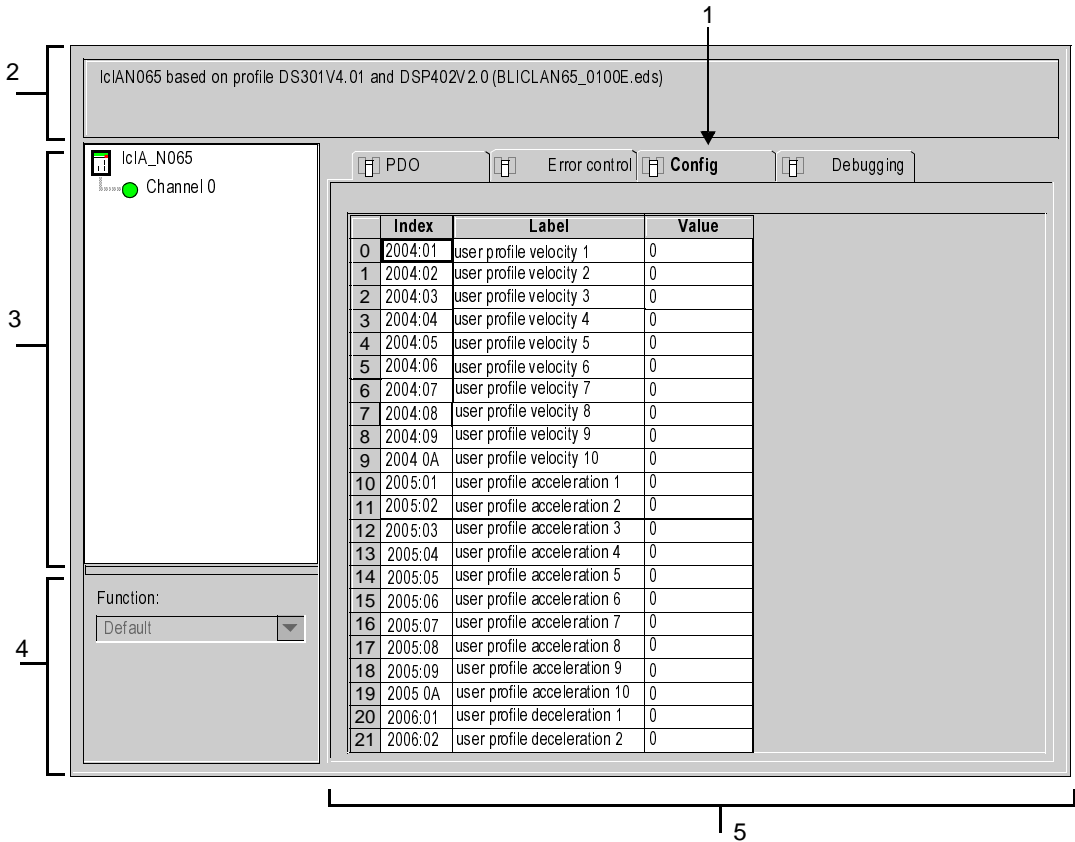
### Procedure

To configure a slave, perform the following actions:

Step	Action
1	Access the CANopen (see <i>How to Access the CANopen Bus Configuration Screen, p. 68</i> ) bus configuration screen.
2	Double-click on the slave to be configured.
3	Configure the usage function using the <b>Config</b> tab.
4	Configure the PDOs using the <b>PDO</b> tab.
5	Select the error control using the <b>Error control</b> tab.

---

**Config tab**      The following figure shows an example of the configuration screen of a slave:



The next table shows the various elements of the configuration screen and their functions:

Number	Element	Function
1	Tabs	The tab in the foreground indicates the type of screen displayed. In this case, it is the configuration screen.
2	Module area	Gives a reminder of the device's shortened name.



Number	Element	Function
3	<b>Channel area</b>	<p>This zone allows you to select the communication channel to be configured.</p> <p>By clicking on the device, you display the following tabs:</p> <ul style="list-style-type: none"><li>● <b>Description</b> : gives the characteristics of the device,</li><li>● <b>CANopen</b>: allows you to access SDO (in online mode),</li><li>● <b>I/O Objects</b>: allows pre-symbolizing of the input/output objects,</li><li>● <b>Fault</b>: accessible in online mode only.</li></ul> <p>By clicking on the channel, you display the following tabs:</p> <ul style="list-style-type: none"><li>● <b>PDO</b>(input/output objects)</li><li>● <b>Error control</b>,</li><li>● <b>Configuration</b>.</li><li>● <b>Debug</b> which can be accessed only in online mode.</li><li>● <b>Diagnostics</b>, accessible only in Online mode.</li></ul>
4	<b>General parameters area</b>	<p>This field allows you to select the slave function.</p>
5	<b>Configuration area</b>	<p>This area is used to set up the channels of the devices.</p> <p>Some devices can be configured with an external tool. In this case, the configuration is stored in the device and you cannot enter configuration parameters because this field is empty.</p>

**Note:** Refer to the documentation of each device for information on general, configuration, adjustment and debugging parameters.

**Note:** All parameters are not sent when the device takes its configuration. The CPU send only parameters which are different from the default values.

**PDO Tab**

PDOs make it possible to manage the communication flow between the CANopen Master and the slaves. The **PDO** tab allows to configure a PDO.

This screen is divided into 3 parts:

The screenshot displays the CANopen Configuration software interface, specifically the PDO tab. The interface is divided into three main sections:

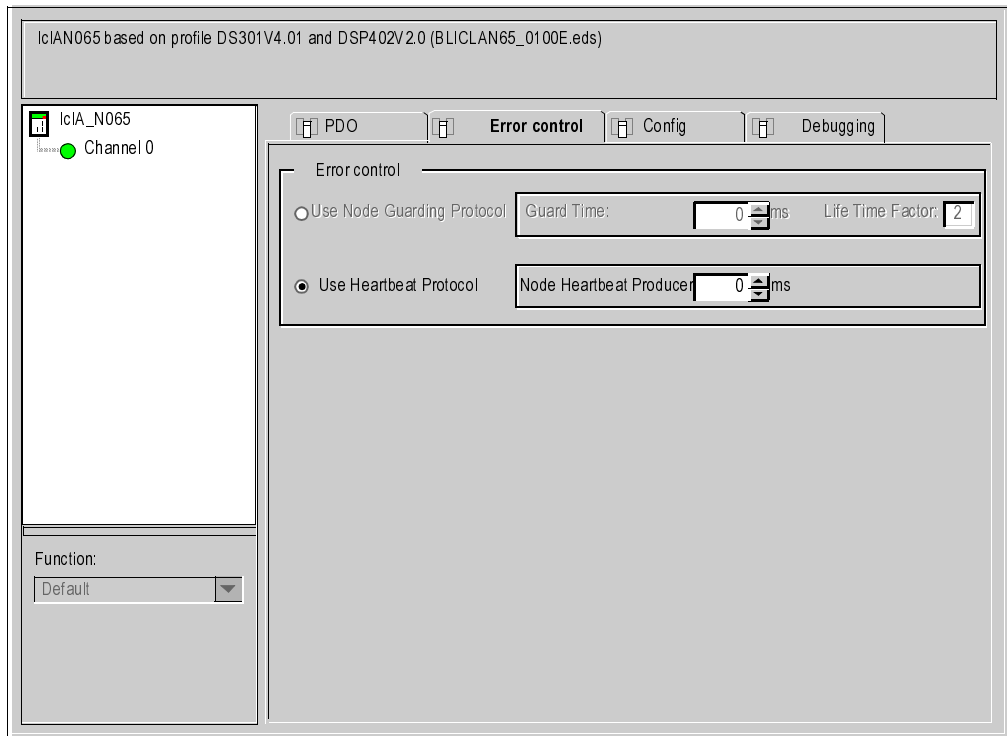
- Transmit (%I):** This section contains a table for configuring transmit PDOs. It lists four PDOs (PDO 1 to PDO 4) with their respective parameters: Tr. Ty, Inhibi, Even, Symbol, Topo. Addr, %M, CO, and Index. PDO 1 is selected, showing a status of 255, 0, 0, and a CO value of 16#181.
- Receive (%Q):** This section contains a table for configuring receive PDOs. It lists four PDOs (PDO 1 to PDO 4) with their respective parameters: Tr. Ty, Inhibi, Even, Symbol, Topo. Addr, %M, CO, and Index. PDO 1 is selected, showing a status of 255, 0, 0, and a CO value of 16#381.
- Variables:** This section on the right lists various parameters and their addresses, such as RAMPsym (3006:01), IO\_act (3008:01), ANA1\_act (3009:01), ANA2\_act (3009:05), PLCopenRx1 (301B:05), PLCopenRx2 (301B:06), PLCopenTx1 (301B:07), PLCopenTx2 (301B:08), JOGactivate (301B:09), \_actionStatus (301C:04), \_p\_actRAMPusr (301F:02), CUR\_target (3020:04), SPEEDn\_target (3021:04), PTPp\_abs (3023:01), PTPp\_relpref (3023:03), PTPp\_target (3023:05), PTPp\_relpact (3023:06), GEARdenom (3026:03), GEARnum (3026:04), Controlword (6040:00), Statusword (6041:00), and position actual valu... (6063:00).

- **Transmit PDOs:** information transmitted by the Slave to the Master,
- **Receive PDOs:** information received by the Slave from the Master,
- **Variables:** variables that can be mapped to the PDOs. To assign a variable to a PDO, drag and drop the variable into the desired PDO. No variable can be assigned with a static PDO.

**Note:** To configure the STB NCO 1010, it's necessary to determine all the objects that are valid for this device and to configure them manually in the PDOs. For more information about the list of the associated objects, please refer to the STB user manual.

For more information about the use of the PDOs.

**Error Control Tab** The **Error control** tab for CANopen slave modules allows you to configure fault monitoring.



Two choices are possible:

- Heartbeat:** The Heartbeat mechanism consists of sending cyclical presence messages generated by a Heartbeat Producer. A Heartbeat transmitter (producer) sends messages recurringly. The sending time is configured with the Node Heartbeat Producer Time Value. One or several elements connected to the network receive this message. The Heartbeat consumer surveys the Heartbeat message reception. If its duration exceeds the Heartbeat Consumer Time ( $1.5 * \text{Producer Heartbeat Time}$ ), an Heartbeat event is created and the device is in default. If a M340 Master PLC is used on the CANopen bus, all the nodes using the Heartbeat control mode are producers. The master surveys the transmission and the reception of the messages and it's the only receiver of the Heartbeat messages sent by the nodes. The Master can send Heartbeat messages to the slaves. The Master Heartbeat producer time is set at 300 ms and is not modifiable.

- **Node guarding:** Node Guarding is the monitoring of network nodes. The NMT (Network Management) master sends an RTR (Remote Transmission Request) at regular intervals (this period is called Guard Time) and the concerned node must answer in a given time lapse (the Node Life Time equals the Guard Time multiplied by the Life Time Factor).

The Life Time value is set at 2 and is not modifiable.

**Note:** Some devices only support Heartbeat or Node Guarding. For devices which support Heartbeat and Node Guarding, the only choice in Unity Pro is the Heartbeat mechanism.

---

## Configuration Using an External Tool: Configuration Software

---

### At a Glance

To configure a STB NCO 2212, a Lexium 05/15, an ICLa, a Tesys U or an ATV61/71 device, it is necessary to use an external tool:

- Advantys Configuration Software for the STB,
- PowerSuite for Lexium 05 V2.2.0 patch V2.2.0B Software for the Lexium 05,
- Powersuite V2.0 Software for the ATV31, ATV61, ATV71 and the Tesys U,
- UNILINK V1.5 for the Lexium 15 LP,
- UNILINK V4.0 for the Lexium 15 MH,
- ICLa CCT for the ICLa N065,
- EasyICla V1.104 for ICLa\_IFA, ICLa\_IFE, ICLa\_IFS.

**Note:** For motion and drive devices, it is highly recommended to use the software in conjunction with the Unity MFB in order to facilitate the configuration and programming.

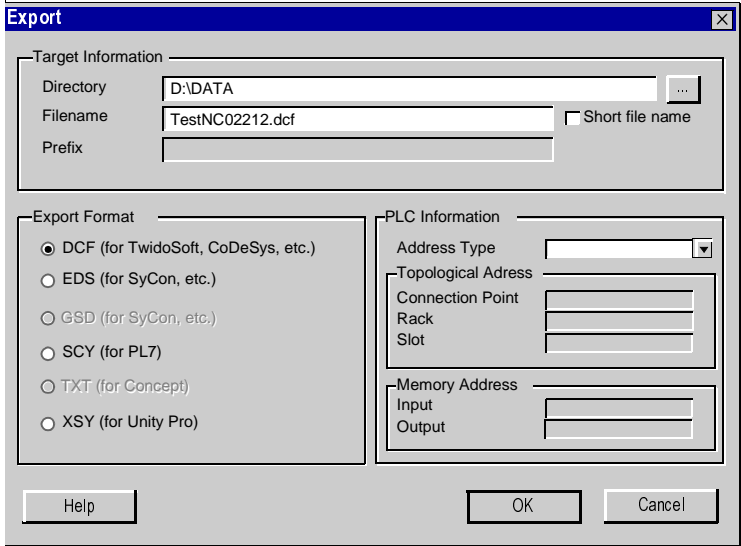
---

## Advantys Configuration Software

Advantys Configuration Software (Version 2.5) has to be used to configure a STB NCO 2212. The Advantys Configuration Software validates the configuration and creates a DCF file that contains all the objects used in the configuration ordered in the proper sequence. The DCF file can be import from Unity Pro.

**Note:** The creation of the DCF file is only possible from the full version of Advantys.

The procedure for adding an island to a CANopen bus is as follows:

Step	Action
1	In Advantys Configuration Software (Version 2.2 or above), create a new Island.
2	Select the STB NCO 2212 Network Interface Module.
3	Select the modules which will be used in the application.
4	Configure the island.
5	When the configuration is over, click on <b>File/Export</b> to export the island in DCF format. The following window is displayed:
	
6	Click <b>OK</b> to confirm.
7	Once the file is exported, launch Unity Pro and open the project in which the island will be used.
8	Add a STB device to the Bus Editor (see <i>How to Add a Device on the Bus</i> , p. 70).
9	Right-click on the STB device, then click on <b>Open the module</b> .
10	In the PDO tab, click the button <b>Import DCF</b> .

---

Step	Action
11	Confirm by clicking <b>OK</b> . The PDOs are configured automatically.

**Note:** The modification of the topology of an island requires recommencing this procedure.

For more information about the STB configuration, please refer to the STB user manual.

---

### **Powersuite Software**

The PowerSuite software development is a tool meant to implement the following Altivar speed drives. It should be used to configure an ATV31/61/71, a Tesys U or a Lexium 05 device (Powersuite 2)

Various functions are integrated for being used on implementing phases such as:

- configurations preparations,
- setting to work,
- maintenance.

The configuration is directly stored in the device.

For more information about the configuration of an ATV31/61/71 and Tesys U using Powersuite Software or about the configuration of a Lexium 05 with Power Suite 2, please refer to the device user manual.

---

### **UNILINK Software**

UNILINK provides simplified parameter setting for Lexium 05 servo drives. It's used to configure, sets and adjusts Lexium MHDA/MHDS drives according to the associated SER/BPH brushless motor and the application requirements.

For more information about the configuration of a Lexium 15 using UNILINK, please refer to the Lexium user manual.

---

## **ICLA CCT Software**

The ICLA CCT software is used to configure an ICL A N065. It includes a graphical user interface and can be used for commissioning, diagnostics and testing.

ICLA CCT offers the following functions:

- Input and display of device parameters,
- Archiving and duplication of device parameters,
- Display of status and device information,
- Positioning of the motor with the PC,
- Initialisation of reference movements,
- Access to all documented parameters,
- Diagnosis of operational malfunctions.

For more information about the configuration of an ICL A N065 using ICLA CCT, please refer to the software user manual.

<p><b>Note:</b> To configure an ICL A N065 using ICLA CCT, an USB/CANopen converter is needed.</p>
--



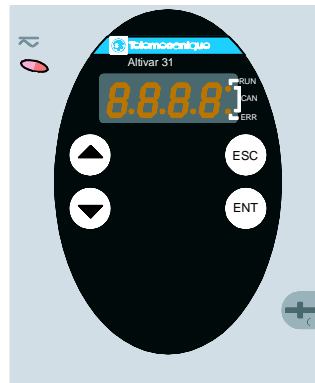
## Manual Configuration

### At a Glance

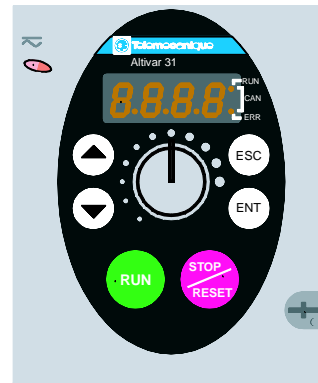
ATV 31 and Icla (except N065) devices can be configured manually from their front panel.

### Configuration of the ATV 31

The following figure presents the different front panels of the ATV 31 servodrive.



ATV31xxxxx



ATV31xxxxxA

The ATV 31 may be configured as follows:

Step	Action
1	Press on the "ENT" key to enter the ATV31 configuration menu.
2	Use the "Arrows" keys to select the "COM" Communication menu then confirm using the "ENT" key.
3	Use the "Arrows" keys to select the "AdCO" menu then confirm using the "ENT" key. Enter a value (Address on the CANopen bus). Confirm using the "ENT" key then exit the menu using the "ESC" key
4	Use the "Arrows" keys to select the "bdCO" menu then confirm using the "ENT" key. Enter a value (Speed on the CANopen bus). Confirm using the "ENT" key then exit the menu using the "ESC" key
5	Press several times on the "ESC" key to exit the configuration menu.

**Note:** The configuration may be modified only when the motor is stopped and when the variable speed controller is locked (cover closed). Any modification entered will become effective after an "Off/On" cycle of the speed controller.  
For more information about the ATV31 configuration, please refer to the Altivar speed drive user manual.

---

## Configuration of Icla Devices

Icla devices, except the Icla N065, have a switch to configure the address and the speed.

**Note:** For more information about Icla configuration, please refer to the Icla user manual.

---

# 5.4 Master Configuration

## At a Glance

**Subject of this Section** This section presents the master configuration.

**What's in this Section?** This section contains the following topics:

Topic	Page
How to Access the CANopen Master Configuration Screen	92
CANopen Master Configuration Screen	94
Description of Master Configuration Screen	96

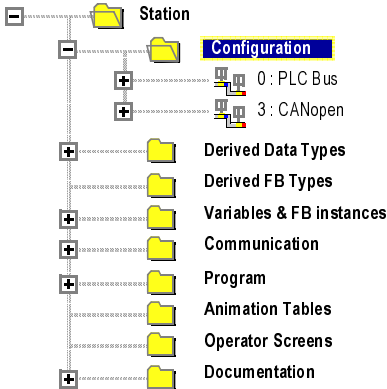
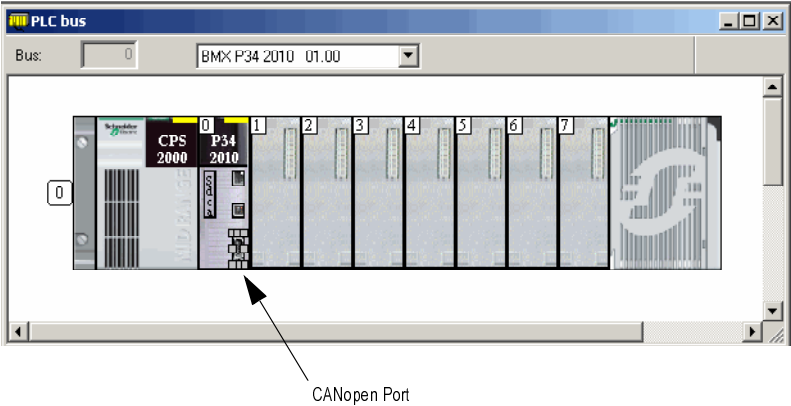
## How to Access the CANopen Master Configuration Screen

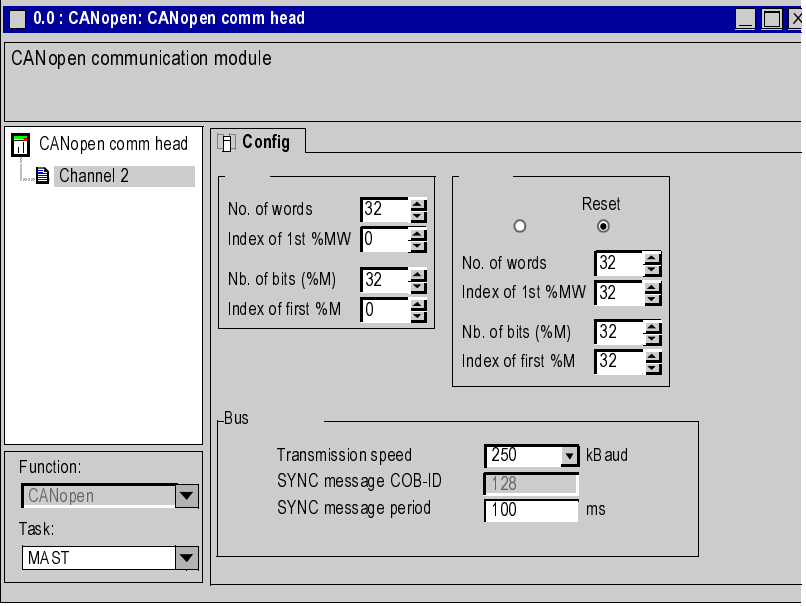
**At a Glance**

This describes how to access the configuration screen of the master for a Modicon M340 PLC with a built-in CANopen link.

**Procedure**

To access the master, carry out the following actions:

Step	Action
1	<p>From the project navigator, deploy the <code>Configuration</code> directory.</p> <p><b>Result:</b> the following screen appears:</p> 
2	<p>Double-click on the <code>PLC Bus</code> subdirectory.</p> <p><b>Result:</b> the following screen appears:</p>  <p>Double-click on the processor's CANopen port.</p>

Step	Action
3	<p>The master configuration screen appears:</p> 

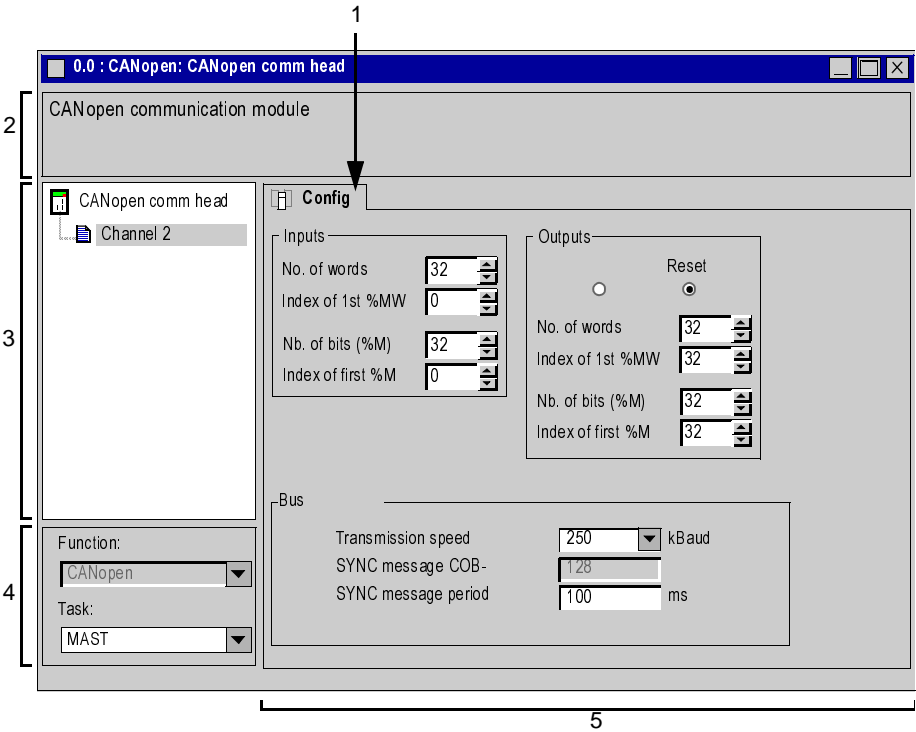
# CANopen Master Configuration Screen

**At a Glance**

This screen is used to declare and configure the master of the CANopen network from a Modicon M340 PLC station.

**Illustration**

The configuration screen of the master is as follows:



## Elements and functions

The table below describes the different areas which make up the master configuration screen:

Read	Number	Function
1	Tab	The tab in the foreground indicates the type of screen displayed. In this case, it is the configuration screen.
2	Module	This area is made up of the abbreviated heading of the processor equipped with a CANopen port.
3	Channel	<p>This zone allows you to select the communication channel to be configured.</p> <p>By clicking on the device, you display the tabs:</p> <ul style="list-style-type: none"> <li>● <b>Description</b> : gives the characteristics of the built-in CANopen port,</li> <li>● <b>Inputs/outputs objects</b>: allows pre-symbolizing of the input/output objects,</li> </ul> <p>By clicking on a channel, you display the tabs:</p> <ul style="list-style-type: none"> <li>● <b>Config</b> . enables you to declare and configure the CANopen master,</li> <li>● <b>Debug</b> : accessible in online mode only,</li> <li>● <b>Fault</b>: accessible in online mode only.</li> </ul>
4	General parameters	<p>This field enables you to:</p> <ul style="list-style-type: none"> <li>● choose the communication function (non modifiable).</li> <li>● associate the CANopen bus to an application task: <ul style="list-style-type: none"> <li>● MAST which is the master task,</li> <li>● FAST which is the rapid task.</li> </ul> </li> </ul> <p>The tasks are asynchronous in relation to exchanges on the bus.</p>
5	Configuration	<p>This field enables you to:</p> <ul style="list-style-type: none"> <li>● configure the PLC internal memory addresses where inputs from the CANopen devices will periodically be copied.</li> <li>● configure the PLC internal memory addresses where outputs from the CANopen devices will periodically be read.</li> <li>● configure the parameters of the CANopen bus.</li> </ul>

## Description of Master Configuration Screen

---

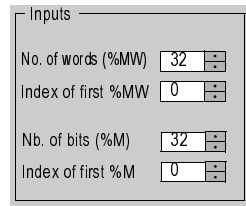
### At a Glance

The configuration screen allows configuration of the bus parameters as well as the inputs and outputs.

---

### Inputs

The figure below illustrates the inputs configuration area:



The screenshot shows a configuration window titled "Inputs". It contains four rows of configuration fields:

Field	Value
No. of words (%MW)	32
Index of first %MW	0
Nb. of bits (%M)	32
Index of first %M	0

To configure the inputs of the bus slaves, it is necessary to indicate the memory areas to which they will be periodically recopied. To define this zone, you must indicate:

- a number of words (%MW): from 0 to 32,464,
  - the address of the first word: from 0 to 32,463,
  - the number of bits (%M): from 0 to 32,634,
  - the address of the first bit: from 0 to 32,633.
-



## Outputs

The figure below illustrates the outputs configuration area:

The fallback mode (maintain/reset) allows to define the behaviour of the device when the CPU is in STOP or in HALT:

- **Maintain:** maintain of outputs (values are kept),
- **Reset:** reset of outputs (values are set to 0).

To configure the outputs, it is necessary to indicate, as for the inputs, the word and bits table that will contain the values of the bus slave outputs:

- a number of words (%MW): from 1 to 32,464,
- the address of the first word: from 0 to 32,463,
- the number of bits (%M): from 1 to 32,634,
- the address of the first bit: from 0 to 32,633.

**Note:** the word tables and bit tables are found in the PLC internal memory. Any crossover between two areas of each table is prohibited. The bits area for the inputs cannot overlap the bits area for the outputs. The words area for the inputs cannot overlap the words area for the outputs.

## ⚠ WARNING

### UNEXPECTED EQUIPMENT OPERATION

Take every precaution at the installation to have the outputs' position safe in case of CANopen bus stopping. When the CANopen bus stops, the behaviour is specific to the equipments connected. See the user manual of those equipments.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

Bus Parameters

The figure below illustrates the bus parameters configuration area:

Bus parameters

Transmission speed

250

kBaud

SYNC message COB-ID

128

SYNC message period

100

ms

To configure the bus, it is necessary to indicate:

- the transmission speed (see *Bus Length*, p. 23): 250 kBauds default,
- the COB-ID of the synchronization message: 128 default,
- the synchronization message period: 100 ms default.

Language Objects

The parameters presented below are represented in the %KW language objects:

Read	Parameter	Language object
Inputs	Number of words %MW	%KW8
	Index of the first word	%KW10
	Number of bits %M	%KW4
	Index of the first bit	%KW6
Outputs	Fallback mode	%KW0
	Number of words %MW	%KW9
	Index of the first word	%KW11
	Number of bits %M	%KW5
	Index of the first bit	%KW7
Bus parameters	Transmission speed	%KW1
	SYNC message COB-ID	%KW2
	SYNC message period	%KW3

---

## At a Glance

**Introduction** This section describes the programming of a CANopen architecture.

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

Topic	Page
Exchanges Using PDOs	100
Exchanges Using SDOs	105
Communication functions exemple	108
Modbus request example	115

## Exchanges Using PDOs

### At a Glance

PDOs use topologic addresses (%I, %IW, %Q, %QW) and internal variables (%M or %MW).

Topologic address

Internal variable

Transmit (%I)

PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	%M...
<input checked="" type="checkbox"/> PDO 1(...)	255	0	0			
<input checked="" type="checkbox"/> Status...				lexium...	%IW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> PDO 2...	255	0	100			
<input checked="" type="checkbox"/> Status...				lexium...	%IW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> Positi...				lexium...	%DI\3.1\0.0.0.8	%MW8
<input checked="" type="checkbox"/> PDO 3...	255	0	100			
<input checked="" type="checkbox"/> Status...				lexium...	%IW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> Veloci...				lexium...	%DI\3.1\0.0.0.10	%MW10
<input checked="" type="checkbox"/> PDO 4...	254	0	0			

Receive (%Q)

PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	%M...
<input checked="" type="checkbox"/> PDO 1(...)	255	0	0			
<input checked="" type="checkbox"/> Status...				lexium...	%QW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> PDO 2...	255	0	100			
<input checked="" type="checkbox"/> Status...				lexium...	%QW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> Positi...				lexium...	%QD\3.1\0.0.0.8	%MW8
<input checked="" type="checkbox"/> PDO 3...	255	0	100			
<input checked="" type="checkbox"/> Status...				lexium...	%QW\3.1\0.0.0.16	%MW16
<input checked="" type="checkbox"/> Veloci...				lexium...	%QD\3.1\0.0.0.10	%MW10
<input checked="" type="checkbox"/> PDO 4...	254	0	0			

Variables

☐ Display only unmapped varia

Parameter Name	Ind...
RAMPsym	3006:01
_IO_act	3008:01
ANA1_act	3009:01
ANA2_act	3009:05
PLCopenRx1	301B:05
PLCopenRx2	301B:06
PLCopenTx1	301B:07
PLCopenTx2	301B:08
JOGactivate	301B:09
_actionStatus	301C:04
_p_actRAMPusr	301F:02
CUR_I_target	3020:04
SPEEDn_target	3021:04
PTPp_abs	3023:01
PTPp_relpref	3023:03
PTPp_target	3023:05
PTPp_relpact	3023:06
GEARdenom	3026:03
GEARnum	3026:04
Controlword	6040:00
Statusword	6041:00
position actual valu	6063:00

There is an equivalence between topologic addresses and internal variables. For example, in the figure above, the topologic address %IW\3.1\0.0.0.16 is equivalent to %MW16 for the PDO 1.

A PDO can be enabled or disabled.

According with the EDS file, some PDOs are already mapped.

A double click on the `transmission type` column displays the following window:

This window allows to configure:

- the transmission type:
  - synchronous acyclic: a transmission type of 0 means that the message shall be transmitted synchronously with the SYNC message but not periodically according with the value.
  - synchronous cyclic: a value between 1 and 240 means that the PDO is transmitted synchronously and cyclically, the transmission type value indicating the number of SYNC messages between two PDO transmissions.
  - asynchronous PDO: the transmission type 254 means that the PDO is transmitted asynchronous. It is fully depending on the implementation in the device. mainly used for digital I/O.
  - synchronous PDO: the transmission type 255 means that the PDO is transmitted asynchronous when the value change.

Verify that the configured transmission type is supported by the selected device.

- the inhibit time: mask the communication during this time),
- the event timer: (time to manage an event in order to start a PDO).

**Note:** PDOs can only be configured using Unity Pro.

## Structure of Topologic Address

The topologic address of input/output objects of a CANopen bus slave is structured in the following way:

**% I, Q X, W, D, F \ b.e \ r . m . c . d**

Family	Element	Values	Meaning
Symbol	%	-	Indicates an IEC object.
Object type	I	-	Input object.
	Q	-	Output object.
Format (size)	X	8 bits (Ebool)	Ebool type Boolean (not compulsory).
	W	16 bits	16 bit WORD-type word.
	D	32 Bit	32 bit DINT-type word.
	F	32 Bit	32 bit REAL-type word.
Module/channel address and connection point	b	3 to 999	Bus number.
	e	1 to 63	Connection point number (CANopen slave number).
Rack number	r	0	Virtual rack number, always 0.
Module number	m	0	Virtual module number, always 0.
Channel number	c	Equal to 0 for all devices except the FTBs (channels numbered 0 to 7, then from 10 to 17).	Channel number.
Rank of data in the channel	d	0...999	Data number of slave. This number can vary from 0 to 999 because a slave can only have a maximum of 1000 input and output words.

## Example of Topologic Addressing

Example of topologic addressing of an item connected to point 4 of the CANopen bus number 3:

<b>Module digital/TOR autonomous with Boolean vision</b>	
<b>%I\3.4\0.0.5</b>	Boolean value is entered on channel 5 (rang 0 omitted).
<b>Module digital standard</b>	
<b>%IW\3.4\0.0.0.2.5</b>	Boolean value is entered on unique channel 0, rank 2, bit 5. Themapping is given when the DCF file is imported.
<b>Digital module on an Advantys STB island</b>	
<b>%IW\3.4\0.0.0.3.2</b>	Word 3, bit 2, data by Advantys Configuration Software.

Numbering starts at:

- 0 for channel,
- 0 for rank.

**Note:** Virtual objects (racks, modules) always have a rank number equal to 0.

Object addressing of CANopen digital input/output follows the same rules as object addressing of digital input/output on rack: words, double words and floaters are in the same block.

Example: device at connection point 4 of CANopen bus 3, on channel 0, with:

<b>Type of data</b>	<b>Topologic address:</b>
2 input words	%IW \3.4\0.0.0.0 or %IW \3.4\0.0.0.1
1 double input word	%ID \3.4\0.0.0.2
1 floating input	%IF \3.4\0.0.0.4
1 output word	%QW\3.4\0.0.0.6

An object can be mapped in a PDO only once. If the same object is mapped several times in the same PDO, Unity Pro displays an error message.

If there's several PDOs with the same mapped object, only one PDO can be enabled. If several PDOs with the same mapped object are enabled, Unity Pro displays an error message when the application is rebuilt.

Exemple with a Lexium 05:

Error: the same object is mapped in two enabled PDO

PDO						
Transmit (%I)						
PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	
<input checked="" type="checkbox"/> PDO 1 (Static)	255	0	0			
<input checked="" type="checkbox"/> Statusword					%IW13.110.0.0.16	
<input checked="" type="checkbox"/> PDO 2 (Static)	255	0	100			
<input checked="" type="checkbox"/> Statusword					%IW13.110.0.0.16	
<input checked="" type="checkbox"/> Position actual...					%D13.110.0.0.8	
<input type="checkbox"/> PDO 3 (Static)	255	0	100			
<input type="checkbox"/> Statusword					%IW13.110.0.0.16	
<input type="checkbox"/> Velocity actual...					%D13.110.0.0.10	
<input type="checkbox"/> PDO 4...	254	0	0			

Receive (%Q)						
PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	
<input checked="" type="checkbox"/> PDO 1 (Static)	255					
<input checked="" type="checkbox"/> Controlword					%IW13.110.0.0.16	
<input type="checkbox"/> PDO 2 (Static)	255					
<input type="checkbox"/> Controlword					%IW13.110.0.0.16	
<input type="checkbox"/> Target position					%D13.110.0.0.8	
<input type="checkbox"/> PDO 3 (Static)	255					
<input type="checkbox"/> Controlword					%IW13.110.0.0.16	
<input type="checkbox"/> Target velocity					%D13.110.0.0.10	
<input type="checkbox"/> PDO 4...	254					

No error: only one PDO is enabled.



---

## Exchanges Using SDOs

---

### At a Glance

The explicit exchange of messages on a CANopen bus is done by read/write protocol.SDO.

There are 3 ways of accessing SDOs:

- using communication functions `READ_VAR` and `WRITE_VAR`,
- using the Unity Pro debugging screen,
- using the request ModBus FC43/0xD.

### WARNING

#### UNEXPECTED EQUIPMENT OPERATION

When modifying a variable, check the consequences of the SDO command in the documentation of the specific target CANopen device.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

---

### Communication Functions

It is possible to access SDOs using the communication functions `READ_VAR` and `WRITE_VAR`.

**Note:** It is possible to send up to 16 `READ_VAR`/`WRITE_VARS` simultaneously. Only one SDO is exchanged at the same time on the bus. It is necessary to await the end of the preceding exchange to begin a new exchange. The end of exchange polling is carried out at each task cycle, so there is one SDO exchange for each task cycle.

For more information about the use of the communication function, see *Communication functions exemple, p. 108*

**Note:** Changing outputs of a device with a write SDO has no effect on the %QW.

---

## Unity Pro

SDO objects allow the access to the variables.

In online mode, the **CANopen** screen (see *Slave Diagnostics*, p. 128 ) allows access to:

- various device objects in read/write mode (only through a listbox),
- description of the variables,
- repeat of communication.

The **CANopen** screen is brought up as follows:

lcIA-IFA CANOpen (lcLA\_IFA.eds)

lcIA\_IFA  
Channel 0

Description | **CANopen** | I/O objects

CANopen slave details

Device name: lcIA\_IFA

Vendor Name: BERGER LAHR

Description: lcIA-IFA (lcLA\_IFA)

Request to send

Request to send: Lecture SDO Value: (120 bytes max.) 16#

Index: 16# Parameter name

Sub-index: 16# Parameter size (Byte)

Send Status

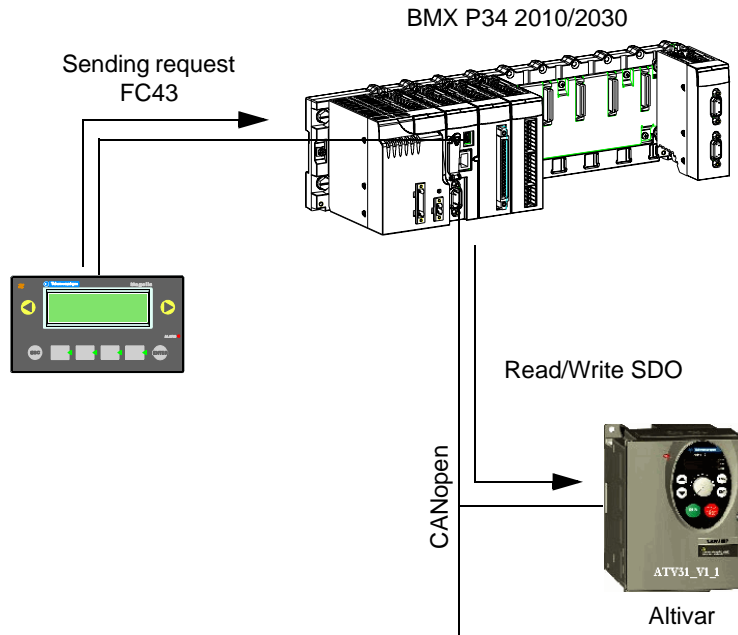
Response received

The value is displayed in the following way:

LSB		...		MSB
-----	--	-----	--	-----

SDO information (read or written) are displayed in a byte format. For 16-bit or 32-bit information, the low byte is displayed first (eg: 0102 in hexadecimal will be displayed as 02 01).

**Modbus Request** From a Human/Machine interface (example: XBT), it is possible to access the SDOs using the Modbus FC43 request



For more information about the use of the Modbus request FC43/0xD, see *Modbus request example, p. 115*

**SDO Timeouts** Various timeouts are implemented. They depend on the type of object as well as the type of access (read/write):

Object	Timeout
1010h	15 s
1011h	3 s
2000h to 6000h	8 s
All other objects	
- SDO Reading	1 s
- SDO Writing	2 s

## Communication functions exemple

### At a Glance

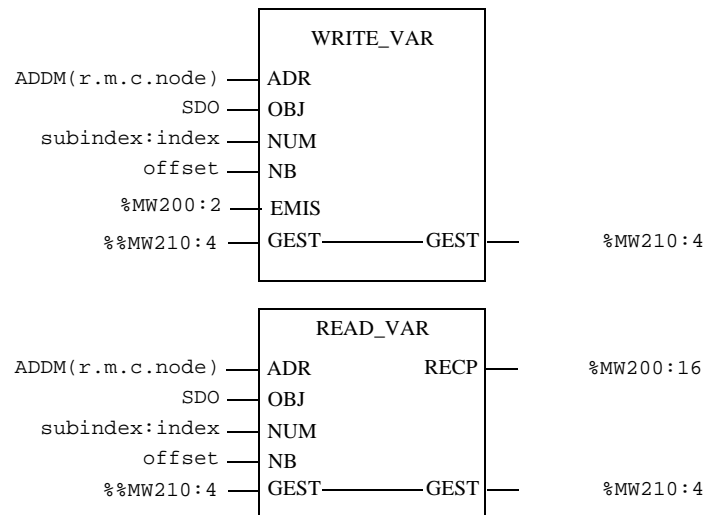
It is possible to access SDOs using the communication functions READ\_VAR and WRITE\_VAR

There are 3 possible representations:

- the FBD representation,
- the Ladder representation,
- the IL representation.

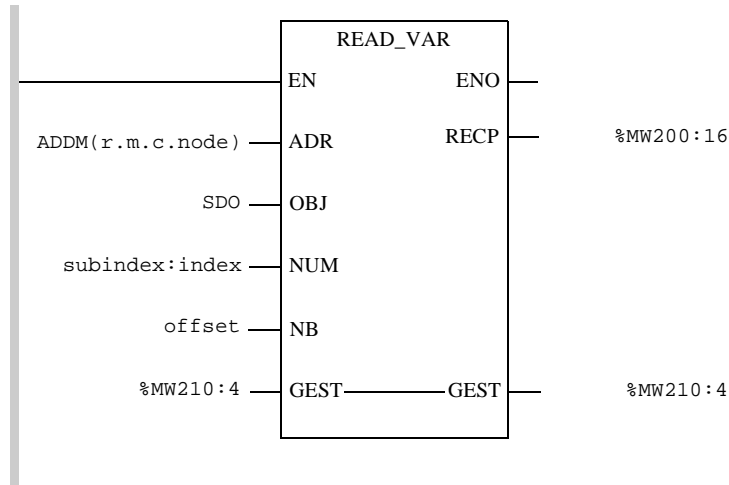
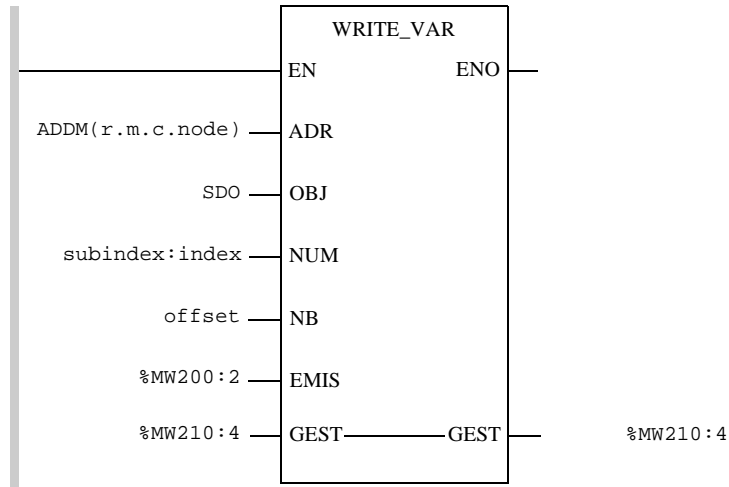
### FBD representation

The FBD representations of the communication functions are the following:



## Ladder representation

The Ladder representations of the communication functions are the following:



**IL representation**    The communication function syntax is as follows:

```

ADDM(
    IN := ' 0.0.2.2'
)

ST %MW2100:8

LD 50
ST %MW2182 (* timeout 5 secondes *)
LD 2
ST %MW2183 (* Length *)

(* Read the "Vendor ID" object, slave @2, CANopen Network *)
READ_VAR (
    ADR := %MW2100:8,
    OBJ := 'SDO',
    NUM := 16#00011018,
    NB := 0,
    GEST := %MW2120:4,
    RECP := %MW2110:4
)

(* Write the value 16#FFFF, slave @2 outputs, CANopen Network *)
LD 16#ffff
ST %MW2200

WRITE_VAR (
    ADR := %MW2100:8
    OBJ := 'SDO',
    NUM := 16#00016300,
    NB := 0,
    EMIS := %MW2200:1,
    GEST := %MW2180:4
)

```

**Note:** The `offset` parameter must be set to 0.

**Note:** The `subindex : index` parameter is encoded in a simple word (subindex is the higher byte).

### Parameter Description of the WRITE\_VAR Function

The following table outlines the various parameters of the `WRITE_VAR` function:

Parameter	Description
<code>ADDM( 'r.m.c.node' )</code>	Address of the destination entity of the exchange: <ul style="list-style-type: none"> <li>● <b>r</b>: the processor rack number,</li> <li>● <b>m</b>: processor slot in the rack (0)</li> <li>● <b>c</b>: channel (only use the channel 2 for CANopen),</li> <li>● <b>node</b>: identifier of the transmitting device on the CANopen bus.</li> </ul>
<code>'SDO'</code>	SDO object type.
<code>subindex:index</code>	Double word or immediate value identifying the CANopen SDO index or subindex: The most significant word making up the double word contains the sub-index and the least significant word contains the index. <b>Example:</b> if you use the double word <code>subindex:index</code> : <ul style="list-style-type: none"> <li>● the 16 most significant bits contain the subindex,</li> <li>● the 16 least significant bits contain the index.</li> </ul>
<code>EMIS</code>	Table of words containing the SDO datum to send (%MW200:2). The receipt buffer of the <code>WRITE_VAR</code> function must be greater than the SDO. The length of a SDO is indicated in device documentation.
<code>GEST</code>	Table of words with 4 inputs (%MW210:4).

### Parameter Description of the READ\_VAR Function

The following table outlines the various parameters for the `READ_VAR` function:

Parameter	Description
<code>ADDM( 'r.m.c.node' )</code>	Address of the destination entity of the exchange: <ul style="list-style-type: none"> <li>● <b>r</b>: the processor rack number,</li> <li>● <b>m</b>: processor slot in the rack (0)</li> <li>● <b>c</b>: channel (only use the channel 2 for CANopen),</li> <li>● <b>node</b>: identifier of the destination device on the bus.</li> </ul>
<code>'SDO'</code>	SDO object type.

Parameter	Description
subindex:index	<p>Double word or immediate value identifying the CANopen SDO index or subindex:</p> <p>The most significant word making up the double word contains the sub-index and the least significant word contains the index.</p> <p><b>Example:</b> if you use the double word subindex:index:</p> <ul style="list-style-type: none"> <li>the 16 most significant bits contain the subindex,</li> <li>the 16 least significant bits contain the index.</li> </ul>
GEST	Table of words with 4 inputs (%MW210:4).
RECP	<p>Table of words with at least one input to receive the SDO datum received (%MW200:16).</p> <p>The receipt buffer of the READ_VAR function must be greater than the SDO. The length of a SDO is indicated in device documentation.</p>

---



**Description of  
control block  
words**

The following table describes the various words of the control block:

Fields	Word	Type	Description
Control byte	0 (least significant)	BYTE	Bit 0 = activity bit Bit 1 = cancellation bit
Exchange ID	0 (most significant)	BYTE	Single number, identifier of the exchange.
ComState	1 (least significant)	BYTE	0x00 = Exchange terminated 0x01 = Time Out 0x02 = User cancelled 0x03 = Incorrect address format 0x04 = Incorrect destination address 0x06 = Incorrect Com Fb parameters 0x07 = Generic transmission problem 0x09 = Buffer received too small 0x0B = No system resources 0xFF = Network exchange error
ExchState	1 (most significant)	BYTE	If ComState = 0x00 : 0x00: request treated 0x01: Cannot be treated 0x02: Incorrect response If ComState = 0xFF 0x07: Generic exchange error 0x0B: The destination device has no more resources. 0x0D: The device cannot be reached. 0x2B: SDO exchange error
Timeout	2	WORD	Timeout value (x 100 ms)
Length	3	WORD	Length in bytes

**Example in ST language**

```
(* read the node 5 SDO, index 1018, subindex 3 *)
if (%M400) then
    subindex_index := 16#00031018 ;
    %MW1052 := 50; (* timeout 5 secondes *)

    READ_VAR(ADDM('0.0.2.5'),'SDO',subindex_index,0,%MW1050:4,%MW1100:2);

    %M400:= 0;
end_if;

(* Write the node 31 SDO, index 203C, subindex 2 *)
if (%M401) then
    subindex_index := 16#0002203C;
    %MW1152 := 50; (* timeout 5 secondes *)
    %MW1153 := 2; (* length 2 bytes *)
    %MW1200 := 16#03E8; (* value of object *)
    WRITE_VAR(ADDM('0.0.2.31'),'SDO',subindex_index,0,%MW1200:1,%MW1150:4);

    %M401:= 0;
end_if;
```

---

## Modbus request example

### At a Glance

From a Man/Machine interface (example : XBT), it is possible to access the SDOs using the Modbus FC43 request

### SDO read example

Node reading 1F, object 1005, subindex 00, length 8 bytes

FC	MEI	Prot	Nid	Index	Sub	Offset	Length
2B	0D	00	1F	10 05	00	00 00	00 08

Response OK: reception of 4 bytes

FC	MEI	Prot	Nid	Index	Sub	Offset	Length	Object value
2B	0D	00	1F	10 05	00	00 00	00 04	80 00 00 00

Failure: SDO cancellation code

FC	MEI	Ext length	MEI	Excpt code	SDO abort code
AB	FF	00 06	0D	EC	06 02 00 00

### Write SDO example

Node reading 1F, object 203C, subindex 02, length 2 bytes 03 E8

FC	MEI	Prot	Nid	Index	Sub	Offset	Length	Data
2B	0D	01	1F	20 3C	02	00 00	00 02	03 E8

Response OK: reception of 4 bytes

FC	MEI	Prot	Nid	Index	Sub	Offset	Length
2B	0D	00	1F	20 3C	02	00 00	00 00

Failure: SDO cancellation code

FC	MEI	Ext length	MEI	Excpt code	SDO abort code
AB	FF	00 06	0D	EC	06 02 00 00



---

# Debugging Communication on the CANopen Bus

7

---

## At a Glance

### Aim of this Chapter

This chapter presents the debugging of the CANopen bus master and slaves.

### What's in this Chapter?

This chapter contains the following topics:

Topic	Page
How to Access the Debug Screens of Remote Devices	118
Debugging Screen of the CANopen Master	119
Slave Debug Screens	121

## How to Access the Debug Screens of Remote Devices

---

### At a Glance

The following operations describe how to access different debug screens of the CANopen network elements.

<b>Note:</b> The debug screens can only be accessed in online mode.
---

---

### Master Debug Screen

To access the master debug screen, perform the following actions:

Step	Action
1	Connect to the manager PLC.
2	Access the CANopen master configuration screen (see <i>How to Access the CANopen Master Configuration Screen</i> , p. 92).
3	Select the <b>Debug</b> tab.

---

### Slave Debug Screen

To access the slave debug screen, perform the following actions:

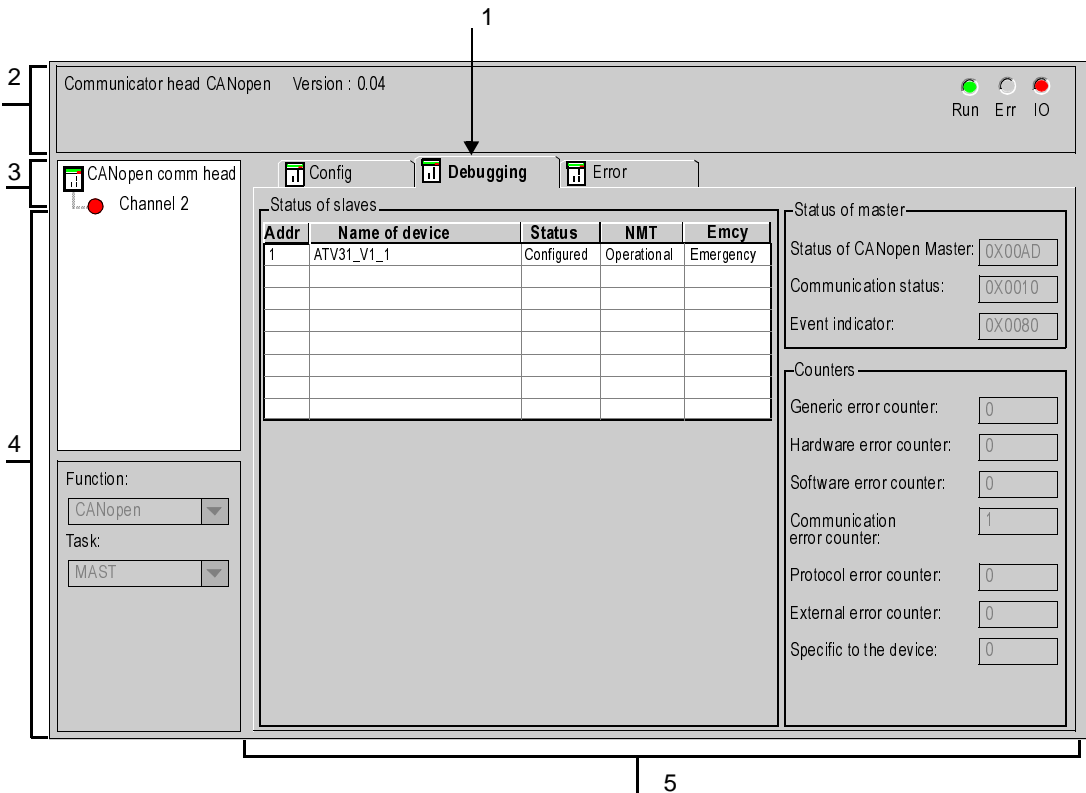
Step	Action
1	Connect to the manager PLC.
2	Access the CANopen slave configuration screen (see <i>Configuration Using Unity</i> , p. 79).
3	Select the <b>Debug</b> tab.

---

## Debugging Screen of the CANopen Master

**At a Glance** This screen can only be used in online mode.

**Illustration** The figure below shows a master debug screen:



**Elements and Functions**

The table below describes the different areas which make up the master debug screen:

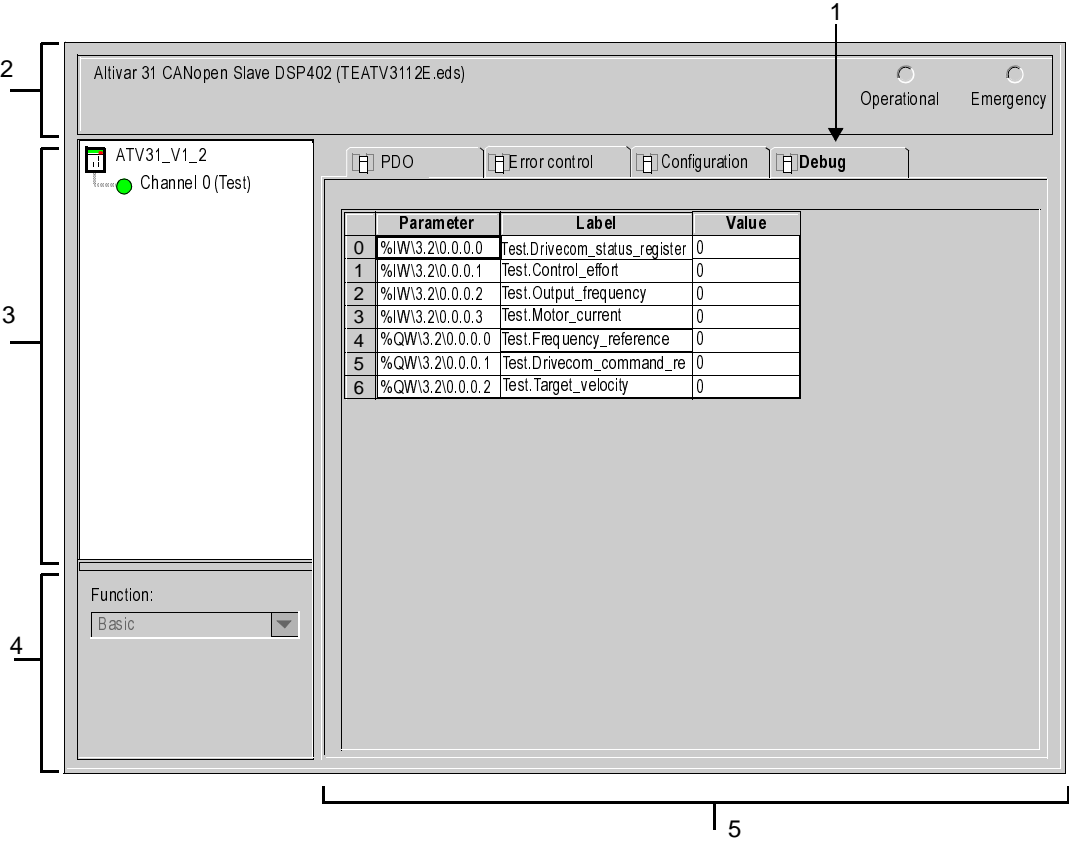
Read	Number	Channel
1	Tab	The tab in the foreground indicates the type of screen displayed. In this case, the debug screen.
2	Module	This area is made up of the abbreviated heading of the module equipped with a CANopen port, as well as 3 LEDs indicating the status of the module.
3	Channel	<p>This area allows you to select the communication channel to be debugged.</p> <p>By clicking on the device, you display the tabs:</p> <ul style="list-style-type: none"><li>● <b>Description</b> : gives the characteristics of the built-in CANopen port,</li><li>● <b>Inputs/outputs objects</b>: allows pre-symbolizing of the input/output objects,</li></ul> <p>By clicking on the channel, you display the tabs:</p> <ul style="list-style-type: none"><li>● <b>Configuration</b> : enables you to declare and configure the CANopen master,</li><li>● <b>Debug</b>: accessible in online mode only.</li><li>● <b>Faults</b>: accessible in online mode only.</li></ul> <p>This area also has an LED indicating the channel status.</p>
4	General parameters	<p>This area is used to view:</p> <ul style="list-style-type: none"><li>● the communication function,</li><li>● the task associated with the CANopen bus</li></ul>
5	Display and command	<p>This area is composed of 3 windows which let you know:</p> <ul style="list-style-type: none"><li>● the CANopen slaves status,</li><li>● the status of the CANopen master,</li><li>● the status of the error counters.</li></ul>



# Slave Debug Screens

**At a Glance** This screen can only be used in online mode.

**Illustration** The figure below shows a slave debug screen:



## Description of the Debug Screen for Standard Devices

The following table shows the various parts of the debugging screen and their functions:

Number	Element	Function
1	Tabs	The tab in the foreground indicates the type of screen displayed. In this case, the debug screen.
2	<b>Module</b> area	Contains the abbreviated title of the module. Two LEDs are found in the same area: <ul style="list-style-type: none"> <li>• a green LED indicating that the device is operational (ON/OFF),</li> <li>• a red LED indicating an emergency (ON/OFF).</li> </ul>
3	<b>Channel</b> area	This area allows you to select the communication channel to be debugged. By clicking on the device, you display the tabs: <ul style="list-style-type: none"> <li>• <b>Description:</b> gives the characteristics of the built-in CANopen port.</li> <li>• <b>Inputs/outputs objects:</b> allows pre-symbolizing of the input/output objects.</li> <li>• <b>CANopen:</b> allows read/write of SDO.</li> <li>• <b>Defaults:</b> accessible in online mode only.</li> </ul> By clicking on the channel, you display the tabs: <ul style="list-style-type: none"> <li>• <b>PDO:</b> enables you to configure the PDOs.</li> <li>• <b>Configuration:</b> enables you to declare and configure the CANopen master.</li> <li>• <b>Debug:</b> accessible in online mode only.</li> <li>• <b>Error control:</b> accessible in online mode only.</li> </ul> This area also has an LED indicating the channel status.
4	<b>General parameters</b> area	Recalls the function associated with the channel.
5	<b>Parameters in progress</b> area	This area displays the information of an inputs/outputs datum for all the channels. It is divided into 3 columns: <ul style="list-style-type: none"> <li>• the <b>Parameter</b> column displays the inputs/outputs objects and the unmarked objects on which the inputs/outputs datum is mapped,</li> <li>• the <b>Label</b> column shows the name of the inputs/outputs datum,</li> <li>• the <b>Value</b> column shows the value of the inputs/outputs datum.</li> </ul>

**Note:** For standard devices, the values are displayed in the following formats:

- decimal (default),
- hexadecimal,
- binary.

To select the format, right-click on a value in the debug screen, then choose the **display mode**.

For devices with boolean vision (FTB) the value can be forced.

**Note:** In the **Value** column, when a variable appears in red, it shows that it's out of range. The range of the variable can be seen by clicking on it. The range is displayed in the status bar.

---



---

## At a Glance

### Aim of this Chapter

This section introduces the diagnostic means of the CANopen bus.

### What's in this Chapter?

This chapter contains the following topics:

Topic	Page
How to perform a diagnostic	126
Master Diagnostics	127
Slave Diagnostics	128

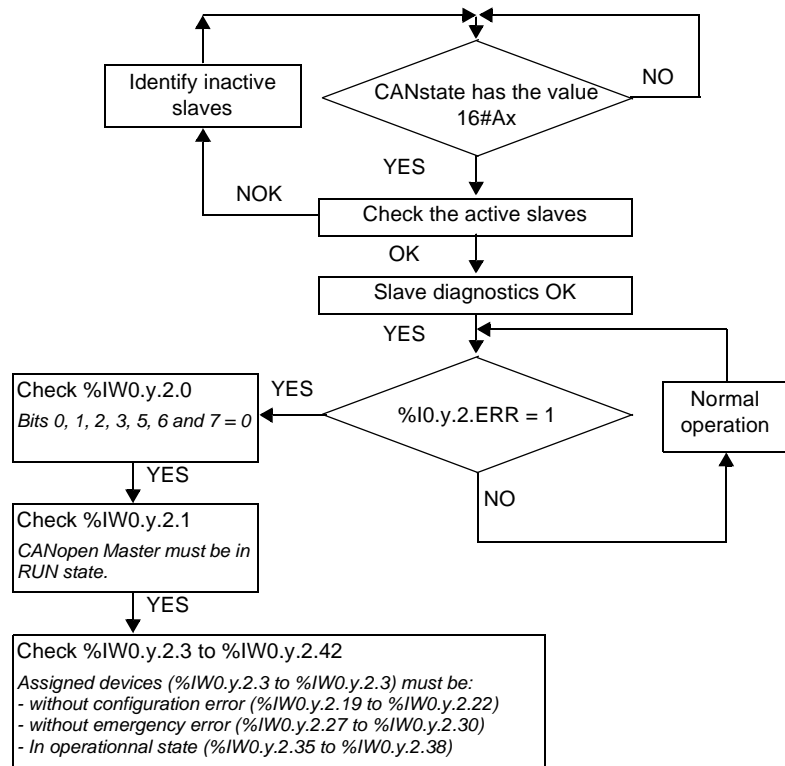
## How to perform a diagnostic

### At a Glance

You can start by using the LEDs located on the forward face of the processor to search for faults on the CANopen bus (see *Visual Diagnostics of CANopen Processors*, p. 33). Next, you can use the procedure (described below) which details bus start up management and the checks to be carried out using the language objects provided by the PLC.

### Procedure

The following diagram indicates the different phases of the procedure:



### How to check %IW0.y.2

To understand the various states of %IW, see *Details of T\_COM\_CO\_BMX Type Implicit Exchange Objects of the IODDT*, p. 144

## Master Diagnostics

### At a Glance

The CANopen bus master can be diagnosed:

- at module level,
- at channel level.

### Module Diagnostics

The Module diagnostics screen displays current errors classed according to their category:

- Internal errors,
- External errors,
- Other errors.

### Channel Diagnostics

The Channel diagnostics screen displays current errors classed according to their category:

- External errors,
- Other errors.

The table below presents the possible errors of a CANopen function:

Error type	Error	Language object
External	The CANopen master is not operational.	%MWr.m.c.2.0
	On or several slaves have errors, or are not operational.	%MWr.m.c.2.1
Other	Configuration error.	%MWr.m.c.2.3
	Overrun of the reception queue low priority.	%IWrr.m.c.0.0
	CAN controller overrun.	%IWrr.m.c.0.1
	CAN controller disconnected from the bus.	%IWrr.m.c.0.2
	CAN controller error.	%IWrr.m.c.0.3
	The CAN controller is no longer in error mode.	%IWrr.m.c.0.4
	Overrun of the transmission queue low priority.	%IWrr.m.c.0.5
	Overrun of the reception queue high priority.	%IWrr.m.c.0.6
	Overrun of the transmission queue high priority.	%IWrr.m.c.0.7
	The task cycle time is greater than the CANopen master cycle time.	%IWrr.m.c.0.8

# Slave Diagnostics

## At a Glance

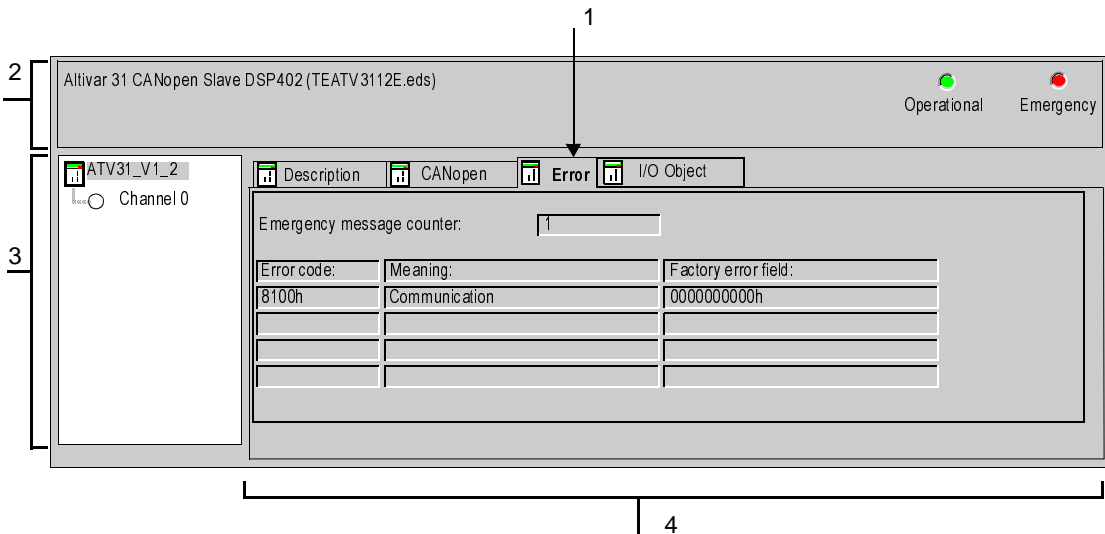
Slave diagnostics are only performed at the device level.

The slave diagnostic screen displays:

- the received emergency messages counter,
- The last four emergency messages (see *Emergency Objects*, p. 159) received in chronological order.

## Illustration

The figure below shows a slave diagnostic screen:





**Elements and Functions**

The table below describes the different areas which make up the master debug screen:

Read	Number	Channel
1	Tab	The tab in the foreground indicates the type of screen displayed. In this case, the diagnostic screen.
2	Module	This area is made up of the abbreviated heading of the module equipped with a CANopen port, as well as 2 LEDs indicating the status of the module.
3	Channel	<p>This area allows you to select the communication channel to be debugged.</p> <p>By clicking on the device, you display the tabs:</p> <ul style="list-style-type: none"><li>● <b>Description</b> : gives the characteristics of the device,</li><li>● <b>CANopen</b>: allows read/write of SDO (online mode only)</li><li>● <b>Faults</b> : allows you to see the last 4 error codes generated by the slave module (tab only accessible in online mode) (see manufacturer's documentation),</li><li>● <b>I/O Objects</b>: allows pre-symbolizing of the input/output objects.</li></ul> <p>This area also has an LED indicating the channel status.</p>
4	Display	<p>This area is composed:</p> <ul style="list-style-type: none"><li>● of error counters,</li><li>● of the last 4 error messages (the last received message is in the upper line).</li></ul>

**Note:** The error counter cannot be reset to 0.



---

# Language Objects

## 9

---

### At a Glance

#### Aim of this Chapter

This chapter describes the implicit and explicit language objects associated with the CANopen master embedded in CPU modules.

**Note:** The system bits %S9 and system words %SW8 and %SW9 are not applicable on CANopen.

**Note:** For information about specific CANopen Master objects, see *Language Object of the CANopen Specific IODDT*, p. 143

#### What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
9.1	Language objects and IODDT for CANopen communication	132
9.2	Language Objects and Generic IODDT Applicable to All Communication Protocols	139
9.3	Language Object of the CANopen Specific IODDT	143
9.4	Emergency objects	159
9.5	The IODDT Type T_GEN_MOD Applicable to All Modules	163

# 9.1                    Language objects and IODDT for CANopen communication

---

## At a Glance

---

**Subject of this Section**                    This chapter describes the language objects and IODDT of CANopen communication.

---

**What's in this Section?**                    This section contains the following topics:

Topic	Page
Introduction to the Language Objects for CANopen Communication	133
Implicit Exchange Language Objects Associated with the Application-Specific Function	134
Explicit Exchange Language Objects Associated with the Application-Specific Function	135
Management of Exchanges and Reports with Explicit Objects	137

---

## Introduction to the Language Objects for CANopen Communication

---

### General

The IODDTs are predefined by the manufacturer and contain inputs/outputs language objects belonging to a channel of a specific application module.

CANopen communication has 1 associated IODDT:

- T\_COM\_STS\_GEN used by all communication protocols,

**Note:** the creation of an IODDT-type variable is performed in two ways:

- I/O object tab,
- Data editor.

---

### Language Object Types

Each IODDT contains a group of language objects which are used to control them and check their operation.

There are two types of language objects:

- implicit exchange objects automatically exchanged at each cycle of the task associated with the module,
- explicit exchange objects exchanged at the request of the application, using explicit exchange instructions.

Implicit exchanges concern the status of the modules, the communication signals, the slaves, etc.

Explicit exchanges allow module parametering and diagnostics.

**Note:** Each slave device has an IODDT (except FTB). For more information, please refer to the user manual of the concerned device.

---

## Implicit Exchange Language Objects Associated with the Application-Specific Function

---

### At a Glance

An integrated application-specific interface or the addition of a module automatically enhances the language objects application used to program this interface or module.

These objects correspond to the input/output images and software data of the module or integrated application-specific interface.

---

### Reminders

The module inputs (%I and %IW) are updated in the PLC memory at the start of the task, the PLC being in RUN or STOP mode.

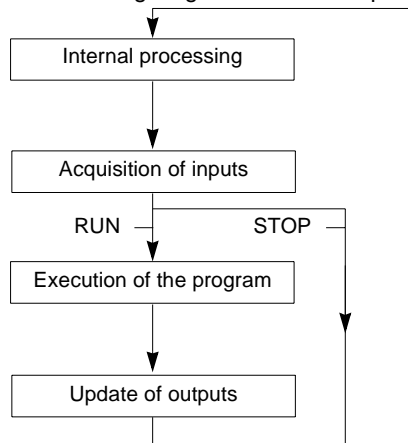
The outputs (%Q and %QW) are updated at the end of the task, only when the PLC is in RUN mode.

**Note:** For BMX P34 processors, when the task occurs in STOP mode, depending on the configuration selected:

- Outputs are set to fallback position (fallback mode),
  - Outputs are maintained at their last value (maintain mode).
- 

### Figure

The following diagram shows the operating cycle of a PLC task (cyclical execution).



# Explicit Exchange Language Objects Associated with the Application-Specific Function

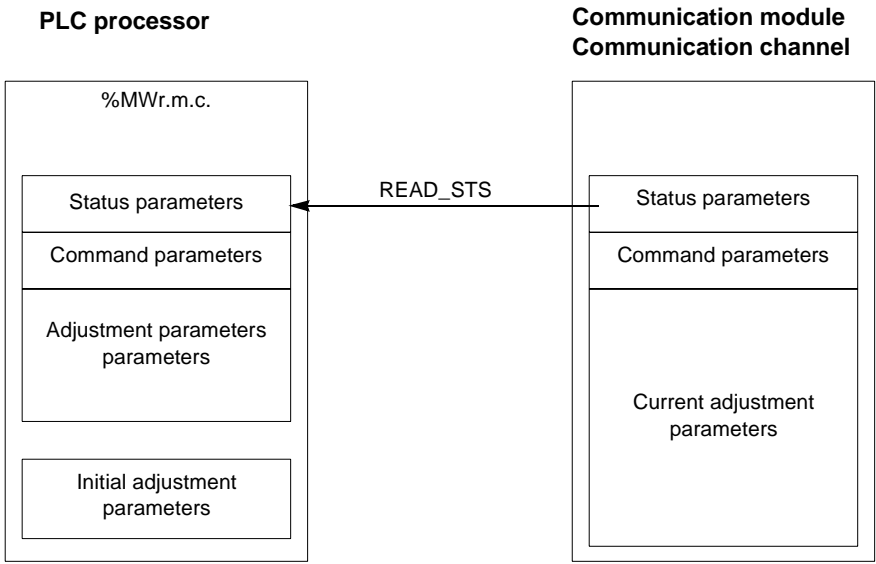
## At a Glance

Explicit exchanges are exchanges performed at the user program's request, and using the `READ_STS` instructions (read of status words).  
These exchanges apply to a set of `%MW` objects of the same type (status) belonging to a channel.

**Note:** These objects provide information about the module (e.g.: type of fault on a channel).

## General Principle for Using Explicit Instructions

The diagram below shows the different types of explicit exchanges that can be made between the processor and module.



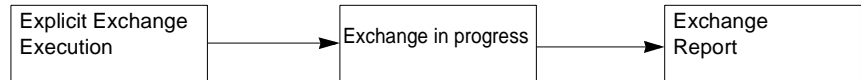
**Managing Exchanges**

During an explicit exchange, it is necessary to check its performance in order that data is only taken into account when the exchange has been correctly executed.

To do this, two types of information is available:

- information concerning the exchange in progress,
- The exchange report.

The following diagram describes the management principle for an exchange



**Note:** In order to avoid several simultaneous explicit exchanges for the same channel, it is necessary to test the value of the word EXCH\_STS (%MWx.m.c.0) of the IODDT associated to the channel before to call any EF using this channel.

---



## Management of Exchanges and Reports with Explicit Objects

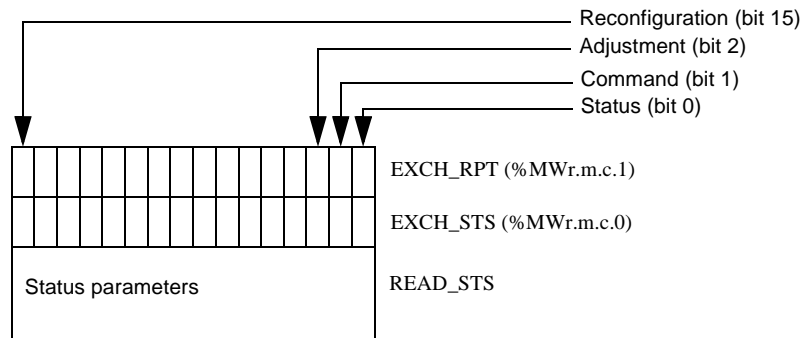
### At a Glance

When data is exchanged between the PLC memory and the module, the module may require several task cycles to acknowledge this information. All IODDTs use two words to manage exchanges:

- EXCH\_STS (%MWr.m.c.0): exchange in progress,
- EXCH\_RPT (%MWr.m.c.1): report.

### Illustration

The illustration below shows the different significant bits for managing exchanges:



### Description of Significant Bits

The rank 0 bits of the words EXCH\_STS (%MWr.m.c.0) and EXCH\_RPT (%MWr.m.c.1) are associated with the status parameters:

- The STS\_IN\_PROGR bit (%MWr.m.c.0.0) indicates whether a read request for the status words is in progress.
- The STS\_ERR bit (%MWr.m.c.1.0) specifies whether a read request for the status words is accepted by the module channel.

**Execution  
Indicators for an  
Explicit  
Exchange:  
EXCH\_STS**

The table below shows the EXCH\_STS (%MWr.m.c.0) explicit exchange control bits:

Standard symbol	Type	Access	Meaning	Address
STS_IN_PROGR	BOOL	R	Reading of channel status words in progress	%MWr.m.c.0.0

**Note:** If the module is not present or is disconnected, explicit exchange objects (READ\_STS, for example) are not sent to the module (STS\_IN\_PROG (%MWr.m.c.0.0) = 0), but the words are refreshed.

---

**Explicit  
Exchange  
Report:  
EXCH\_RPT**

The table below presents the EXCH\_RPT (%MWr.m.c.1) report bits:

Standard symbol	Type	Access	Meaning	Address
STS_ERR	BOOL	R	Error reading channel status words (1 = failure)	%MWr.m.c.1.0

---

## 9.2 Language Objects and Generic IODDT Applicable to All Communication Protocols

---

### At a Glance

---

#### Aim of this Section

This section presents the language objects and generic IODDT applicable to all communication protocols.

#### What's in this Section?

This section contains the following topics:

Topic	Page
Details of IODDT Implicit Exchange Objects of Type T_COM_STS_GEN	140
Details of IODDT Explicit Exchange Objects of Type T_COM_STS_GEN	141

---

## Details of IODDT Implicit Exchange Objects of Type T\_COM\_STS\_GEN

---

### At a Glance

The following table presents the IODDT implicit exchange objects of type T\_COM\_STS\_GEN applicable to all communication protocols except Fipio.

---

### Error Bit

The table below presents the meaning of the error bit CH\_ERROR (%Ir.m.c.ERR).

Standard symbol	Type	Access	Meaning	Address
CH_ERROR	EBOOL	R	Communication channel error bit.	%Ir.m.c.ERR

---

## Details of IODDT Explicit Exchange Objects of Type T\_COM\_STS\_GEN

### At a Glance

This section presents the T\_COM\_STS\_GEN type IODDT explicit exchange objects applicable to all communication protocols except Fipio. It includes the word type objects whose bits have a specific meaning. These objects are presented in detail below.

Sample Variable Declaration: IODDT\_VAR1 of type T\_COM\_STS\_GEN.

### Observations

- In general, the meaning of the bits is given for bit status 1. In specific cases an explanation is given for each status of the bit.
- Not all bits are used.

### Execution Flags of an Explicit Exchange: EXCH\_STS

The table below shows the meaning of channel exchange control bits from channel EXCH\_STS (%MWr.m.c.0).

Standard symbol	Type	Access	Meaning	Address
STS_IN_PROGR	BOOL	R	Reading of channel status words in progress.	%MWr.m.c.0.0
CMD_IN_PROGR	BOOL	R	Current parameter exchange in progress.	%MWr.m.c.0.1
ADJ_IN_PROGR	BOOL	R	Adjustment parameter exchange in progress.	%MWr.m.c.0.2

### Explicit Exchange Report: EXCH\_RPT

The table below presents the meaning of the exchange report bits EXCH\_RPT (%MWr.m.c.1).

Standard symbol	Type	Access	Meaning	Address
STS_ERR	BOOL	R	Reading error for channel status words.	%MWr.m.c.1.0
CMD_ERR	BOOL	R	Error during command parameter exchange.	%MWr.m.c.1.1
ADJ_ERR	BOOL	R	Error during adjustment parameter exchange.	%MWr.m.c.1.2

### Standard Channel Faults, CH\_FLT

The table below shows the meaning of the bits of the status word CH\_FLT (%MWr.m.c.2). Reading is performed by a READ\_STS (IODDT\_VAR1).

Standard symbol	Type	Access	Meaning	Address
NO_DEVICE	BOOL	R	No device is working on the channel.	%MWr.m.c.2.0
1_DEVICE_FLT	BOOL	R	A device on the channel is faulty.	%MWr.m.c.2.1

Standard symbol	Type	Access	Meaning	Address
BLK	BOOL	R	Terminal block fault (not connected).	%MWr.m.c.2.2
TO_ERR	BOOL	R	Time out error (defective wiring).	%MWr.m.c.2.3
INTERNAL_FLT	BOOL	R	Internal error or channel self-testing.	%MWr.m.c.2.4
CONF_FLT	BOOL	R	Different hardware and software configurations.	%MWr.m.c.2.5
COM_FLT	BOOL	R	Problem communicating with the PLC.	%MWr.m.c.2.6
APPLI_FLT	BOOL	R	Application error (adjustment or configuration error).	%MWr.m.c.2.7

---

---

# 9.3                      Language Object of the CANopen Specific IODDT

---

## At a Glance

---

**Subject of this Section**                      This section describes the implicit and explicit language objects of the CANopen specific IODDT, T\_COM\_CO\_BMX.

---

**What's in this Section?**                      This section contains the following topics:

Topic	Page
Details of T_COM_CO_BMX Type Implicit Exchange Objects of the IODDT	144
Details of T_COM_CO_BMX Type Explicit Exchange Objects of the IODDT	156
Language Objects Associated with Configuration	158

---

## Details of T\_COM\_CO\_BMX Type Implicit Exchange Objects of the IODDT

---

### At a Glance

Implicit exchange objects are automatically exchanged at each cycle of a task associated with the channel. There objects are %I, %IW, %Q and %QW.

The table below presents the various implicit exchange objects of IODDT T\_COM\_CO\_BMX.

The parameters r,m and c shown in the following-tables represent the topologic addressing of the module. Each parameter had the following signification:

- **r** represents the rack number,
- **m** represents the module number,
- **c** represents the channel number.

---

### Channel Error

The table below presents the bit %Ir.m.c.ERR:

Standard symbol	Type	Access	Description	Address
CH_ERROR	BOOL	R	Channel error	%Ir.m.c.ERR

---

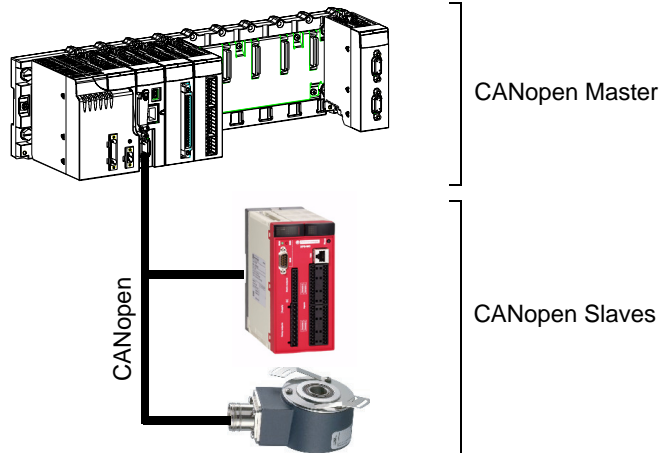


## Master Status and Event Indicator

The table below shows the words %IW<sub>r</sub>.m.c.0 to %IW<sub>r</sub>.m.c.2:

Standard symbol	Type	Access	Description	Address
COMM_STS	INT	R	Communication status of master	%IW <sub>r</sub> .m.c.0
CAN_STS	INT	R	Status of CANopen Master	%IW <sub>r</sub> .m.c.1
EVT_STS	INT	R	Event indicator	%IW <sub>r</sub> .m.c.2

The following figure gives an example of Master status indicator.



In this example, the word %IW<sub>0</sub>.0.2.1 gives the status of the CANopen Master. The parameters are as follows:

- **r:** '0',
- **m:** '0',
- **c:** '2' (CANopen channel).

The last parameter ('1') indicates the used word (CAN\_STS).

The table below shows the meaning of bits from various status words from the master and event indicators:

Addresses	Description	Bit meaning
%IWrr.m.c.0	Communication status of master	<p><b>Bit 0=1:</b> Overflow of the reception queue low priority. The CANopen master is receiving "Heartbeat" and "Node guarding" messages as well as SSDOs and CSDOs via the low priority queue.</p> <p><b>Bit 1=1:</b> FIFO overwrite of CAN controller</p> <p><b>Bit 2=1:</b> The CAN controller has status "BUS Off".</p> <p><b>Bit 3=1:</b> CAN controller fault. Bit reset to 0 when the fault disappears.</p> <p><b>Bit 4=1:</b> The CAN controller has left fault state.</p> <p><b>Bit 5=1:</b> Overflow of the emission queue low priority. The CANopen master is transmitting "Heartbeat" and "Node guarding" messages as well as SSDOs and CSDOs via the low priority transmission queue.</p> <p><b>Bit 6=1:</b> Overflow of the reception queue high priority. The CANopen master is receiving RPDOs, NMT commands, the message Sync and emergency messages via the high priority reception queue.</p> <p><b>Bit 7=1:</b> Overflow of the reception queue high priority. The CANopen master is sending TPDOs, NMT commands, the message Sync and emergency messages via the high priority queue.</p> <p><b>Bit 8=1:</b> Indicates the task cycle is faster than the CANopen master cycle (outputs can be overwritten). To avoid overwrite, you are advised to have a longer task cycle time than the CANopen cycle. The cycle values are available in the words %IWrr.m.c.59 à %IWrr.m.c.61.</p>

Addresses	Description	Bit meaning
%IWrm.c.1	Status of CANopen Master	<p><b>0x00: INIT:</b> The CANopen master is not initialized. This corresponds to the "INITIALISATION" status of the CANopen module. In this state, the CANopen master cannot communicate with the network.</p> <p><b>0x40: RESET :</b> The CANopen master is configured as master during "NMT startup". The object dictionary of CANopen master can be configured by SDOs via the CAN bus and the interface of the SDO command. The application has access rights to read/write to the object dictionary via the SDO command. The initialization of network manager has not yet started.</p> <p><b>=0x60: NET –INIT:</b> Starting according to CIA DSP-302. The CANopen master is checking the allocation of slaves.</p> <p><b>=0x61: NET RESET :</b> The network is reinitialized by the NMT command "Reset communication all nodes"</p> <p><b>=0x62: NET –WAIT:</b> The CANopen master is waiting for the modules to be able to run the command "Reset communication".</p> <p><b>0x64: BOOT –CONF:</b> The CANopen master is running the individual initialization of modules according to CIA DSP-302.</p> <p><b>0x8x: CLEAR :</b> The network is scanned. The master is waiting for a start command ("Start CANopen Master/Manager" or "Start network").</p> <p><b>0xAx: RUN</b> The network is in "Operational" state.</p> <p><b>0xCx: STOP</b> The network is in "Stop" state.</p> <p><b>0Ex: PREOPERATIONAL :</b> The network is in "Pre-operational" status.</p> <p><b>0x9x: FATAL ERROR :</b> A fatal error has occurred. The CANopen master must be reinitialized.</p> <p>The network is scanned. The 4 heavy bits of the status variable indicate the state of the network (CLEAR, RUN, STOP, PREOPERATIONAL). The 4 light bits contain additional information:</p> <p>Bit 0: Error bit for optional modules.</p> <ul style="list-style-type: none"> <li>● =0 : No error.</li> <li>● =1 : At least one of the optional modules doesn't correspond to the configuration of the expected network.</li> </ul> <p>Bit 1: Error bit for obligatory modules.</p> <ul style="list-style-type: none"> <li>● =0 : No error.</li> <li>● =1 : At least one of the obligatory modules is not in the expected status.</li> </ul> <p>Bit 2: Bit "Operational"</p> <ul style="list-style-type: none"> <li>● =0 : No module including the CANopen Master is in CANopen "Operational" status</li> <li>● =1 : At least one of the modules is in "Operational" status (excluding the CANopen Master)</li> </ul> <p>Bit 3: "Operational" bit of CANopen Master</p> <ul style="list-style-type: none"> <li>● =0 : The CANopen master is not in "Operational" state</li> <li>● =1 : The CANopen Master is in "Operational" status.</li> </ul>

Addresses	Description	Bit meaning
%IWrr.m.c.2	Event indicator	<p><b>Bit 0=1</b> This bit was still at 1 when a communication error occurred with the network. The communication status of CANopen Master gives the exact reason. (The CANopen master is a fatal error).</p> <p><b>Bit 1=1</b> A module is using the node number of CANopen Master. (The CANopen master is a fatal error.)</p> <p><b>Bit2=1</b> : An obligatory module is faulty with network monitoring (Heartbeat or Nodeguarding). The consequences of this fault depend on the configuration of the "NMT Startup" object. This bit is significant if the "NMT Startup" object does not generate a bus reset.</p> <p><b>Bit 3=1</b> Identity error or error from the object dictionary DCF of an obligatory object. (The CANopen Master is in fatal error status.)</p> <p><b>Bit4=1</b> : Identity error of an optional module. The concerned module is in "Stop" state.</p> <p><b>Bit 5=1</b>: Failure during the creation of the process image and the configuration of PDOs during the self-configuration phase. (The CANopen Master is in fatal error status.)</p> <p><b>Bit6=1</b> : Network monitor fault during the auto-configuration phase. Detection of a late device connection.</p> <p><b>Bit7=1</b> : This bit is still at 1 if a bit in the list of bits changes status.</p> <p><b>Bit 8=1</b>: At the beginning of the starting procedure, the CANopen master checks every slave. This bit is set at 1 if the Master doesn't support certain functions of the device (example: bits 4 to 6 of the object 1F81h).</p> <p><b>Bit9=1</b> : The CANopen Master has received an RPDO with too little data. (The CANopen master has a fatal error.)</p> <p><b>Bit10=1</b> : Signals a fault during the configuration of a device. For example: object is not supported by the device. (The CANopen master has a fatal error.)</p> <p><b>Bit11=1</b> : This bit indicates an overflow of the queue for application specific for the SDO interface.</p> <p><b>Bit12=1</b> : The master cycle time is greater than 256 ms.</p> <p><b>Bit13=1</b> : Reserved</p> <p><b>Bit14=1</b> : Reserved.</p> <p><b>Bit15=1</b> : The Master is alone on the bus (Check that the cable is connected).</p>

**Assigned Slaves**    The table below shows the words %IWr.m.c.3 to %IWr.m.c.6:

Standard symbol	Type	Access	Description	Address
SLAVE_ASSIGNED_1_16	INT	R	For assigned slaves from 1 to 16	%IWr.m.c.3
SLAVE_ASSIGNED_17_32	INT	R	For assigned slaves from 17 to 32	%IWr.m.c.4
SLAVE_ASSIGNED_33_48	INT	R	For assigned slaves from 33 to 48	%IWr.m.c.5
SLAVE_ASSIGNED_49_64	INT	R	For assigned slaves from 49 to 63	%IWr.m.c.6

If the bit is equal to 0, no slave is assigned to this bit.

If the bit is equal to 1, a slave is assigned to this bit.

The node number corresponds to the number of the bit + 1.

---

**Slaves  
Configured**

The table below shows the words %IWr.m.c.11 to %IWr.m.c.14:

Standard symbol	Type	Access	Description	Address
SLAVE_CONF_1_16	INT	R	For configured slaves from 1 to 16	%IWr.m.c.11
SLAVE_CONF_17_32	INT	R	For configured slaves from 17 to 32	%IWr.m.c.12
SLAVE_CONF_33_48	INT	R	For configured slaves from 33 to 48	%IWr.m.c.13
SLAVE_CONF_49_64	INT	R	For configured slaves from 49 to 63	%IWr.m.c.14

If the bit is equal to 0, the slave is not configured and cannot start.

If the bit is equal to 1, the slave is configured and can be started.

The node number corresponds to the number of the bit + 1.

---

**Slaves with  
Configuration  
Faults**

The table below shows the words %IWr.m.c.19 to %IWr.m.c.22:

Standard symbol	Type	Access	Description	Address
SLAVE_FLT_1_16	INT	R	Slaves with configuration faults from 1 to 16	%IWr.m.c.19
SLAVE_FLT_17_32	INT	R	Slaves with configuration faults from 17 to 32	%IWr.m.c.20
SLAVE_FLT_33_48	INT	R	Slaves with configuration faults from 33 to 48	%IWr.m.c.21
SLAVE_FLT_49_64	INT	R	Slaves with configuration faults from 49 to 63	%IWr.m.c.22

If the bit is equal to 0, the assigned slave corresponds to the configuration.

If the bit is equal to 1, the assigned slave does not correspond to the configuration.

The node number corresponds to the number of the bit + 1.

---

**Faulty Slaves**

The table below shows the words %IW<sub>r</sub>.m.c.27 to %IW<sub>r</sub>.m.c.30:

Standard symbol	Type	Access	Description	Address
SLAVE_EMCY_1_16	INT	R	Faulty slaves from 1 to 16	%IW <sub>r</sub> .m.c.27
SLAVE_EMCY_17_32	INT	R	Faulty slaves from 17 to 32	%IW <sub>r</sub> .m.c.28
SLAVE_EMCY_33_48	INT	R	Faulty slaves from 33 to 48	%IW <sub>r</sub> .m.c.29
SLAVE_EMCY_49_64	INT	R	Faulty slaves from 49 to 63	%IW <sub>r</sub> .m.c.30

If the bit is equal to 0, the slave is error free.

If the bit is equal to 1, the slave has an error.

The node number corresponds to the number of the bit + 1.

**Operational  
Slaves from 1 to  
16**

The table below presents the word %IW<sub>r</sub>.m.c.35:

Standard symbol	Type	Access	Description	Address
SLAVE_ACTIV_1	BOOL	R	Slave operational on the bus: device 1	%IW <sub>r</sub> .m.c.35.0
SLAVE_ACTIV_2	BOOL	R	Slave operational on the bus: device 2	%IW <sub>r</sub> .m.c.35.1
SLAVE_ACTIV_3	BOOL	R	Slave operational on the bus: device 3	%IW <sub>r</sub> .m.c.35.2
SLAVE_ACTIV_4	BOOL	R	Slave operational on the bus: device 4	%IW <sub>r</sub> .m.c.35.3
SLAVE_ACTIV_5	BOOL	R	Slave operational on the bus: device 5	%IW <sub>r</sub> .m.c.35.4
SLAVE_ACTIV_6	BOOL	R	Slave operational on the bus: device 6	%IW <sub>r</sub> .m.c.35.5
SLAVE_ACTIV_7	BOOL	R	Slave operational on the bus: device 7	%IW <sub>r</sub> .m.c.35.6
SLAVE_ACTIV_8	BOOL	R	Slave operational on the bus: device 8	%IW <sub>r</sub> .m.c.35.7
SLAVE_ACTIV_9	BOOL	R	Slave operational on the bus: device 9	%IW <sub>r</sub> .m.c.35.8
SLAVE_ACTIV_10	BOOL	R	Slave operational on the bus: device 10	%IW <sub>r</sub> .m.c.35.9
SLAVE_ACTIV_11	BOOL	R	Slave operational on the bus: device 11	%IW <sub>r</sub> .m.c.35.10
SLAVE_ACTIV_12	BOOL	R	Slave operational on the bus: device 12	%IW <sub>r</sub> .m.c.35.11
SLAVE_ACTIV_13	BOOL	R	Slave operational on the bus: device 13	%IW <sub>r</sub> .m.c.35.12
SLAVE_ACTIV_14	BOOL	R	Slave operational on the bus: device 14	%IW <sub>r</sub> .m.c.35.13
SLAVE_ACTIV_15	BOOL	R	Slave operational on the bus: device 15	%IW <sub>r</sub> .m.c.35.14
SLAVE_ACTIV_16	BOOL	R	Slave operational on the bus: device 16	%IW <sub>r</sub> .m.c.35.15

The node number corresponds to the number of the bit + 1.

**Operational  
Slaves from 17 to  
32**

The table below presents the word %IW<sub>r</sub>.m.c.36:

Standard symbol	Type	Access	Description	Address
SLAVE_ACTIV_17	BOOL	R	Slave operational on the bus: device 17	%IW <sub>r</sub> .m.c.36.0
SLAVE_ACTIV_18	BOOL	R	Slave operational on the bus: device 18	%IW <sub>r</sub> .m.c.36.1
SLAVE_ACTIV_19	BOOL	R	Slave operational on the bus: device 19	%IW <sub>r</sub> .m.c.36.2
SLAVE_ACTIV_20	BOOL	R	Slave operational on the bus: device 20	%IW <sub>r</sub> .m.c.36.3
SLAVE_ACTIV_21	BOOL	R	Slave operational on the bus: device 21	%IW <sub>r</sub> .m.c.36.4
SLAVE_ACTIV_22	BOOL	R	Slave operational on the bus: device 22	%IW <sub>r</sub> .m.c.36.5
SLAVE_ACTIV_23	BOOL	R	Slave operational on the bus: device 23	%IW <sub>r</sub> .m.c.36.6
SLAVE_ACTIV_24	BOOL	R	Slave operational on the bus: device 24	%IW <sub>r</sub> .m.c.36.7
SLAVE_ACTIV_25	BOOL	R	Slave operational on the bus: device 25	%IW <sub>r</sub> .m.c.36.8
SLAVE_ACTIV_26	BOOL	R	Slave operational on the bus: device 26	%IW <sub>r</sub> .m.c.36.9
SLAVE_ACTIV_27	BOOL	R	Slave operational on the bus: device 27	%IW <sub>r</sub> .m.c.36.10
SLAVE_ACTIV_28	BOOL	R	Slave operational on the bus: device 28	%IW <sub>r</sub> .m.c.36.11
SLAVE_ACTIV_29	BOOL	R	Slave operational on the bus: device 29	%IW <sub>r</sub> .m.c.36.12
SLAVE_ACTIV_30	BOOL	R	Slave operational on the bus: device 30	%IW <sub>r</sub> .m.c.36.13
SLAVE_ACTIV_31	BOOL	R	Slave operational on the bus: device 31	%IW <sub>r</sub> .m.c.36.14
SLAVE_ACTIV_32	BOOL	R	Slave operational on the bus: device 32	%IW <sub>r</sub> .m.c.36.15

---



### Operational Slaves from 33 to 48

The table below shows the word %IW<sub>r</sub>.m.c.37:

Standard symbol	Type	Access	Description	Address
SLAVE_ACTIV_33	BOOL	R	Slave operational on the bus: device 33	%IW <sub>r</sub> .m.c.37.0
SLAVE_ACTIV_34	BOOL	R	Slave operational on the bus: device 34	%IW <sub>r</sub> .m.c.37.1
SLAVE_ACTIV_35	BOOL	R	Slave operational on the bus: device 35	%IW <sub>r</sub> .m.c.37.2
SLAVE_ACTIV_36	BOOL	R	Slave operational on the bus: device 36	%IW <sub>r</sub> .m.c.37.3
SLAVE_ACTIV_37	BOOL	R	Slave operational on the bus: device 37	%IW <sub>r</sub> .m.c.37.4
SLAVE_ACTIV_38	BOOL	R	Slave operational on the bus: device 38	%IW <sub>r</sub> .m.c.37.5
SLAVE_ACTIV_39	BOOL	R	Slave operational on the bus: device 39	%IW <sub>r</sub> .m.c.37.6
SLAVE_ACTIV_40	BOOL	R	Slave operational on the bus: device 40	%IW <sub>r</sub> .m.c.37.7
SLAVE_ACTIV_41	BOOL	R	Slave operational on the bus: device 41	%IW <sub>r</sub> .m.c.37.8
SLAVE_ACTIV_42	BOOL	R	Slave operational on the bus: device 42	%IW <sub>r</sub> .m.c.37.9
SLAVE_ACTIV_43	BOOL	R	Slave operational on the bus: device 43	%IW <sub>r</sub> .m.c.37.10
SLAVE_ACTIV_44	BOOL	R	Slave operational on the bus: device 44	%IW <sub>r</sub> .m.c.37.11
SLAVE_ACTIV_45	BOOL	R	Slave operational on the bus: device 45	%IW <sub>r</sub> .m.c.37.12
SLAVE_ACTIV_46	BOOL	R	Slave operational on the bus: device 46	%IW <sub>r</sub> .m.c.37.13
SLAVE_ACTIV_47	BOOL	R	Slave operational on the bus: device 47	%IW <sub>r</sub> .m.c.37.14
SLAVE_ACTIV_48	BOOL	R	Slave operational on the bus: device 48	%IW <sub>r</sub> .m.c.37.15

**Operational  
Slaves from 49 to  
64**

The table below shows the word %IW<sub>r</sub>.m.c.38:

Standard symbol	Type	Access	Description	Address
SLAVE_ACTIV_49	BOOL	R	Slave operational on the bus: device 49	%IW <sub>r</sub> .m.c.38.0
SLAVE_ACTIV_50	BOOL	R	Slave operational on the bus: device 50	%IW <sub>r</sub> .m.c.38.1
SLAVE_ACTIV_51	BOOL	R	Slave operational on the bus: device 51	%IW <sub>r</sub> .m.c.38.2
SLAVE_ACTIV_52	BOOL	R	Slave operational on the bus: device 52	%IW <sub>r</sub> .m.c.38.3
SLAVE_ACTIV_53	BOOL	R	Slave operational on the bus: device 53	%IW <sub>r</sub> .m.c.38.4
SLAVE_ACTIV_54	BOOL	R	Slave operational on the bus: device 54	%IW <sub>r</sub> .m.c.38.5
SLAVE_ACTIV_55	BOOL	R	Slave operational on the bus: device 55	%IW <sub>r</sub> .m.c.38.6
SLAVE_ACTIV_56	BOOL	R	Slave operational on the bus: device 56	%IW <sub>r</sub> .m.c.38.7
SLAVE_ACTIV_57	BOOL	R	Slave operational on the bus: device 57	%IW <sub>r</sub> .m.c.38.8
SLAVE_ACTIV_58	BOOL	R	Slave operational on the bus: device 58	%IW <sub>r</sub> .m.c.38.9
SLAVE_ACTIV_59	BOOL	R	Slave operational on the bus: device 59	%IW <sub>r</sub> .m.c.38.10
SLAVE_ACTIV_60	BOOL	R	Slave operational on the bus: device 60	%IW <sub>r</sub> .m.c.38.11
SLAVE_ACTIV_61	BOOL	R	Slave operational on the bus: device 61	%IW <sub>r</sub> .m.c.38.12
SLAVE_ACTIV_62	BOOL	R	Slave operational on the bus: device 62	%IW <sub>r</sub> .m.c.38.13
SLAVE_ACTIV_63	BOOL	R	Slave operational on the bus: device 63	%IW <sub>r</sub> .m.c.38.14

---

**Slave in Stop  
State**

The table below shows the words %IW<sub>r</sub>.m.c.43 to %IW<sub>r</sub>.m.c.46:

Standard symbol	Type	Access	Description	Address
SLAVE_STOPPED_1_16	INT	R	Stopped slaves from 1 to 16	%IW <sub>r</sub> .m.c.43
SLAVE_STOPPED_17_32	INT	R	Stopped slaves from 17 to 32	%IW <sub>r</sub> .m.c.44
SLAVE_STOPPED_33_48	INT	R	Stopped slaves from 33 to 48	%IW <sub>r</sub> .m.c.45
SLAVE_STOPPED_49_64	INT	R	Stopped slaves from 49 to 63	%IW <sub>r</sub> .m.c.46

---

**Pre-Operational Slaves**

The table below shows the words %IW<sub>r</sub>.m.c.51 to %IW<sub>r</sub>.m.c.54:

Standard symbol	Type	Access	Description	Address
SLAVE_PREOP_1_16	INT	R	Pre-operational slaves from 1 to 16.	%IW <sub>r</sub> .m.c.51
SLAVE_PREOP_17_32	INT	R	Pre-operational slaves from 17 to 32.	%IW <sub>r</sub> .m.c.52
SLAVE_PREOP_33_48	INT	R	Pre-operational slaves from 33 to 48.	%IW <sub>r</sub> .m.c.53
SLAVE_PREOP_49_64	INT	R	Pre-operational slaves from 49 to 63.	%IW <sub>r</sub> .m.c.54

**Master Cycle Time**

The table below shows the meaning of status words relative to the time cycle of the master:

Addresses	Description	Meaning
%IW <sub>r</sub> .m.c.59	Minimum master cycle time	Minimum value of the CANopen master cycle time in ms.
%IW <sub>r</sub> .m.c.60	Current master cycle time	Current value of the CANopen master cycle time in ms.
%IW <sub>r</sub> .m.c.61	Maximum master cycle time	Maximum value of the CANopen master cycle time in ms.

**Reset Emergency Default**

The table below shows the meaning of the command word of the CANopen master:

Addresses	Description	Bit meaning
%QW <sub>r</sub> .m.c.0	Command word of the CANopen master	<p><b>Bit 0=1:</b> Reset emergency slaves bitlist. This bit is set to zero after the reset of the bitlist.</p> <p><b>Bit 1:</b> Reset bit 8 of the communication status of the master (%IW0.0.2.0). The bit 8 indicates that the task cycle is faster than the CANopen master cycle. The bit 1 is set to zero after the reset of the bit 8.</p> <p><b>Bit 2 to bit 15:</b> Reserved.</p>

## Details of T\_COM\_CO\_BMX Type Explicit Exchange Objects of the IODDT

---

### At a Glance

This part shows the explicit exchange language objects for the CANopen master. These objects are exchanged on the application's request, using the instruction `READ_STS`.

The parameters `r`, `m` and `c` shown in the following tables represent the topological addressing of the module. Each parameter has the following signification:

- **r**: represents the rack number,
- **m**: represents the position of the module on the rack,
- **c**: represents the channel number.

---

### Execution Indicator: EXCH\_STS

The table below shows the meanings of channel exchange control bits from channel `EXCH_STS (%MWr.m.c.0)`:

Symbol	Type	Access	Description	Number
STS_IN_PROGR	BOOL	R	Status parameter read in progress	%MW <sub>r.m.c.0.0</sub>

---

### Exchange Report: EXCH\_RPT

The table below presents the meaning of the run report bits of the channel `EXCH_RPT (%MWr.m.c.1)`:

Symbol	Type	Access	Description	Number
STS_ERR	BOOL	R	Error while reading channel status	%MW <sub>r.m.c.1.0</sub>

**Standard** The following table explains the meaning of the CH\_FLT (%MWr.m.c.2) status word bits. Reading is performed by a READ\_STS:  
**Channel Faults:**  
**CH\_FLT**

Object	Function	Meaning
%MWr.m.c.2	Status of the CANopen Master	<b>Bit 0=1:</b> The CANopen Master is not in operationnal state. <b>Bit 1=1:</b> Slave has an error, one or more slaves have errors or are not in operationnal state. <b>Bit 2:</b> Reserved. <b>Bit 3=1:</b> Configuration error. <b>Bit 4 to bit 7:</b> Reserved. <b>Bit 8 to Bit 10:</b> CAN ERR led: <ul style="list-style-type: none"> <li>● 000 = off,</li> <li>● 001 = single flash,</li> <li>● 010 = double flash,</li> <li>● 011 = triple flash,</li> <li>● 111 = on.</li> </ul> <b>Bit 11 to Bit 13:</b> CAN RUN led: <ul style="list-style-type: none"> <li>● 001 = single flash,</li> <li>● 100 = blinking,</li> <li>● 111 = on.</li> </ul> <b>Bit 14 to Bit 15:</b> Reserved.
%MWr.m.c.3	Generic error count	Number of received emergency messages with code 10xxH
%MWr.m.c.4	Device hardware error count	Number of received emergency messages with code 50xxH
%MWr.m.c.5	Device software error count	Number of received emergency messages with code 60xxH
%MWr.m.c.6	Communication error count	Number of received emergency messages with code 81xxH
%MWr.m.c.7	Protocol error count	Number of received emergency messages with code 82xxH
%MWr.m.c.8	External error count	Number of received emergency messages with code 90xxH
%MWr.m.c.9	Device-specific	Number of received emergency messages with code FFxxH

## Language Objects Associated with Configuration

---

### At a Glance

The configuration of a CANopen master is stored in the configuration constants (%KW).

The parameters r,m and c shown in the following tables represent the topologic addressing of the module. Each parameter has the following signification:

- **r**: represents the rack number,
  - **m**: represents the position of the module on the rack,
  - **c**: represents the channel number.
- 

### Configuration Objects

The following table lists all process control language objects associated configuration of CANopen network:

Number	Type	Function	Description
%KWr.m.c.0	INT	Constant value used by the system	Least significant byte: 16#00: <ul style="list-style-type: none"><li>● 0: reset,</li><li>● 1: maintain.</li></ul> Most significant byte: 16#37.
%KWr.m.c.1	INT	Baud rate (see <i>Length Limitations of the CANopen Network</i> , p. 23)	Values are encoded : <ul style="list-style-type: none"><li>● 0 = 1000 Kbaud,</li><li>● 2 = 500 Kbaud,</li><li>● 3 = 250 Kbaud,</li><li>● 4 = 125 Kbaud,</li><li>● 5 = 50 Kbaud,</li><li>● 6 = 20 Kbaud.</li></ul>
%KWr.m.c.2	INT	COB-ID Synchronization	Default value: 0080h.
%KWr.m.c.3	INT	Synchronization period	1 .. 1000 ms.
%KWr.m.c.4	INT	Configuration bits	Size of input image zone TOR in the memory (in number of bits).
%KWr.m.c.5	INT	Configuration bits	Size of output image zone TOR in the memory (in number of bits).
%KWr.m.c.6	INT	Configuration bits	Address of the start of the input image zone TOR(%M).
%KWr.m.c.7	INT	Configuration bits	Address of the start of the output image zone TOR (%M).
%KWr.m.c.8	INT	Configuration bits	Size of input image zone in the memory (in number of words).
%KWr.m.c.9	INT	Configuration bits	Size of output image zone in the memory (in number of words).
%KWr.m.c.10	INT	Configuration bits	Address of the start of the input image zone (%MW).
%KWr.m.c.11	INT	Configuration bits	Address of the start of the input image zone (%MW).

---

## 9.4 Emergency objects

### Emergency Objects

#### At a Glance

Emergency objects (EMCY) have been defined for CANopen for diagnostic applications.

The COB-ID of these objects contain the identity of the node of the device which produced the emergency message. The COB-ID of emergency objects are constructed in the following manner:

$$\text{COB-ID}_{\text{EMCY}} = 0x80 + \text{node identity}$$

The data field of an EMCY object is composed of 8 bytes containing:

- Emergency error code (2 bytes),
- the error register (1 byte),
- The factory-specific error information (5 bytes).

The following illustration shows the structure of an EMCY object:

COB-ID	Error code		Register error	Error Information manufacturer specific				
0x80+node-ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

**Note:** the contents of the error code and error register are specified by CiA.

With the `Error` tab (see *Slave Diagnostics*, p. 128), you can consult the 4 last emergency messages received, in chronological order.

#### Error Code 00xx

The following table describes the content of error code 00xx:

Error code (hex)	Description
00xx	Error reset to zero or no error

#### Error Code 10xx

The following table describes the content of error code 10xx:

Error code (hex)	Description
10xx	Generic error

**Error Code 2xxx**      The following table describes the content of error code 2xxx:

Error code (hex)	Description
20xx	Current
21xx	Current, input side of the device
22xx	Internal current to the device
23xx	Current, output side of the device

---

**Error Code 3xxx**      The following table describes the content of error code 3xxx:

Error code (hex)	Description
30xx	Voltage
31xx	Principal voltage
32xx	Internal voltage to the device
33xx	Output voltage

---

**Error Code 4xxx**      The following table describes the content of error code 4xxx:

Error code (hex)	Description
40xx	Temperature
41xx	Ambient temperature
42xx	Device temperature

---

**Error Code 50xx**      The following table describes the content of error code 50xx:

Error code (hex)	Description
50xx	Device hardware

---

**Error Code 6xxx**      The following table describes the content of error code 6xxx:

Error code (hex)	Description
60xx	Device software
61xx	Internal software
62xx	User software
63xx	Data set

---



**Error Code 70xx** The following table describes the content of error code 70xx:

Error code (hex)	Description
70xx	Additional modules

**Error Code 8xxx** The following table describes the content of error code 8xxx:

Error code (hex)	Description
80xx	Monitoring
81xx	Communication
8110	CAN overflow (objects lost)
8120	CAN in passive error mode
8130	Life Guard error or Heartbeat error
8140	Recovered from bus
8150	Collision during COB-ID transmission
82xx	Protocol error
8210	PDO not processed due to length error
8220	PDO length exceeded

**Error Code 90xx** The following table describes the content of error code 90xx:

Error code (hex)	Description
90xx	External error

**Error Code Fxxx** The following table describes the content of error code Fxxx:

Error code (hex)	Description
F0xx	Additional functions
FFxx	Specific to the device



## 9.5 The IODDT Type T\_GEN\_MOD Applicable to All Modules

### Details of the Language Objects of the IODDT of Type T\_GEN\_MOD

**At a Glance** All the modules of Modicon M340 PLCs have an associated IODDT of type T\_GEN\_MOD.

**Observations** In general, the meaning of the bits is given for bit status 1. In specific cases an explanation is given for each status of the bit.  
Some bits are not used.

**List of Objects** The table below presents the objects of the IODDT.

Standard symbol	Type	Access	Meaning	Address
MOD_ERROR	BOOL	R	Module error bit	%I.r.m.MOD.ERR
EXCH_STS	INT	R	Module exchange control word	%MWr.m.MOD.0
STS_IN_PROGR	BOOL	R	Reading of status words of the module in progress	%MWr.m.MOD.0.0
EXCH_RPT	INT	R	Exchange report word	%MWr.m.MOD.1
STS_ERR	BOOL	R	Fault when reading module status words	%MWr.m.MOD.1.0
MOD_FLT	INT	R	Internal error word of the module	%MWr.m.MOD.2
MOD_FAIL	BOOL	R	Internal error, module failure	%MWr.m.MOD.2.0
CH_FLT	BOOL	R	Faulty channel(s)	%MWr.m.MOD.2.1
BLK	BOOL	R	Terminal block fault	%MWr.m.MOD.2.2
CONF_FLT	BOOL	R	Hardware or software configuration fault	%MWr.m.MOD.2.5
NO_MOD	BOOL	R	Module missing or inoperative	%MWr.m.MOD.2.6
EXT_MOD_FLT	BOOL	R	Internal error word of the module (Fipio extension only)	%MWr.m.MOD.2.7
MOD_FAIL_EXT	BOOL	R	Internal fault, module unserviceable (Fipio extension only)	%MWr.m.MOD.2.8
CH_FLT_EXT	BOOL	R	Faulty channel(s) (Fipio extension only)	%MWr.m.MOD.2.9
BLK_EXT	BOOL	R	Terminal block fault (Fipio extension only)	%MWr.m.MOD.2.10
CONF_FLT_EXT	BOOL	R	Hardware or software configuration fault (Fipio extension only)	%MWr.m.MOD.2.13
NO_MOD_EXT	BOOL	R	Module missing or inoperative (Fipio extension only)	%MWr.m.MOD.2.14



---

# Quick start : example of CANopen implementation



---

## At a glance

### Overview

This section presents an example of CANopen implementation.

### What's in this Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
10	Description of the application	167
11	Installing the application using Unity Pro	171
12	Starting the Application	209



---

## Description of the application

10

---

### Overview of the application

#### At a glance

The application described in this document is used for the driving of a working mobile.

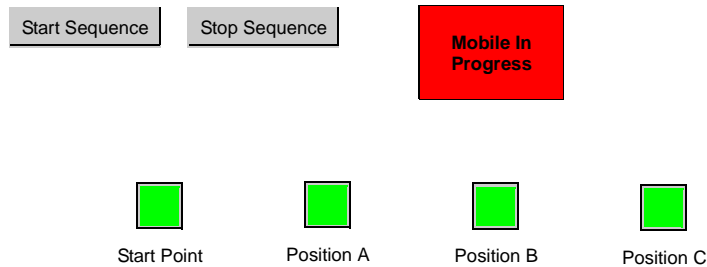
The mobile goes to different working positions following a defined position sequence. The mobile stops for few seconds at these positions.

The application's control resources are based on an operator screen which shows the status of the various position sensors and the actual mobile position value. A warning message blinks when the mobile is moving.

---

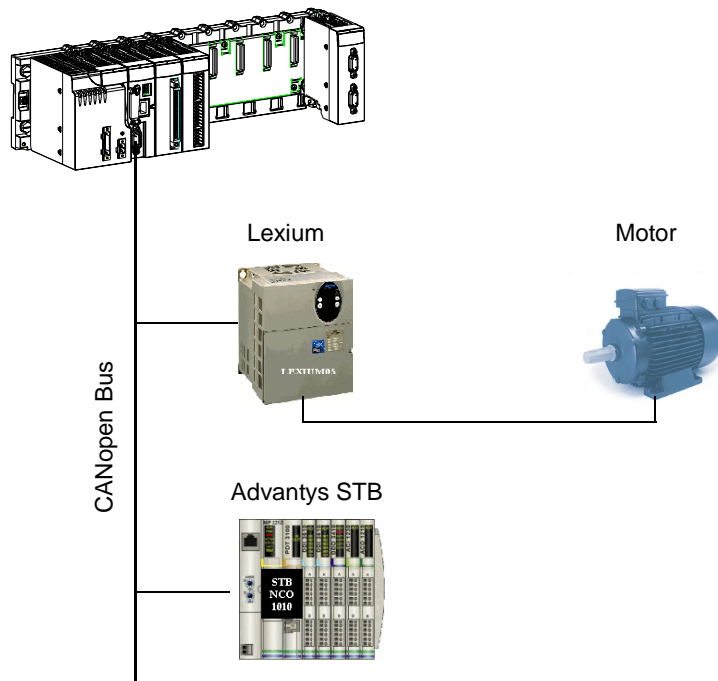
### Illustration of the application

This is the application's final operator screen:



The equipments can be connected as follow:

BMX P34 2010





**Operating mode**

The operating mode is as follows:

- A **Start Sequence** button is used to start the defined sequence.
  - In this example, the mobile first goes to B position then to the A position and, at the end, to the C position, before coming back to the Start Point, waiting for a new start-up request.
  - The mobile stops for few seconds at each position to simulate an action time.
  - A **Stop Sequence** button interrupts the mobile sequence. The mobile stops to the last targeted position and comes back to the Start Point, waiting for a new start-up request.
-



---

# Installing the application using Unity Pro

11

---

## At a glance

### Subject of this chapter

This chapter describes the procedure for creating the application described. It shows, in general and in more detail, the steps in creating the different components of the application.

### What's in this Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	Presentation of the solution used	173
11.2	Developping the application	176



---

# 11.1                    Presentation of the solution used

---

## At a glance

---

**Subject of this section**                    This section presents the solution used to develop the application. It explains the technological choices and gives the application's creation timeline.

---

**What's in this Section?**                    This section contains the following topics:

Topic	Page
Technological choices used	174
The different steps in the process using Unity Pro	175

---

## Technological choices used

---

**At a glance**

There are several ways of writing a mobile driving application using Unity Pro. The one proposed, uses a Lexium 05 servo drives and Advantys STB island set up on a CANopen network.

---

**Technological choices**

The following table shows the technological choices used for the application:

Objects	Choices used
Lexium Operating Mode	Use of the Positioning Mode. This mode allows you to send a target position to the Lexium 05 servo drives through the CANopen network.
Sensor Interface	Use of a STB Advantys. This device is an assembly of distributed I/O, power, and other modules that function together as an island node on an open field bus network
Supervision screen	Use of elements from the library and new objects.
Main supervision program	This program is developed using a sequential function chart (SFC), also called GRAFCET. The various sections and transitions are created in Ladder Diagram (LD) language and in Structured Text language (ST).

**Note:** This example shows PDO and SDO exchange towards a speed drive. However, for speed drive configuration and control, the use of Motion Function Block is recommended.

---

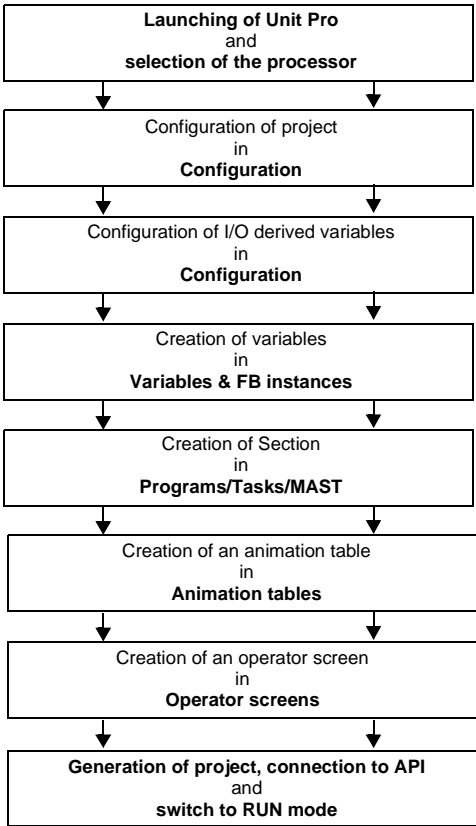
## The different steps in the process using Unity Pro

### At a glance

The following logic diagram shows the different steps to follow to create the application. A chronological order must be respected in order to correctly define all of the application elements.

### Description

Description of the different types:



# 11.2            Developping the application

## At a glance

**Subject of this section**            This section gives a step-by-step description of how to create the application using Unity Pro.

**What's in this Section?**            This section contains the following topics:

Topic	Page
Creating the project	177
Configuration of the CANopen Bus	178
Configuration of the CANopen Master	182
Configuration of the equipments	184
Declaration of variables	188
Creating the program in SFC for managing the move sequence	192
Creating a Program in LD for Application Execution	197
Creating a Program in LD for the operator screen animation	199
Creating a program in ST for the Lexium configuration	200
Creating an Animation Table	203
Creating the Operator Screen	205



## Creating the project

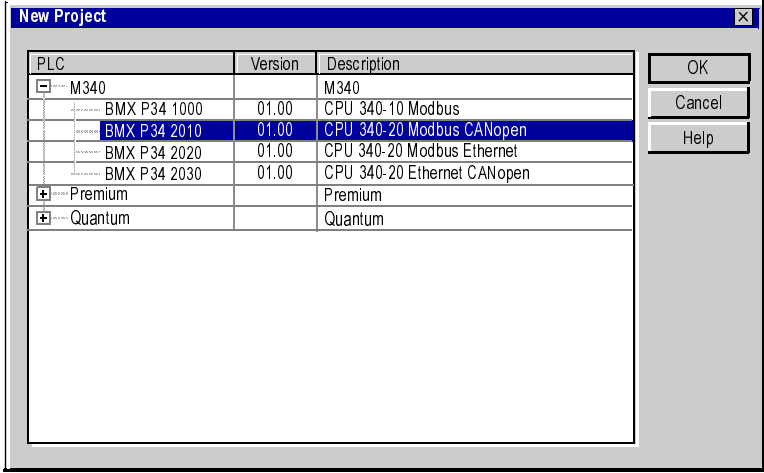
### At a glance

Developing an application using Unity Pro involves creating a project associated with a PLC.

**Note:** For more information, see Unity Pro online help (click on ?, then Unity, then Unity Pro, then Operate modes, and Project configuration).

### Procedure for creating a project

The table below shows the procedure for creating the project using Unity Pro.

Etape	Action
1	Launch the Unity Pro software (XL version in this example).
2	<div>Click on File then New to select a CANopen Master PLC (BMX P34 2010 for example): </div>
3	Confirm with OK.

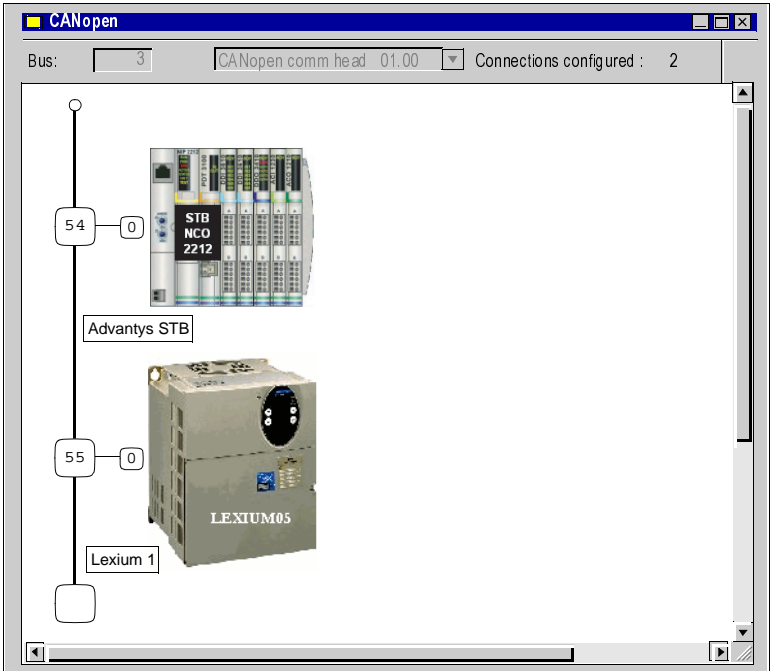
# Configuration of the CANopen Bus

## At a glance

Developing a CANopen application involves choosing the right slave devices and appropriate configuration.

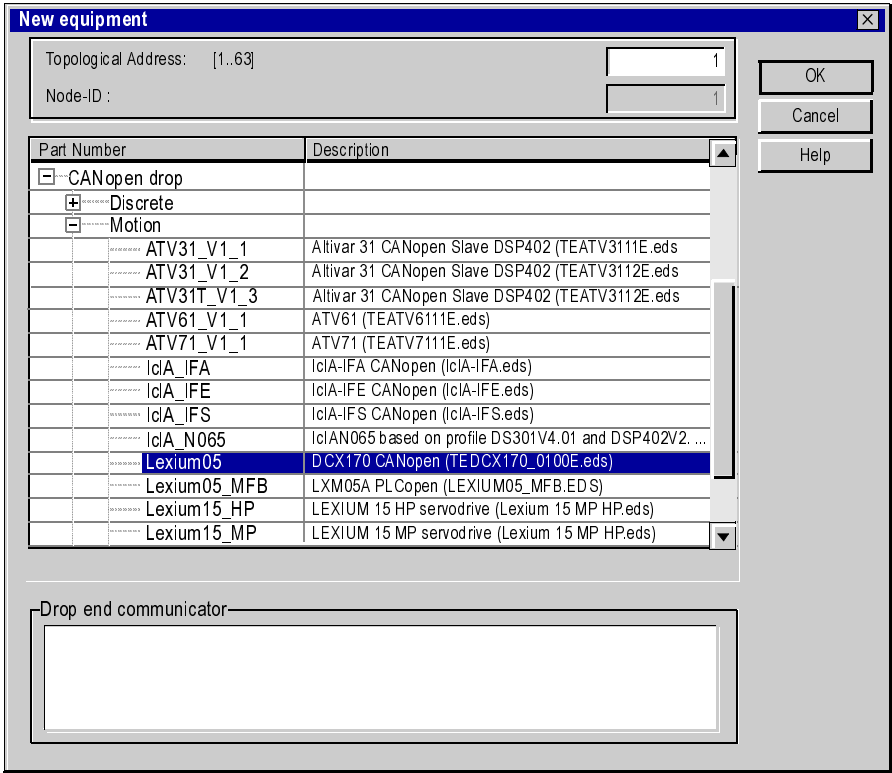
## Illustration of the CANopen bus

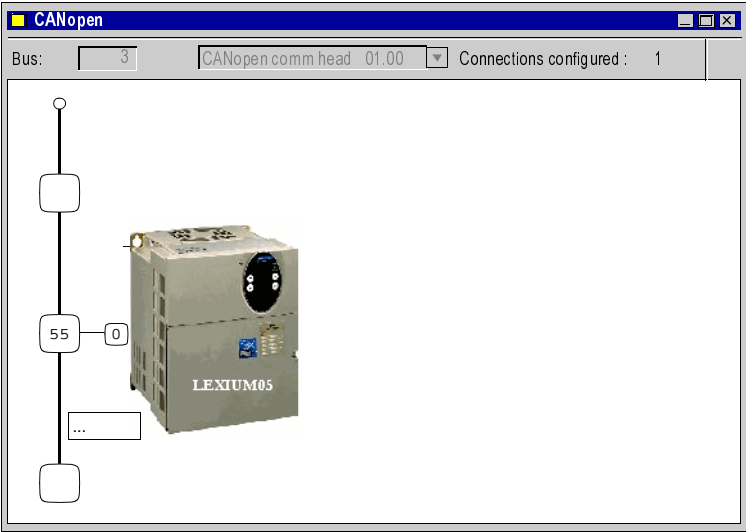
The following screen shows the configured CANopen bus:



# **CANopen bus Configuration**

The table below shows the procedure for selecting the CANopen slaves:

Step	Action
1	In the Project browser, double-click on Configuration then on 3 : CANopen. The CANopen Micro window opens.
2	In the CANopen Micro window, double-click on the node where the slave must be linked to. Result: the following window opens.
	
3	In the New Device window, enter the node number (55), then double click on Motion and select the Lexium05.

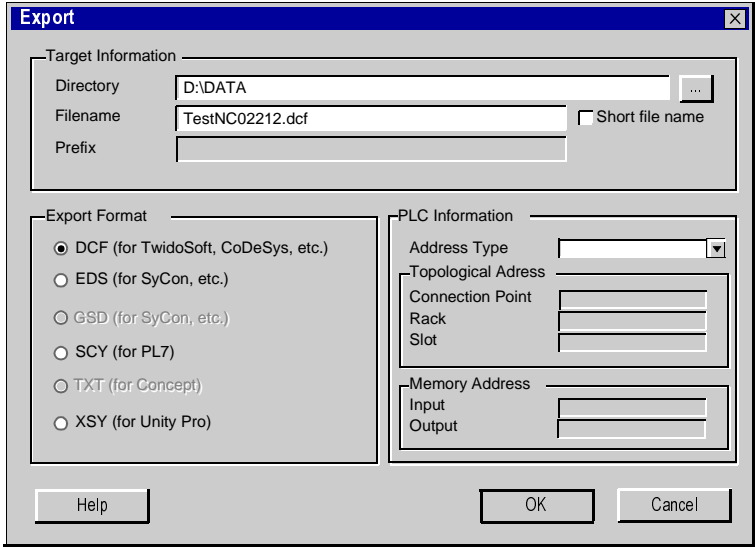
Step	Action
4	<div>Confirm with OK. Result: the slave module is declared.</div> <div></div>
5	<div>Follow the same procedure to declare the Advantys STB island. In the New Device window, enter the node number (54), then double click on Other and select the STB_NCO_2212.</div>

**Note:** This example shows PDO and SDO exchange towards a speed drive. However, for speed drive configuration and control, the use of Motion Function Block is recommended.

**Note:** This Advantys STB island configuration has to be set up using the Advantys Configuration Software.

# **STB island configuration**

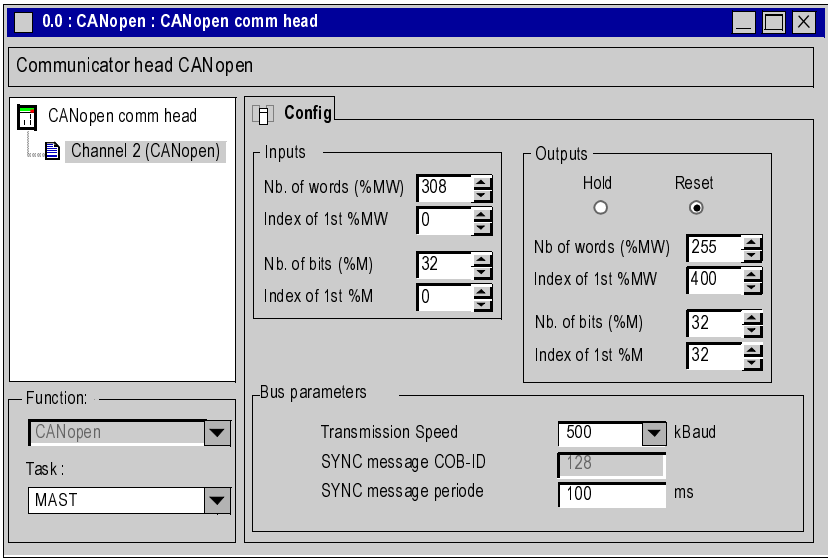
The table below shows the procedure to configure a STB island with Advantys Configuration Software:

Step	Action
1	Open Advantys Configuration Software Version 2.2.0.2 and create a new STB Island.
2	Insert a STB NCO2212 supply module, a STB DDI3420 discrete input module and a STB DD03410 discrete output module on the island.
3	<p>Save the configuration and click on <b>File/Export</b> for exporting the island in DCF format.</p> <p>The <b>Export</b> window opens:</p> 
4	Click on <b>OK</b>
5	Launch <b>Unity Pro</b> and open a project where an STB island will be used.
6	Add the STB equipment in the bus editor.
7	Right-click on the STB equipment then click on <b>Open the module</b>
8	In the <b>PDO</b> tab, click on the <b>Import DCF</b> button (see <i>Configuration of the STB</i> , p. 185).
9	Click on <b>OK</b> to validate.

## Configuration of the CANopen Master

**At a glance**      Developing a CANopen application involves choosing the right CANopen Master PLC configuration.

**CANopen Master PLC configuration**      The table below show the procedure for configuring the CANopen Master PLC:

Step	Action
1	In the Project browser double-click on Configuration then on 0:BMS XBP 0800 then on 0:BMX P34 2010. double click on CANopen to access to the CANopen Comm Head window.
2	In the input and output configuration zones, enter the index of the 1st word (%MW) and the needed number of words.
3	In the Bus Parameter zone, select the application transmission speed. In this example, select 500 kBauds. <div></div>
4	Click on the <input checked="" type="checkbox"/> button in the toolbar to validate the configuration.

**Note:** When the project is build, warning and error messages can be displayed in the output window. If it not displayed, click on `View/Output Window`.  
Warning messages indicates that there are more configured words than necessary on the bus.  
Error messages indicate that configured words are missing.

---

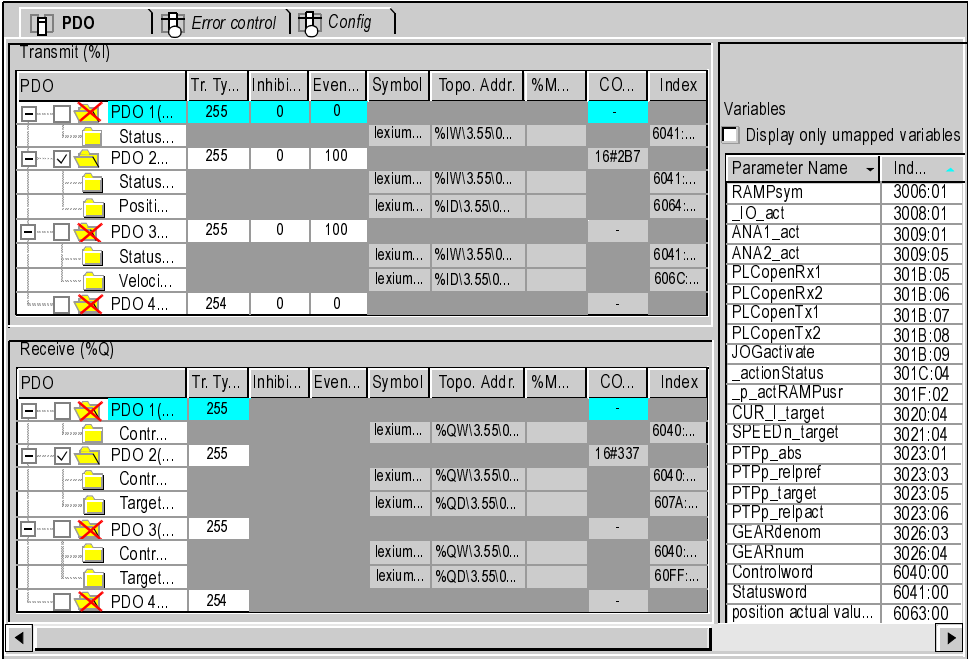
## Configuration of the equipments

### At a glance

Once the slave is declared, it's possible to have access to its configuration window.

### Configuration of the Lexium Servo Drives

The table below shows the procedure for the Lexium configuration:

Step	Action
1	In the <b>Project</b> browser, double-click on <b>Configuration</b> then 3: <b>CANopen</b> .
2	In the <b>CANopen</b> window, double-click on the Lexium representation. The Lexium configuration window opens.
3	Click on the <b>PDO</b> tab to see the PDO configuration, the variables and their topological addresses.
4	For this example, select <b>PDO2(Static)</b> in the <b>Transmit (%I)</b> and the <b>Received (%Q)</b> windows.
	
5	Click on the <b>Error control</b> tab and set the <b>Node Heartbeat producer time</b> to 300ms
6	Click on the <input checked="" type="checkbox"/> button in the toolbar to validate the configuration.
7	Close the window.



# Configuration of the STB

The table below shows the procedure to load the configuration defined with the Advantys Configuration software:

Step	Action
1	In the <code>Project</code> browser, double-click on <code>Configuration</code> then 3: <code>CANopen</code> .
2	In the <code>CANopen</code> window, double-click on the Advantys STB representation. The STB NCO2212 configuration window opens.
3	<p>In <code>Function</code> zone, select <code>Autoconf</code>.</p> <div data-bbox="238 412 480 480" data-label="Image"> </div> <p>In this example, we use the <code>Autoconf</code> function because autoconfigurable modules are inserted on the STB island (see <i>Advantys STB configuration</i>, p. 214).</p>
4	Click on the <code>PDO</code> tab to see the PDO configuration, the variables and their topological addresses. Click on the right button of the horizontal scroll bar to see the <code>Import DCF</code> button.
5	<p>Click on <code>Import DCF</code> button to load the DCF configuration file generated with the Advantys Configuration Software.</p> <div data-bbox="200 688 1171 1347" data-label="Image"> </div>
6	Click on the <code>Error control</code> tab and set the <code>Node Heartbeat producer time</code> to 300ms
7	Click on the <input checked="" type="checkbox"/> button in the toolbart to validate the configuration.

Step	Action
8	Close the window. For more information about STB configuration, see <i>STB island configuration</i> , p. 235

**Declaration of I/O objects**

The table below shows the procedure to load the configuration defined with the Advantys Configuration software:

Step	Action
1	Open the \3.55\0.0 : Lexium05 window by clicking on the Lexium module icon in the CANopen window. Click on the Lexium05 and then on the I/O object tab.
2	Click on the I/O object prefix address %CH then on the Update grid button, the channel address appears in the I/O object grid.
3	Click on the line %CH\3.55\0.0 and then, in the I/O object creation window, enter a channel name in the prefix for name zone, Lexium for example.
4	Now click on different Implicit I/O object prefix addresses then on update grid button to see the names and addresses of the implicit I/O objects.

Overview

CANopen

I/O objects

I/O variable creation

Prefix for name:

Type:

Create

Comment:

I/O object

Channel: ☒ %CH

Configuration ☐ %KW ☐ %KD ☐ %KF

System ☐ %MW

Status ☐ %MW

Parameter ☐ %MW ☐ %MD ☐ %MF

Command ☐ %MW ☐ %MD ☐ %MF

Implicits ☐ %I ☒ %IW ☒ %ID ☐ %IF ☐ %ERR

☐ %Q ☒ %QW ☒ %QD ☐ %QF

Select all

Unselect all

Update

Update grid

Filter on usage

	Address	Name
1	%CH\3.55\0.0	Lexium
2	%ID\3.55\0.0	Lexium.Cap1Pos
3	%ID\3.55\0.0.2	Lexium.Cap2Pos
4	%ID\3.55\0.0.4	Lexium.param27
5	%ID\3.55\0.0.6	Lexium.param27
6	%ID\3.55\0.0.8	Lexium.p.actRAM
7	%ID\3.55\0.0.10	Lexium.position_a
8	%ID\3.55\0.0.12	Lexium.position
9	%ID\3.55\0.0.14	Lexium.Velocity_a
10	%IW\3.55\0.0.16	Lexium.IO_act
11	%IW\3.55\0.0.17	Lexium.ANA1_act
12	%IW\3.55\0.0.18	Lexium.ANA2_act
13	%IW\3.55\0.0.19	Lexium.Cap1Cou
14	%IW\3.55\0.0.20	Lexium.Cap2Cou
15	%IW\3.55\0.0.21	Lexium.actionStat
16	%IW\3.55\0.0.22	Lexium.Statuswo
17	%QD\3.55\0.0	Lexium.param27
18	%QD\3.55\0.0.2	Lexium.param27
19	%QD\3.55\0.0.4	Lexium.param35
20	%QD\3.55\0.0.6	Lexium.param35
21	%QD\3.55\0.0.8	Lexium.param35
22	%QD\3.55\0.0.10	Lexium.GEARden
23	%QD\3.55\0.0.12	Lexium.GEARnu
24	%QD\3.55\0.0.14	Lexium.Target_P
25	%QD\3.55\0.0.16	Lexium.Profile_Vel
26	%QD\3.55\0.0.18	Lexium.Target_Vel
27	%QW\3.55\0.0.20	Lexium.Param_6
28	%QW\3.55\0.0.21	Lexium.JOGactiva
29	%QW\3.55\0.0.22	Lexium.CUR_1_ta
30	%QW\3.55\0.0.23	Lexium.SPEEDn
31	%QW\3.55\0.0.24	Lexium.param35
32	%QW\3.55\0.0.25	Lexium.Controlw

186

35013944 01 November 2007

**Note:** Repeat the same procedure to create a CANopen I/O object named BusMaster (%CH0.0.2). In the PLC bus window, double-click on the CANopen port then click on CANopen comm head to access the I/O objects tab.

---

## Declaration of variables

---

### At a glance

All of the variables used in the different sections of the program must be declared.  
Undeclared variables cannot be used in the program.

**Note:** For more information, see Unity Pro online help (click on **?**, then **Unity**, then **Unity Pro**, then **Operate modes**, and **Data editor**).

### Procedure for declaring variables

The table below shows the procedure for declaring application variables:

























Step	Action
1	In <b>Project browser</b> / <b>Variables &amp; FB instances</b> , double-click on <b>Elementary variables</b>
2	In the <b>Data editor</b> window, select the box in the <b>Name</b> column and enter a name for your first variable.
3	Now select a <b>Type</b> for this variable.
4	When all your variables are declared, you can close the window.

## Variables used for the application

The following table shows the details of the variables used in the application:

Variable	Type	Definition
Action_Time	TIME	Mobile stopping time at each position.
Configuration_Done	BOOL	The Lexium configuration is done.
Homing_Done	BOOL	The definition of the origin point is done.
index_subindex	DINT	CANopen parameter addresses for the WRITE_VAR block.
Lexium_Config_Step	INT	Configuration steps (program)
Lexium_Disabling	INT	Shutdown command
Lexium_operation_enable	INT	Command to start the Lexium drive.
Mobile_at_Position_A	BOOL	Mobile at the A position.
Mobile_at_Position_B	BOOL	Mobile at the B position.
Mobile_at_Position_C	BOOL	Mobile at the C position.
Mobile_at_start_position	BOOL	Mobile at the start position.
Mobile_in_Progress	BOOL	The mobile is moving.
New_SetPoint	BOOL	Start the next move.
Operation_done	BOOL	The mobile operation is done.
Position_A	DINT	First positioning value.
Position_B	DINT	Second positioning value.
Position_C	DINT	Third positioning value.
Ready_For_Stop	BOOL	The mobile goes to the last targeted position indicated before stopping the application. Then it comes back to the start position.
Run	BOOL	Start of the sequence.
Sequence_Number	INT	Number of displacements made by the mobile.
Start_Configuration	EBOOL	Start the Lexium configuration.
Stop	BOOL	The mobile stops the sequence and comes back to the start point.
Target_Reached	BOOL	The target position is reached.

The following screen shows the application variables created using the data editor :

Data Editor				
Variables   DDT types   Function blocks   DFB types				
Filter  Name <input type="text" value="*"/> <input checked="" type="checkbox"/> EDT <input type="checkbox"/> DDT <input type="checkbox"/> IODDT				
Name	Type	Address	Value	Comment
 Action_Time	TIME		t#3s	
 Configuration_Done	BOOL...			
 Homing_Done	BOOL	%IW3.55\0.0.0.22.14		
 index_subindex	DINT			
 Lexium_Config_Step	INT			
 Lexium_Disabling	INT		6	
 Lexium_operation_enable	INT		15	
 Mobile_at_Position_A	BOOL			
 Mobile_at_Position_B	BOOL			
 Mobile_at_Position_C	BOOL			
 Mobile_at_Start_Position	BOOL			
 Mobile_In_Progress	BOOL			
 New_SetPoint	BOOL	%QW3.55\0.0.0.25.4		
 Operation_Done	BOOL			
 Position_A	DINT		50000	
 Position_B	DINT		100000	
 Position_C	DINT		200000	
 Ready_For_Stop	BOOL			
 Run	BOOL			
 Sequence_Number	INT			
 Start_Configuration	EBOOL			
 Stop	BOOL			
 Target_Reached	BOOL	%IW3.55\0.0.0.22.10		
				

**Note:** At start-up, the Lexium 05 is in Ready to switch on state (rdy is displayed). To be able to drive the motor, the Lexium must be in Operation enable state. To switch in this state, a bus command sets the 4 last bits of the Lexium control word to '1' (00001111 (binary) = 15 (decimal)). To switch the Lexium 05 to the Ready to switch on state, a bus command sets the sixth and the seventh bit of the Lexium control word to '1' (00000110 (binary) = 6 (decimal)) For more information on Lexium control word, consult the Lexium manufacturer manuel

---

## Creating the program in SFC for managing the move sequence

---

### At a glance

The main program is written in SFC (Grafcet). The different sections of the grafcet steps and transitions are written in LD. This program is declared in a MAST task, and will depend on the status of a Boolean variable.

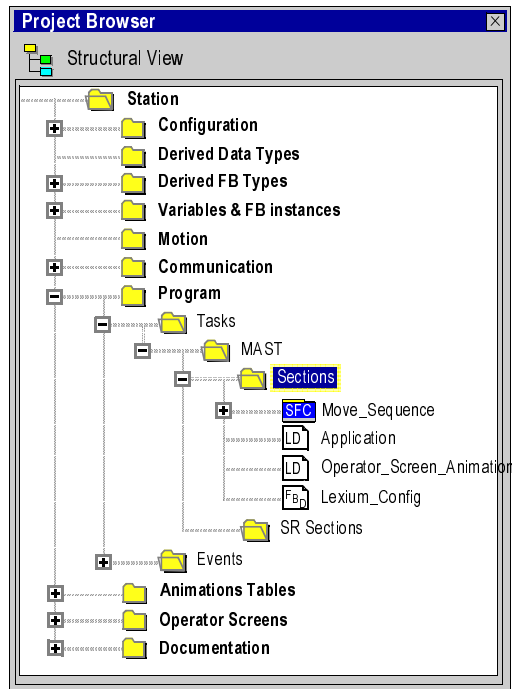
The main advantage of SFC language is that its graphic animation allows us to monitor in real time the execution of an application.

Several sections are declared in the MAST task:

- The **Move\_Sequence** (See *Illustration of the Move\_Sequence section, p. 195*) section, written in SFC and describing the operate mode.
- The **Application** (See *Creating a Program in LD for Application Execution, p. 197*) section, written in LD, which executes the mobile action delay time and resets the positioning start bit `New_Setpoint`.
- The **Operator\_Screen\_Animation** (See *Creating a Program in LD for the operator screen animation, p. 199*) section, written in LD which is used to animate the operator screen.
- The **Lexium\_Config** (See *Creating a program in ST for the Lexium configuration, p. 200*) section, written in ST and describing the different steps of the Lexium configuration.



In the project browser, the sections are represented as follow:





**Note:** The LD, SFC and FBD-type sections used in the application must be animated in online mode (See *Execution of Application in Standard Mode*, p. 209), with the PLC in RUN

**Note:** If task cycle is faster than CANopen Master cycle, outputs can be overwritten. To avoid that, it is recommended to have a task cycle higher than the CANopen Master cycle.

**Procedure for  
Creating an SFC  
Section**

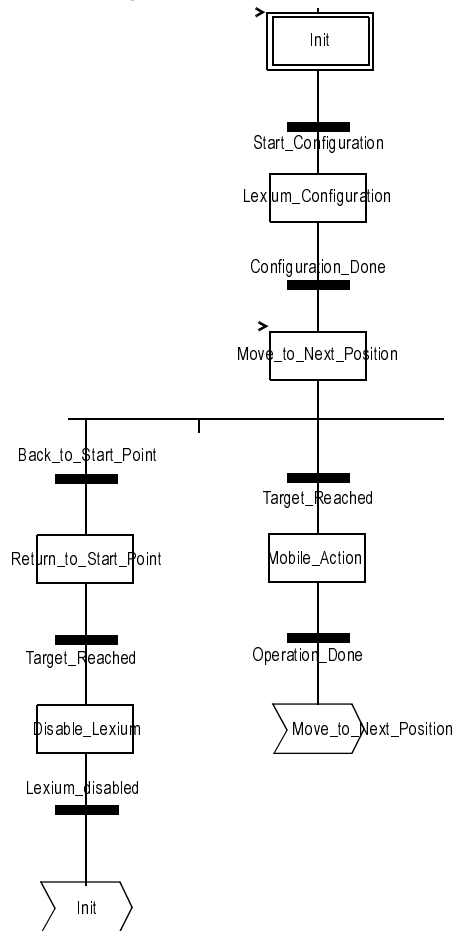
The table below shows the procedure for creating an SFC section for the application.

Step	Action
1	In <code>Project Browser\Program\Tasks</code> , double-click on <code>MAST</code> .
2	Right click on <code>Section</code> then select <code>New section</code> . Give your section a name ( <code>Movement_sequence</code> for the SFC section) then select SFC language.
3	The name of your section appears, and can now be edited by double clicking on it.
4	<p>The SFC edit tools appear in the window, which you can use to create your Grafcet.</p> <p>For example, to create a step with a transition:</p> <ul style="list-style-type: none"><li>● To create the step, click on  then place it in the editor,</li><li>● To create the transition, click on  then place it in the editor (generally under the preceding step).</li></ul>

---

### Illustration of the Move\_Sequence section

The following screen shows the application Grafcet. There is no condition defined:




For actions and transitions used in the grafcet, see *Actions and transitions*, p. 239

**Note:** For more information on creating an SFC section, see Unity Pro online help (click on ?, then Unity, then Unity Pro, then Operate modes, then Programming and SFC editor)

**Description of  
the  
Move\_Sequence  
Section**

The following table describes the different steps and transitions of the Move\_Sequence Grafcet:

Step / Transition	Description
Init	This is the initial state.
Start_Configuration	This transition is active when the variables: <ul style="list-style-type: none"><li>● Stop = 0,</li><li>● Run = 1.</li></ul>
Lexium_Configuration	The Lexium 05 is enabled and the 0 position is defined (using the Lexium's Homing function).
Configuration_done	The transition is active when the Lexium is initialized.
Move_to_next_position	The next target position is loaded in the Lexium 05. When this step is activated, the sequence number is incremented.
Target_reached	This variable is set to '1' by the Lexium 5 when the target position is reached.
Mobile_action	The mobile is at the target position and is operating an action.
Operation_done	This transition is active when the mobile operation is over.
Back_to_start_point	This transition is active when the sequence is over or when a stop request is ordered.
Return_to_start_point	The start point is defined at the target position.
Disable_Lexium	The Lexium 05 drive is disabled.
Lexium_disabled	This transition is valid when the Lexium is disabled.

**Note:** You can see all the steps and actions and transitions of your SFC by clicking on  in front of the name of your SFC section.

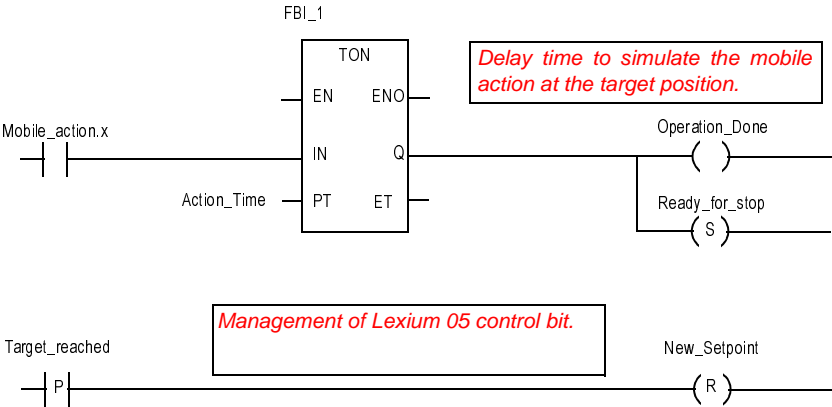
# Creating a Program in LD for Application Execution

## At a glance

This section executes the mobile action delay time and resets the positioning start bit New\_Setpoint.

## Illustration of the Application Section

The section below is part of the MAST task. It has no condition defined for it so is permanently executed:






## Description of the Application Section

- The first line is used to simulate the action time once the mobile is at the target position. When the Mobile\_Action step is active, a TON timer is triggered. When the PT time is reached, the TON output switches to '1', validate the transition variable Operation\_done and set the Ready\_for\_stop variable.
- The second line resets the variable New\_Setpoint on the Target\_reached positive transition.

**Procedure for  
Creating an LD  
Section**

The table below describes the procedure for creating part of the **Application** section.

Step	Action
1	In <code>Project Browser\Program\Tasks</code> , double-click on <code>MAST</code> .
2	Right click on <code>Section</code> then select <code>New section</code> . Name this section <code>Application</code> , then select the language type <code>LD</code> . The edit window opens.
3	To create the contact <code>Action_Mobile.x</code> , click on  then place it in the editor. Double-click on this contact then enter the name of the step with the suffix ".x" at the end (signifying a step of an SFC section). Confirm with OK.
4	To use the TON block you must instantiate it. Right click in the editor then click on <code>Data Selection</code> and on  . Click on the <code>Function and Function Block Types</code> tab. Click on <code>Libset</code> , select the TON block in the list then confirm with OK and position your block. To link the <code>Action_Mobile.x</code> contact to the Input of the TON function block, align the contact and the input horizontally, click on  and position the link between the contact and the input.

**Note:** For more information on creating an LD section, see Unity Pro online help (click on `?`, then `Unity`, then `Unity Pro`, then `Operate modes`, then `Programming and LD editor`).

---

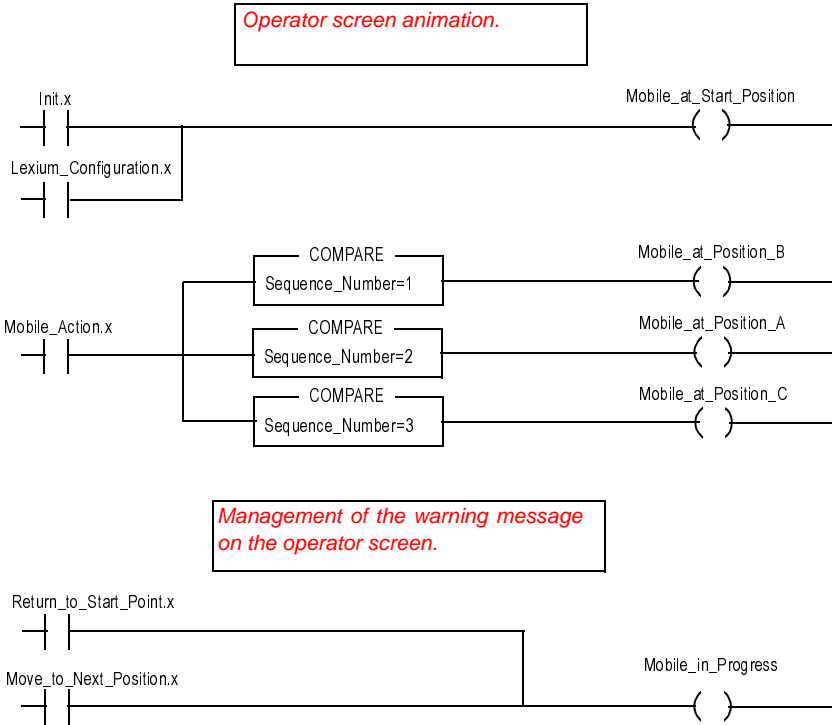
# Creating a Program in LD for the operator screen animation

## At a glance

This section animates the operator screen.

## Illustration of the Operator\_Screen\_Animation section

The section below is part of the MAST task. It has no condition defined for it so is permanently executed:



## Procedure for Creating an LD Section

For creating a LD section, see *Procedure for Creating an LD Section, p. 198*

## Creating a program in ST for the Lexium configuration

### At a glance

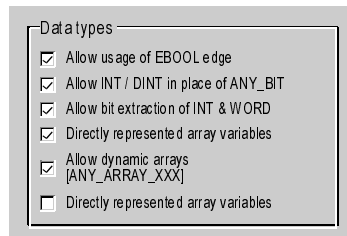
This section executes the different steps of the Lexium configuration. This section is only active when the step `Lexium_Configuration` is reached in the grafcet (see *Illustration of the Move\_Sequence section, p. 195*)

### Programming structure

The programming structure is as follow:

Step number	Step description
0	Starting command of the Lexium.
10	If the Lexium is in <b>Run State</b> , then it switch in <b>Homing</b> mode using a <code>WRITE_VAR</code> function.
20	If the result of <code>WRITE_VAR</code> is conclusive then go to step 30.
30	<b>Homing</b> method definition using a <code>WRITE_VAR</code> function. For more information about the reference movement method, please refer to the Lexium user manual).
40	If the result of <code>WRITE_VAR</code> is conclusive then go to step 50.
50	Starting of the <b>Homing</b> method.
60	The Homing is done.
70	The Lexium switches in <b>Positionning</b> Mode using a <code>WRITE_VAR</code> function.
80	If the result of <code>WRITE_VAR</code> is conclusive then the Lexium configuration is done.

**Note:** For a correct variable declaration, click on **Tools/Project Settings/ Language extension** then check "Directly represented array variables" and "Allow dynamic arrays".





**ST Program**

The example is programmed in ST structured literal language. The dedicated section is under the same master task (MAST).

```

CASE Lexium_Config_Step OF
0: (* Lexium is in "Ready to switch on" position *)
  IF (Lexium.statusword.0) THEN
    Lexium.controlword:=Lexium_operation_enable;
    Lexium_Config_Step := 10;
  END_IF;

10: (* Lexium is in "Run" position *)
  IF (Lexium.statusword.2) THEN (* Operating mode: Homing *)
    index_subindex:=16#00006060 (*CANopen parameter address*)
    %MW200:=6; (*Definition of the Lexium Function: Homing*)
    %MW162:=5; (*Time out 500ms*)
    %MW163:=1; (*Length 1 byte*)
    WRITE_VAR(ADDM('0.0.2.55'),'SDO',index_subindex,0,%MW200:1,
%MW160:4);
    Lexium_Config_Step:=20;
  END_IF;

20: (* Test WRITE_VAR function result *)
  IF (NOT %MW160.0) THEN (* test activity bit*)
    IF (%MW161=0) THEN (* correct exchange*)
      Lexium_Config_Step := 30;
    END_IF;
  END_IF;

30: (* Homing method: set dimensions *)
  index_subindex:=16#00006098
  %MW150:=35; (*Definition of Homing method*)
  %MW252:=5; (*Time out 500ms*)
  %MW253:=1; (*Length 1 byte*)
  WRITE_VAR(ADDM('0.0.2.55'),'SDO',index_subindex,0,%MW150:1,
%MW250:4);

```

```
Lexium_Config_Step:=40;

40: (* Test WRITE_VAR function result *)
  IF (NOT %MW250.0) THEN (* test activity bit*)
    IF (%MW251=0) THEN (* correct exchange*)
      New_Setpoint:=0;
      Lexium_Config_Step := 50;
    END_IF;
  END_IF;

50: (* Trigger homing *)
  New_Setpoint :=1;
  Lexium_Config_Step:=60;

60: (* Homing done *)
  IF (Target_Reached) AND (Homing_Done) THEN
    New_Setpoint:=0;
    Lexium_Config_Step:=70;
  END_IF;

70: (* Operating mode: Positionnig *)
  index_subindex:=16#00006060
  %MW450:=1; (*Definition of Positionning method*)
  %MW352:=5; (*Time out 500ms*)
  %MW353:=1; (*Length 1 byte*)
  WRITE_VAR(ADDM('0.0.2.55'),'SDO',index_subindex,0,%MW450:1,
  %MW350:4);
  Lexium_Config_Step:=80;

80: (* Test WRITE_VAR function result *)
  IF (NOT %MW350.0) THEN (* test activity bit*)
    IF (%MW351=0) THEN (* correct exchange*)
      Configuration_Done := 1;
    END_IF;
  END_IF;

END_CASE;
```

---

## Creating an Animation Table


### At a glance

An animation table is used to monitor the values of variables, and modify and/or force these values. Only those variables declared in `Variables & FB` instances can be added to the animation table

**Note: Note:** For more information, consult the Unity Pro online help (click `?`, then `Unity`, then `Unity Pro`, then `Operate modes`, then `Debugging and adjustment` then `Viewing and adjusting variables and Animation tables`).

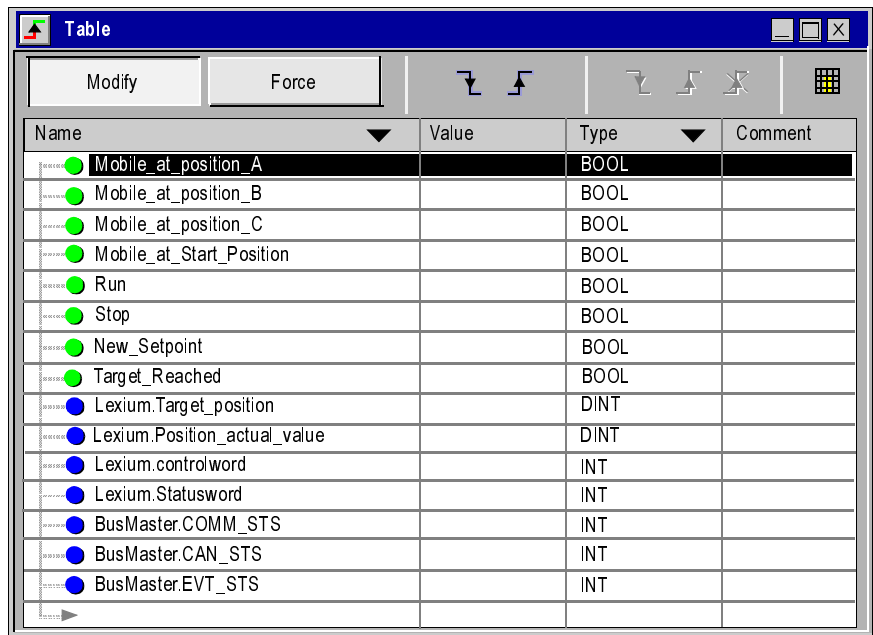
### Procedure for Creating an Animation Table

The table below shows the procedure for creating an animation table.

Step	Action
1	In the <code>Project browser</code> , right click on <code>Animation tables</code> then click on <code>New Animation Table</code> . The edit window opens.
2	Click on first cell in the <code>Name</code> column, then on the  button, and add the variables you require.

**Animation Table  
Created for the  
Application**

The following screen shows the animation table used by the application:



Name	Value	Type	Comment
Mobile_at_position_A		BOOL	
Mobile_at_position_B		BOOL	
Mobile_at_position_C		BOOL	
Mobile_at_Start_Position		BOOL	
Run		BOOL	
Stop		BOOL	
New_Setpoint		BOOL	
Target_Reached		BOOL	
Lexium.Target_position		DINT	
Lexium.Position_actual_value		DINT	
Lexium.controlword		INT	
Lexium.Statusword		INT	
BusMaster.COMM_STS		INT	
BusMaster.CAN_STS		INT	
BusMaster.EVT_STS		INT	

For more information about the creation of the Lexium and the BusMaster objects, see *Declaration of I/O objects, p. 186*

**Note:** The animation table is dynamic only in online mode (display of variable values)

**Note:** COMM\_STS, CAN\_STS and EVT\_STS words are used to check the application good operating. For more information, consult the CANopen user manual.

**Note:** To fill the animation table quickly, select several variables by maintaining the **Control** button.

## Creating the Operator Screen

---

### At a glance

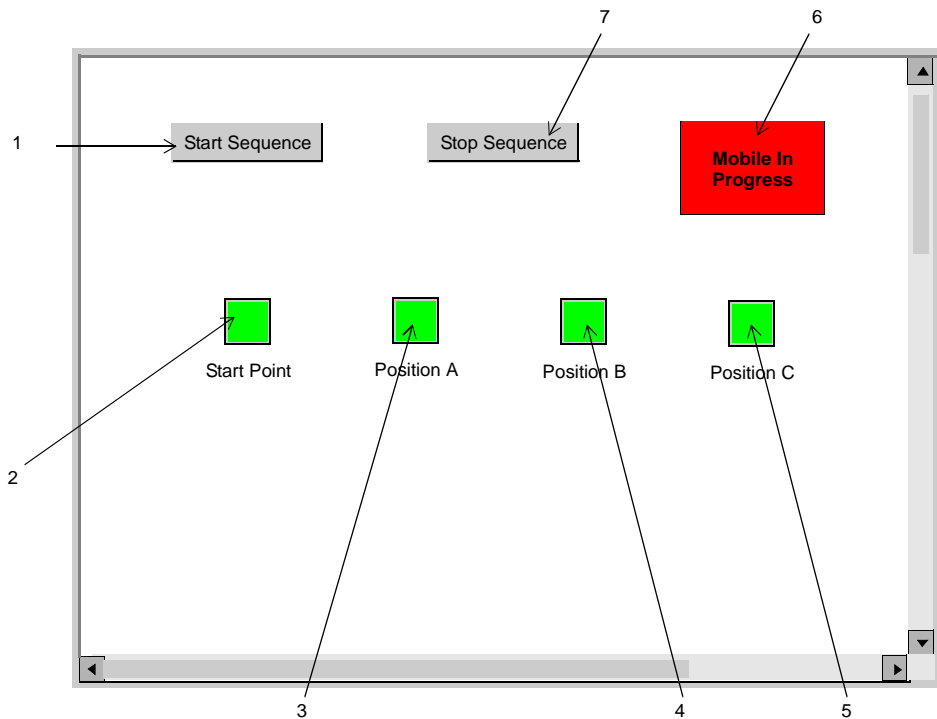
The operator screen is used to animate graphic objects that symbolize the application. These objects can belong to the Unity Pro library, or can be created using the graphic editor.

<p><b>Note:</b> For more information, see Unity Pro online help (click on ?, then Unity, then Unity Pro, then Operate modes, and Operator screens).</p>
---

---


**Illustration of the Operator Screen**

The following illustration shows the application operator screen:





The associated variables are presented in the table below:

N°	Description	Associated variable
1	Start button	Run
2	Start point light indicator	Mobile_At_Start_Position
3	"Position" A light indicator	Mobile_At_Position_A
4	"Position B" light indicator	Mobile_At_Position_B
5	"Position C" light indicator	Mobile_At_Position_C
6	"Mobile in progress" light indicator	Mobile_in_Progress
7	Stop button	Stop




**Note:** To animate objects in online mode, you must click on . By clicking on this button, you can validate what is written.

### Procedure for Creating an Operator Screen

The table below shows the procedure for creating the Start button.

Step	Action
1	In the <code>Project</code> browser, right click on <code>Operator screens</code> and click on <code>New screen</code> . The operator screen editor appears.
2	Click on the  and position the new button on the operator screen. Double click on the button and in the <code>Control</code> tab, select the <code>Run</code> variable by clicking the button  and confirm with OK. Then, enter the button name in the text zone.

The table below shows the procedure for inserting and animating indicator light.

Step	Action
1	In the <code>Tools</code> menu, select <code>Operator screens Library</code> . Double click on <code>Display unit</code> then <code>Indicator light</code> . Select the dynamic green light from the runtime screen and Copy (Ctrl+C) then Paste (Ctrl+V) it into the drawing in the operator screen editor.
2	The light is now in your operator screen. Select your light then click on  . Press enter and the object properties window opens. Select the <code>Animation</code> tab and enter the concerned variable, by clicking on  (in the place of %MW1.0). Click on  and enter the same variable.
3	Confirm with apply and OK.





# Starting the Application

12

## Execution of Application in Standard Mode

**At a glance** To work in standard mode, you need to associate defined variables to PDO addresses of the equipment declared on CANopen Bus.

**Note:** For more information on addressing, see Unity Pro online help (click on [?](#), then [Unity](#), then [Unity Pro](#), then [Languages](#) reference, then [Data description](#) and [Data instances](#)

### Assignment of variables

The table below shows the procedure for direct addressing of variables:


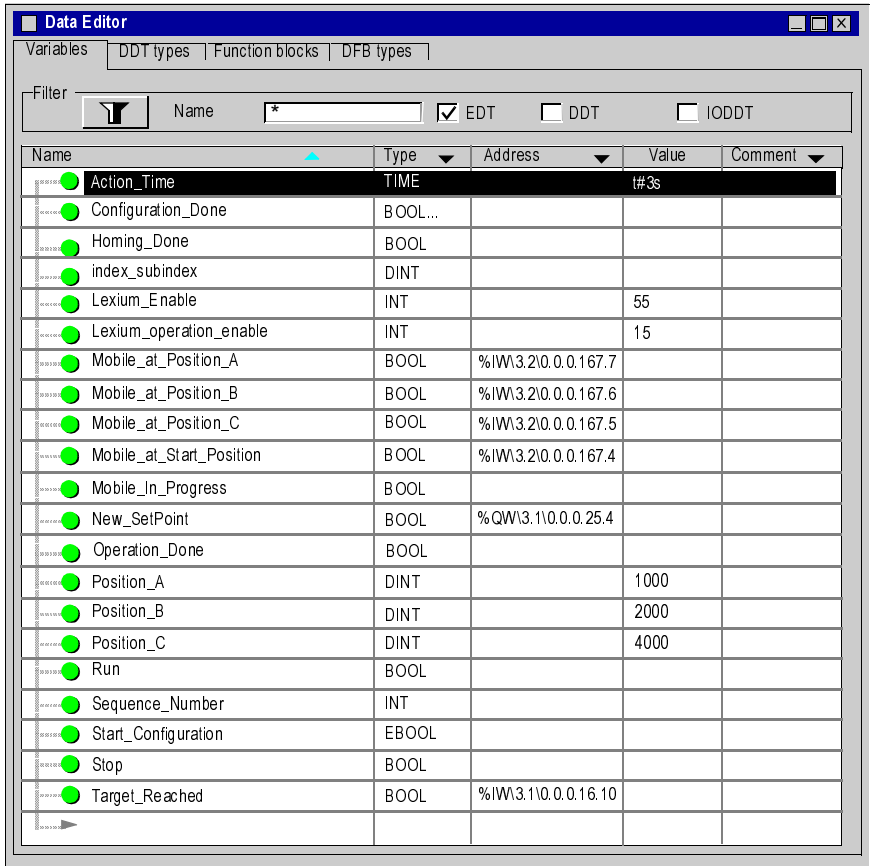
Step	Action
1	In the <code>Project browser</code> and in <code>Variables &amp; FB instances</code> , double-click on <code>Elementary variables</code> .
2	In the <code>Address</code> column, enter the address associated with the variable in the form <code>\Bus.Node\Rack.Module.Channel.Data</code> . 
3	Repeat the same procedure for all located variables.

Illustration of assigned variables

The following screen shows the application variables assignment:



The screenshot shows the 'Data Editor' window with the 'Variables' tab selected. The window contains a table of variable assignments. The table has columns for Name, Type, Address, Value, and Comment. The variables are listed in the table, including Action\_Time, Configuration\_Done, Homing\_Done, index\_subindex, Lexium\_Enable, Lexium\_operation\_enable, Mobile\_at\_Position\_A, Mobile\_at\_Position\_B, Mobile\_at\_Position\_C, Mobile\_at\_Start\_Position, Mobile\_In\_Progress, New\_SetPoint, Operation\_Done, Position\_A, Position\_B, Position\_C, Run, Sequence\_Number, Start\_Configuration, Stop, and Target\_Reached. The values for some variables are shown, such as t#3s for Action\_Time, 55 for Lexium\_Enable, 15 for Lexium\_operation\_enable, 1000 for Position\_A, 2000 for Position\_B, 4000 for Position\_C, and %IW3.2\0.0.0.167.7 for Mobile\_at\_Position\_A.

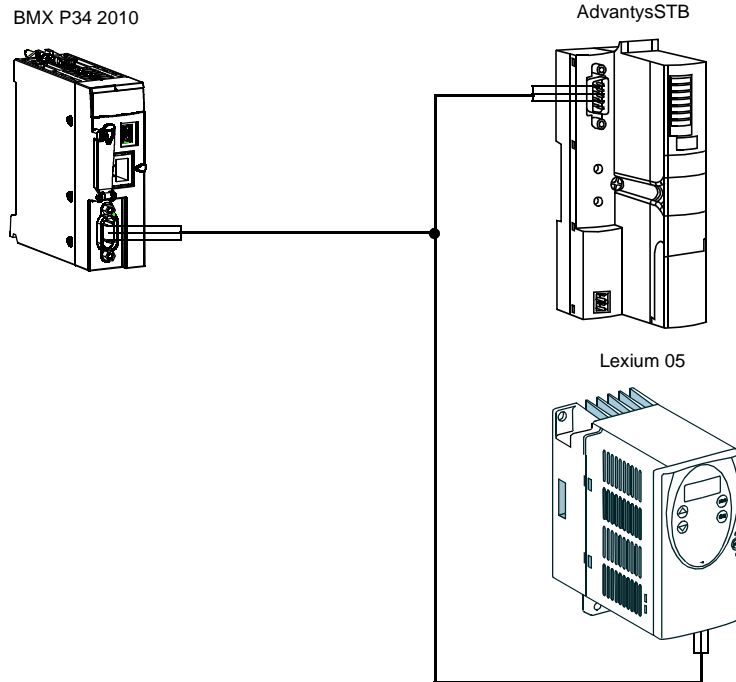
Name	Type	Address	Value	Comment
Action_Time	TIME		t#3s	
Configuration_Done	BOOL...			
Homing_Done	BOOL			
index_subindex	DINT			
Lexium_Enable	INT		55	
Lexium_operation_enable	INT		15	
Mobile_at_Position_A	BOOL	%IW3.2\0.0.0.167.7		
Mobile_at_Position_B	BOOL	%IW3.2\0.0.0.167.6		
Mobile_at_Position_C	BOOL	%IW3.2\0.0.0.167.5		
Mobile_at_Start_Position	BOOL	%IW3.2\0.0.0.167.4		
Mobile_In_Progress	BOOL			
New_SetPoint	BOOL	%QW3.1\0.0.0.25.4		
Operation_Done	BOOL			
Position_A	DINT		1000	
Position_B	DINT		2000	
Position_C	DINT		4000	
Run	BOOL			
Sequence_Number	INT			
Start_Configuration	EBOOL			
Stop	BOOL			
Target_Reached	BOOL	%IW3.1\0.0.0.16.10		

Description of variables assignment.

- The first four Boolean variables are assigned to the four discrete inputs of the STB DDI 3420 module.
- New\_Setpoint is assigned to the Lexium 05 control bit. A positive transition of this bit triggers a new positioning.
- Target\_Reached is assigned to the Lexium 05 status bit which is set to '1' when the target is reached.

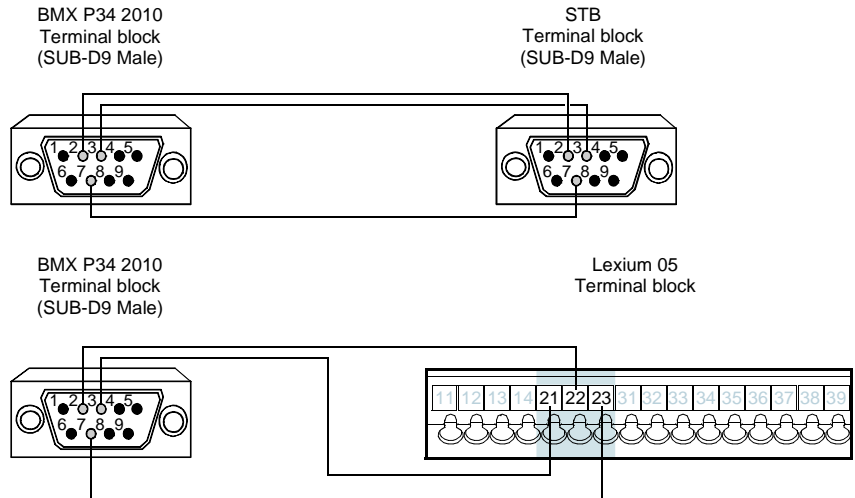
**CANopen bus wiring**

The CANopen bus is connected as follow:



**Note:** The Lexium 05 is at the end of the CANopen Bus. Set the Terminating resistor CAN switch to '1'.

The assignment of the pins connectors is as follow:



BMX P34 2010 terminal block description:

Pin number	Symbol	Description
1	-	Reserved
2	CAN_L	CAN_L bus line (Low)
3	CAN_GND	CAN ground
4	-	Reserved
5	Reserved	Optional CAN protection
6	(GND)	Optional ground
7	CAN_H	CAN_H bus line (High)
8	-	Reserved
9	Reserved	CAN external positive supply (optionnal)

STB terminal block description:

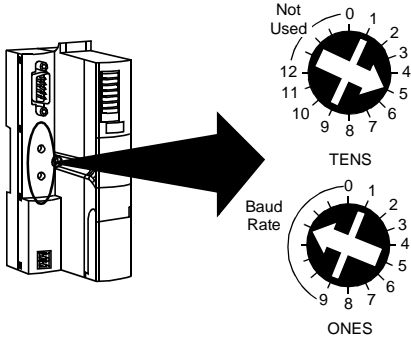
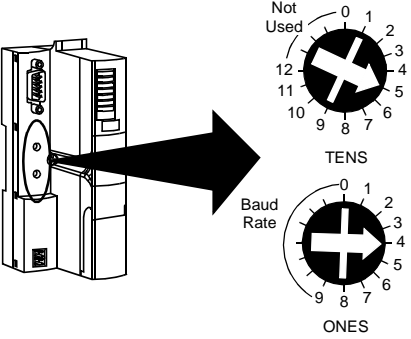
Pin number	Symbol	Description
1	-	Reserved
2	CAN_L	CAN_L bus line (Low)
3	CAN_GND	CAN ground
4	-	Reserved
5	(CAN_SHLD)	Optional CAN protection
6	(GND)	Optional ground
7	CAN_H	CAN_H bus line (High)
8	-	Reserved
9	-	Reserved

Lexium 05 terminal block description:

Pin number	Symbol	Description
21	CAN_GND	CAN ground
22	CAN_L	CAN_L bus line (Low)
23	CAN_H	CAN_H bus line (High)

Advantys STB configuration

The table below shows the procedure for configuring the Lexium 05:

Step	Action
1	Shut down the STB.
2	<p>Using the rotary switches (located on the front of the CANopen NIM), configure the baud rate. The rotary switches are positionned as followed (5 = 500 kbits/s):</p> 
3	Start up then shut down the STB.
4	<p>Using the rotary switches, configure the address of the STB. For example, is the node number of the equiment is '54', the rotary switches are positionned as followed:</p> 
5	Start up the STB and press the reset button located on the STB NCO module during for 5 seconds.
6	The STB is configured automatically.

**Lexium  
configuration**

The table below shows the procedure for configuring the Lexium 05:

Step	Action
1	Start up the Lexium 05. RDY is displayed on ther interface.
2	Press <code>Enter</code>
3	Press the down arrow key until <code>COM-</code> is displayed. Then press <code>Enter</code> .
4	Press the down arrow key until <code>CoAD</code> (CANopen Address) is displayed. Then press <code>Enter</code> .
5	Using the arrow keys, configure the node number. Then press <code>Esc</code> .
6	Press the down arrow key until <code>CoBD</code> (CANopen Baud Rate) is displayed. Then press <code>Enter</code> .
7	Using the arrow keys, configure the baud rate (500). Then press <code>Esc</code> .
8	Press the <code>Esc</code> until RDY displayed.





---

# Appendices



---

## At a glance

**Overview** These appendices contain information that should be useful for programming the application.

**What's in this Appendix?** The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	CANopen Master local object dictionary entry	219
B	Relation between PDOs and STB variables	235
C	Actions and transitions	239



---

# CANopen Master local object dictionary entry



---

## At a glance

**Subject of this chapter** This chapter contains the local object dictionary entry for CANopen Master.

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

Topic	Page
Object Dictionary entries according Profile DS301	220
Object Dictionary entries according Profile DS302	225
Midrange Manufacturer Specific Object Dictionary Entries	227

## Object Dictionary entries according Profile DS301

### Object Dictionary entries

The table below presents the object dictionary entries according profile DS301.

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
1000		Device Type	VAR	Unsigned32	0x000F 0191
1001		Error Register	VAR	Unsigned8	
1005		COB-ID SYNC	VAR	Unsigned32	
1006		Communication Cycle Period	VAR	Unsigned32	
1007		Synchronous Window Length	VAR	Unsigned32	
1008		Manufacturer Device Name	VAR	String	BMX CPU 20x0
1009		Manufacturer Hardware Version	VAR	String	MIDRANGE BASIC
100A		Manufacturer Software Version	VAR	String	COMM_FW_01_xx
1012		COB-ID Time Stamp Message	VAR	Unsigned32	
1016		Consumer Heartbeat Time	ARRAY		
	0	Number of entries : 64		Unsigned8	
	1	Consumer Heartbeat Time		Unsigned32	
	...			Unsigned32	
	64			Unsigned32	
1017		Producer Heartbeat Time	VAR	Unsigned16	
1018		Identity Object	RECORD		
	0	Number of entries		Unsigned8	4
	1	Vendor ID		Unsigned32	0x0600 005A
	2	Product Code		Unsigned32	0x3300 FFFF
	3	Revision Number		Unsigned32	0xyyyy xxxx
	4	Serial Number		Unsigned32	0x0
				Unsigned32	
1020		Verify Configuration	ARRAY		
	0	Number of entries : 2		Unsigned8	
	1	Configuration date		Unsigned32	
	2	Configuration time		Unsigned32	
1200		1. Server SDO	RECORD		
	0	Number of entries		Unsigned8	

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
	1	COB-ID Client -> Server (Rx)		Unsigned32	600H + Node-ID
	2	COB-ID Server -> Client (Tx)		Unsigned32	580H + Node-ID
1280		1. Client SDO	RECORD		
	0	Number of entries		Unsigned8	
	1	COB-ID Client -> Server (Rx)		Unsigned32	
	2	COB-ID Server -> Client (Tx)		Unsigned32	
	3	Node-ID of the Server SDO		Unsigned8	
1281		2. Client SDO	RECORD		
	0	Number of entries		Unsigned8	
	1	COB-ID Client -> Server (Rx)		Unsigned32	
	2	COB-ID Server -> Client (Tx)		Unsigned32	
	3	Node-ID of the Server SDO		Unsigned8	
1282		3. Client SDO	RECORD		
	0	Number of entries		Unsigned8	
	1	COB-ID Client -> Server (Rx)		Unsigned32	
	2	COB-ID Server -> Client (Tx)		Unsigned32	
	3	Node-ID of the Server SDO		Unsigned8	
1400		1. Receive PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3			Unsigned16	
	4			Unsigned8	
	5	Event timer		Unsigned16	
1401		2. Receive PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3			Unsigned16	
	4			Unsigned8	
	5	Event timer		Unsigned16	
.....					
14FF		256. Receive PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3			Unsigned16	
	4			Unsigned8	
	5	Event timer		Unsigned16	
1600		1. Receive PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	
1601		2. Receive PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	
	.....				
16FF		256. Receive PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	
1800		1. Transmit PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3	Inhibit time		Unsigned16	
	4	Reserved		Unsigned8	
	5	Event timer		Unsigned16	
1801		2. Transmit PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3	Inhibit time		Unsigned16	
	4	Reserved		Unsigned8	
	5	Event timer		Unsigned16	
.....					
18FF		256. Transmit PDO	RECORD		
	0	Largest sub-index supported		Unsigned8	
	1	COB-ID used by PDO		Unsigned32	
	2	Transmission type		Unsigned8	
	3	Inhibit time		Unsigned16	
	4	Reserved		Unsigned8	
	5	Event timer		Unsigned16	
1A00		1. Transmit PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	
1A01		2. Transmit PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	
	.....				
1AFF		256. Transmit PDO Mapping			
	0	Number of mapped application objects in PDO		Unsigned8	Depends on PDO mapping of the application
	1	PDO mapping for the 1. Application object to be mapped		Unsigned32	Index (16 bit)   Sub-index (8 bit)   length (8 bit)
	2	PDO mapping for the 2. Application object		Unsigned32	
	.....				
	8	PDO mapping for the 8. Application object		Unsigned32	



## Object Dictionary entries according Profile DS302

### Object Dictionary entries

The table below presents the object dictionary entries according profile DS302.

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
1F22		Concise DCF	ARRAY		
	0	Number of entries	VAR	Unsigned8	
	1	Device with Node-ID 1	VAR	DOMAIN	
	...				
	127	Device with Node-ID 127		DOMAIN	
1F26		Expected Configuration Date	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F27		Expected Configuration Time	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F80		NMT Startup	VAR	Unsigned32	
1F81	...	Slave Assignment	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F82	...	Request NMT	ARRAY		
	0	Number of entries		Unsigned8	
	1	Request NMT for Node-ID 1		Unsigned8	
	...				
	128	Request NMT for all Nodes		Unsigned8	
1F84	...	Device Type Identification	ARRAY		
	0	Number of entries		Unsigned8	

Index (Hex)	Sub- index	Description	Object type	Data type	Comments
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F85	...	Vendor Identification	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F86	...	Product Code	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	
1F87	...	Revision Number	ARRAY		
	0	Number of entries		Unsigned8	
	1	Device with Node-ID 1		Unsigned32	
	...				
	127	Device with Node-ID 127		Unsigned32	

---

## Midrange Manufacturer Specific Object Dictionary Entries

**Project Data** The table below presents the Object Entry 2010 (Project Data).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
2010		Project Data	RECORD		
	0	Number of entries		Unsigned8	
	1	Current byte length		Unsigned16	Read only access Updated by the Master Manager
	2	Project data domain		DOMAIN	

**CANopen Master Timing Control** The table below presents the Object Entry 2100 (CANopen Master Timing Control).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
2100		CANopen Master Timing Control	ARRAY		
	0	Number of entries		Unsigned8	
	1	Max. number of TPDOs to transmit during one cycle		Unsigned8	
	2	Max. number of high priority receive queue accesses during one cycle (RPDOs, EMCY)		Unsigned8	
	3	Max. number of low priority receive queue accesses during one cycle (SDOs, Heartbeat/Guarding)		Unsigned8	

**CANopen Master Status**     The table below presents the Object Entry 4100 (CANopen Master Status).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4100		CANopen Master Status	ARRAY		
	0	Number of entries		Unsigned8	
	1	Global_events		Unsigned16	
	2	COMM_state		Unsigned8	
	3	COMM_diagnostic		Unsigned8	
	4	Config_bits		Unsigned16	
	5	LED_control		Unsigned16	
	6	Minimum Cycle Time		Unsigned8	
	7	Maximum Cycle Time		Unsigned8	

---

**Nd\_asg**     The table below presents the Object Entry 4101 (Nd\_asg).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4101		Nd_asg	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_asg[0,1,2,3		Unsigned32	
	2	Nd_asg[4,5,6,7		Unsigned32	
	3	Nd_asg[8,9,10,11		Unsigned32	
	4	Nd_asg[12,13,14,15		Unsigned32	

---

**Nd\_cfg**     The table below presents the Object Entry 4102 (Nd\_cfg).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4102		Nd_cfg	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_cfg[0,1,2,3		Unsigned32	
	2	Nd_cfg[4,5,6,7		Unsigned32	
	3	Nd_cfg[8,9,10,11		Unsigned32	
	4	Nd_cfg[12,13,14,15		Unsigned32	

---

**Nd\_asf** The table below presents the Object Entry 4103 (Nd\_asf).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4103		Nd_asf	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_asf[0,1,2,3		Unsigned32	
	2	Nd_asf[4,5,6,7		Unsigned32	
	3	Nd_asf[8,9,10,11		Unsigned32	
	4	Nd_asf[12,13,14,15		Unsigned32	

**Nd\_oper** The table below presents the Object Entry 4104 (Nd\_oper).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4104		Nd_oper	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_oper[0,1,2,3		Unsigned32	
	2	Nd_oper[4,5,6,7		Unsigned32	
	3	Nd_oper[8,9,10,11		Unsigned32	
	4	Nd_oper[12,13,14,15		Unsigned32	

**Nd\_stop** The table below presents the Object Entry 4105 (Nd\_stop).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4105		Nd_stop	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_stop[0,1,2,3		Unsigned32	
	2	Nd_stop[4,5,6,7		Unsigned32	
	3	Nd_stop[8,9,10,11		Unsigned32	
	4	Nd_stop[12,13,14,15		Unsigned32	

**Nd\_preop**                      The table below presents the Object Entry 4106 (Nd\_preop).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4106		Nd_preop	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_preop[0,1,2,3		Unsigned32	
	2	Nd_preop[4,5,6,7		Unsigned32	
	3	Nd_preop[8,9,10,11		Unsigned32	
	4	Nd_preop[12,13,14,15		Unsigned32	

---

**Nd\_err**                        The table below presents the Object Entry 4107 (Nd\_err).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4107		Nd_err	ARRAY		
	0	Number of entries		Unsigned8	
	1	Nd_err[0,1,2,3		Unsigned32	
	2	Nd_err[4,5,6,7		Unsigned32	
	3	Nd_err[8,9,10,11		Unsigned32	
	4	Nd_err[12,13,14,15		Unsigned32	

---

**Node Error Count**                      The table below presents the Object Entry 4110 (Node Error Count).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4110		Node Error Count	ARRAY		
	0	Number of entries		Unsigned8	
	1	Number of the received emergency messages of node number 1		Unsigned8	
	...				
	127	Number of the received emergency messages of node number 127		Unsigned8	

**Error Code  
Specific Error  
Counters**

The table below presents the Object Entries 4111 to 4117 (Error Code Specific Error Counters).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4111		Generic_error_count (Code 10xxH)	VAR	Unsigned8	
4112		Device_hardware_error_count (Code 50xxH)	VAR	Unsigned8	
4113		Device_software_error_count (Code 60xxH)	VAR	Unsigned8	
4114		Communication_error_count (Code 81xxH)	VAR	Unsigned8	
4115		Protocol_error_count (Code 82xxH)	VAR	Unsigned8	
4116		External_error_count (Code 90xxH)	VAR	Unsigned8	
4117		Device_specific (Code FFxxH)	VAR	Unsigned8	

**Emergency  
History**

The table below presents the Object Entry 4118 (Emergency History).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4118		Emergency History	ARRAY		
	0	Number of entries		Unsigned8	
	1	Emergency history of node number 1		Domain	
	...				
	127	Emergency history of node number 127		Domain	

**input Process  
Image**

The table below presents the Object Entry 4200 (Input Process Image).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4200		Input Process Image	RECORD		
	0	Number of entries		Unsigned8	
	1	Current byte length		Unsigned16	Read only access Updated by the Master Manager

**Output Process Image**      The table below presents the Object Entry 4201 (Output Process Image).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4201		Output Process Image	RECORD		
	0	Number of entries		Unsigned8	
	1	Current byte length		Unsigned16	Read only access Updated by the Master Manager

**Additional Master Information**      The table below presents the Object Entry 4205 (Additional Master Information).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4205		Additional Master Information	RECORD		
	0	Number of entries		Unsigned8	ro
	1	Coupler (CPU) type		Unsigned8	rw
	2	CAN baudrate table index		Unsigned8	ro
	3	Highest used Node-ID		Unsigned8	ro
	4	Number of used RxPDOs		Unsigned16	ro
	5	Number of used TxPDOs		Unsigned16	ro
	6	Number of mapped objects PI input		Unsigned16	ro
	7	Number of mapped objects PI output		Unsigned16	ro
	8	Covered bytes by the concise DCF		Unsigned8	ro
	9	Byte size of the concise DCF buffer		Unsigned16	ro
	10	Configuration signature		Unsigned16	rw
	11	Control		Unsigned16	rw

Access type : ro (read only), rw (read / write)

---



**Additional Slave Assignment**      The table below presents the Object Entry 4250 (Additional Slave Assignment).

Index (Hex)	Sub-index	Description	Object type	Data type	Comments
4250		Additional Slave Assignment	ARRAY		
	0	Number of entries		Unsigned8	
	1	Boot behaviour for Node-ID 1		Unsigned8	
	...				
	127	Boot behaviour for Node-ID 127		Unsigned8	

Bit 0 = 0 : Bootup according DS-302

Bit 1 = 1 : Bootup do not overwrite config parameter



---

## Relation between PDOs and STB variables

A large gray square containing a bold black letter 'B'.

---

### STB island configuration

#### At a glance

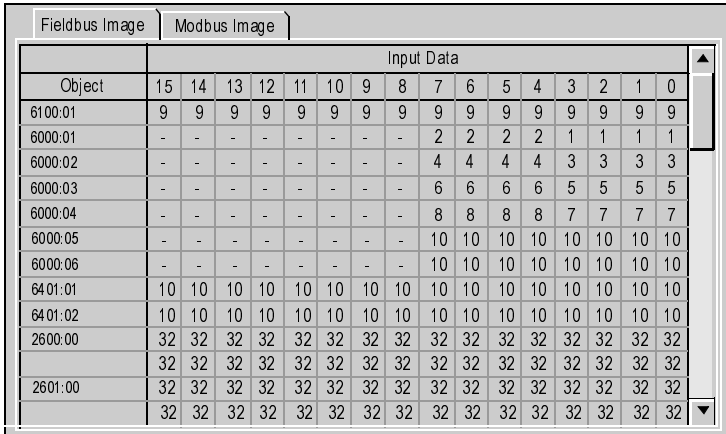
Using the COBid, it's possible to establish a link between PDOs and STB variables.

STB islands can be configured:

- using Advantys Configuration Software (STB NCO 2212),
  - using Unity Pro Software (STB NCO 2212 and NCO 1010).
-

### Configuration using Advantys Configuration Island

The procedure for configuring a STB island using Advantys Configuration Software is as follow. It only concerns the STB NCO 2212 module:

Step	Action
1	In Advantys Configuration Software (Version 2.2.0.2 or above), create a new island
2	Select the STBNCO2212 Network Interface Module
3	Select the modules which will be used in the application
4	<p>In the menu click on <b>Island</b> and on <b>I/O image overview</b></p>  <p>This window represents an I/O image overviev while offline. The variable indexes are the same as for Unity Pro Software. It allows to find the content of PDO quickly and easily.</p>
5	When the configuration is over, click on <b>File/Export</b> to export the island in DCF format, which will be imported in Unity Pro.

**Configuration  
using Unity Pro  
Software**

The procedure to configure a STB island using Unity Pro Software is as follow:

Step	Action																																																																									
1	In the Project browser , double-click on Configuration then 3:CANopen.																																																																									
2	In the CANopen window, double-click on the Advantys STB representation. The STB configuration window opens																																																																									
3	In Function zone, select Advanced. <div><div>Function: Advanced</div></div>																																																																									
4	Click on the PDO tab to see the PDO configuration, the variables and their topological addresses. <div><div><div><div>PDO</div><div>Error control</div><div>Config</div></div><div><div>Transmit (%)</div><table><tr><th>PDO</th><th>Tr. Ty...</th><th>Inhibi...</th><th>Even...</th><th>Symbol</th><th>Topo. Addr.</th><th>%M...</th><th>CO...</th><th>Index</th></tr><tr><td><div><div><div></div><div></div></div><div>PDO 1</div></div></td><td>255</td><td>0</td><td></td><td></td><td>%IW13.2I0.0...</td><td>%MW184</td><td>16#182</td><td>6000:01</td></tr><tr><td><div><div><div></div><div></div></div><div>Digital 8...</div></div></td><td></td><td></td><td></td><td></td><td>%IW13.2I0.0...</td><td>%MW185</td><td></td><td>6000:02</td></tr></table><div><div>Import DCF</div><div>Variables</div><div><input type="checkbox"/> Display only unmapped variables</div><table><tr><th>Parameter Name</th><th>Ind...</th></tr><tr><td>Island Diagnostics: ...</td><td>4000:00</td></tr><tr><td>Island Diagnostics: l...</td><td>4001:00</td></tr><tr><td>Configured Nodes 1...</td><td>4002:01</td></tr><tr><td>Configured Nodes 3...</td><td>4002:02</td></tr><tr><td>Configured Nodes 4...</td><td>4002:03</td></tr><tr><td>Configured Nodes 6...</td><td>4002:04</td></tr><tr><td>Configured Nodes 8...</td><td>4002:05</td></tr><tr><td>Configured Nodes 9...</td><td>4002:06</td></tr><tr><td>Configured Nodes 1...</td><td>4002:07</td></tr><tr><td>Configured Nodes 1...</td><td>4002:08</td></tr><tr><td>Optionnal Nodes 1...</td><td>4003:01</td></tr><tr><td>Optionnal Nodes 3...</td><td>4003:02</td></tr><tr><td>Optionnal Nodes 4...</td><td>4003:03</td></tr><tr><td>Optionnal Nodes 6...</td><td>4003:04</td></tr><tr><td>Optionnal Nodes 8...</td><td>4003:05</td></tr><tr><td>Optionnal Nodes 9...</td><td>4003:06</td></tr><tr><td>Optionnal Nodes 1...</td><td>4003:07</td></tr><tr><td>Optionnal Nodes 1...</td><td>4003:08</td></tr><tr><td>Nodes with Error 16...</td><td>4004:01</td></tr><tr><td>Nodes with Error 32...</td><td>4004:02</td></tr><tr><td>Nodes with Error 48...</td><td>4004:03</td></tr><tr><td>Nodes with Error 64...</td><td>4004:04</td></tr></table></div></div></div></div>	PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	%M...	CO...	Index	<div><div><div></div><div></div></div><div>PDO 1</div></div>	255	0			%IW13.2I0.0...	%MW184	16#182	6000:01	<div><div><div></div><div></div></div><div>Digital 8...</div></div>					%IW13.2I0.0...	%MW185		6000:02	Parameter Name	Ind...	Island Diagnostics: ...	4000:00	Island Diagnostics: l...	4001:00	Configured Nodes 1...	4002:01	Configured Nodes 3...	4002:02	Configured Nodes 4...	4002:03	Configured Nodes 6...	4002:04	Configured Nodes 8...	4002:05	Configured Nodes 9...	4002:06	Configured Nodes 1...	4002:07	Configured Nodes 1...	4002:08	Optionnal Nodes 1...	4003:01	Optionnal Nodes 3...	4003:02	Optionnal Nodes 4...	4003:03	Optionnal Nodes 6...	4003:04	Optionnal Nodes 8...	4003:05	Optionnal Nodes 9...	4003:06	Optionnal Nodes 1...	4003:07	Optionnal Nodes 1...	4003:08	Nodes with Error 16...	4004:01	Nodes with Error 32...	4004:02	Nodes with Error 48...	4004:03	Nodes with Error 64...	4004:04
PDO	Tr. Ty...	Inhibi...	Even...	Symbol	Topo. Addr.	%M...	CO...	Index																																																																		
<div><div><div></div><div></div></div><div>PDO 1</div></div>	255	0			%IW13.2I0.0...	%MW184	16#182	6000:01																																																																		
<div><div><div></div><div></div></div><div>Digital 8...</div></div>					%IW13.2I0.0...	%MW185		6000:02																																																																		
Parameter Name	Ind...																																																																									
Island Diagnostics: ...	4000:00																																																																									
Island Diagnostics: l...	4001:00																																																																									
Configured Nodes 1...	4002:01																																																																									
Configured Nodes 3...	4002:02																																																																									
Configured Nodes 4...	4002:03																																																																									
Configured Nodes 6...	4002:04																																																																									
Configured Nodes 8...	4002:05																																																																									
Configured Nodes 9...	4002:06																																																																									
Configured Nodes 1...	4002:07																																																																									
Configured Nodes 1...	4002:08																																																																									
Optionnal Nodes 1...	4003:01																																																																									
Optionnal Nodes 3...	4003:02																																																																									
Optionnal Nodes 4...	4003:03																																																																									
Optionnal Nodes 6...	4003:04																																																																									
Optionnal Nodes 8...	4003:05																																																																									
Optionnal Nodes 9...	4003:06																																																																									
Optionnal Nodes 1...	4003:07																																																																									
Optionnal Nodes 1...	4003:08																																																																									
Nodes with Error 16...	4004:01																																																																									
Nodes with Error 32...	4004:02																																																																									
Nodes with Error 48...	4004:03																																																																									
Nodes with Error 64...	4004:04																																																																									
5	On the right side of the window, there's the list of STB mapped or unmapped variables. The indexes are the same as Advantys Configuration Software. Variables can be found quickly and easily. Drag and drop the variables to the right PDO to configure the STB island.																																																																									



---

# Actions and transitions



---

## At a glance

**Subject of this chapter** This chapter contains the actions and the transitions used in the grafcet (See *Creating the program in SFC for managing the move sequence*, p. 192)

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

Topic	Page
Transitions	240
Actions	241

---

## Transitions

---

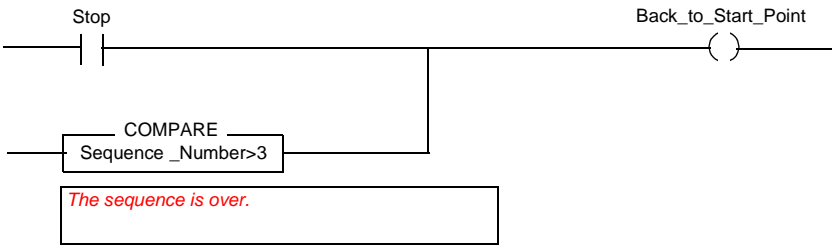
### At a glance

The next tasks, written in LD, are used in different transitions of the grafcet.

---

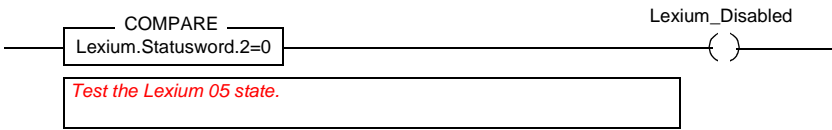
### Back\_to\_Start\_Point transition

The action associated to the **Back\_to\_Start\_Point** transition is as follows:



### Lexium\_Disabled transition

The action associated to the **Lexium\_Disabled** transition is as follows:





## Actions

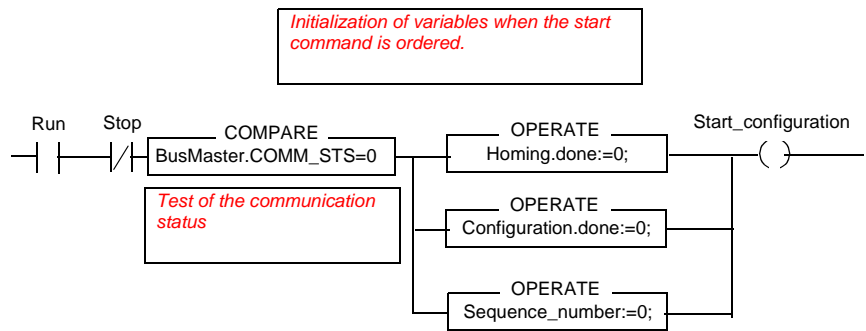
### At a glance

The next tasks, written in LD and ST are used in different steps of the grafcet.

**Note:** To use the following actions, in Tools/Project Setting/Languages extension, select Allow dynamic arrays and Directly represented array variables options.

### Init step

The action associated to the **Init** step is as follows:



### **Move\_to\_Next\_Position step**

Two actions are associated to the **Move\_to\_Next\_Position** step.

The first action is as follows:

```
(* Definition of the target position*)
CASE Sequence_number OF
    1: Lexium.Target_Position:=Position_B;
    2: Lexium.Target_Position:=Position_A;
    3: Lexium.Target_Position:=Position_C;
END_CASE;
IF (Sequence_number<4) AND NOT (Stop) THEN
    (* Start the new positionning *)
    New_SetPoint:=1;
    Ready_for_Stop:=0;
END IF;
```

The second action is as follows:

```
(*Incrementation before new move starts*)
INC(Sequence_Number);
```

**Note:** For the incrementation action, the qualifier must be positionned on P (rising edge).

### **Return\_to\_Start\_Point step**

The action associated to the **Return\_to\_Start\_Point** step is as follows:

```
(*Target Position Loading*)
Lexium.Target_Position:=0;
(*Start a new positioning*)
New_Setpoint:=1;
```

### **Disable\_Lexium**

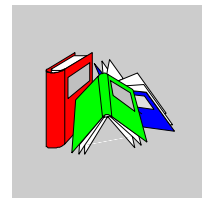
The action associated to the **Disable\_Lexium** step is as follows:

```
(*Lexium voltage disabling*)
Lexium.Controlword:=Lexium_disabling;
```

---

---

# Glossary



---

## A

**ADVANTYS**      Schneider CANopen Configuration tool for PLC islands.

---

## B

**BOOL**      Boolean.

---

## C

**CAN**      **Controller Area Network** : field bus originally developed for automobile applications and now used in many sectors.

**CiA**      **CAN in Automation** : international organization of users and manufacturers of CAN devices.

**COB**      **Communication Object**: transport unit on CANopen bus. A COB is identified by a unique identifier, which is coded on 11 bits, [0, 2047]. A COB contains a maximum of 8 data bytes. The transmission priority of a COB is given by its identifier. The weaker the identifier, the more the associated COB is priority.

**COB-ID**      **COB Identifier** : unique identifier of a COB on a CANopen network. The identifier determines the priority of a COB.

**CSDO** SDO Client

---

**D**

**DINT** **Double integer** : 32 bit word.

**Discrete Module** Tout Ou Rien.

**DS** **Draft Standard**: specifications document created by the CIA organization.

---

**E**

**EBOOL** Boolean with edge detection and forcing possibilities.

**EDM** **Multi-language Electronic Data Sheet** : extended version of EDS file. Extensions include European multilingual support as well as a description of physical characteristics of a device.

**EDS** **Electronic Data Sheet**: Description of a CANopen device profile description normalized by the DSP306 CiA specification.

**EMCY** **Emergency** : A trigger event, generated by an internal error/fault. This object is transmitted with each new error, since error codes are independent mechanisms.

**ETS** **Empty Terminal Support** : Additional information is stored in the PLC application for uploading.

---

**H**

**HEALTH** **bit from 1** : Mode functions correctly  
**bit from 0** :

- Bad configuration, or,
- Module configured but absent, or;
- module already configures, but with the same address as an existing module, or
- No Communication

---

**I**

<b>INT</b>	<b>Integer</b> : Integer 16 bit word.
<b>IODDT</b>	<b>Input/Output Derived Data Type</b>

---

**M**

<b>Mapping</b>	Transformation of data consigned in a special and different format.
----------------	---

---

**N**

<b>NIM</b>	<b>Network Interface Module</b> : Communication between the device and field bus.
<b>NMT</b>	<b>Network Management</b> : This is responsible for managing the execution, configuration and errors in a CAN network.

---

**P**

<b>PDO</b>	<b>Process Data Object</b> : object for data exchange between different elements is CAN open.
<b>PROCESS IMAGE</b>	Part of the system memory where the E/S values are stored from PDO exchanges on the CANopen bus. This section is managed by the CANopen stack. The inputs are copied in the user application memory at the start of each task cycle and the outputs at the end of each task cycle.

---

**R**

<b>REAL</b>	Real number.
-------------	--------------

---

**RPDO** Received PDO

---

**S**

**SDO** **Service Data Object**: peer to peer communication with access to Dictionary Object of a CANopen bus element.

**SSDO** SDO Server

**STB** **S**mall **T**erminal **B**lock.

**SYNC** Synchronisation Object

---

**T**

**TPDO** PDO Transmission

---

**U**

**UDINT** **Unsigned double integer** : Unsigned double integer

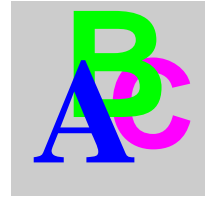
**UINT** **Unsigned integer** : Unsigned integer

---

---

# Index

---



## A

addressing  
topological, 100

## B

BMXP342010, 29  
BMXP342030, 29  
bus lengths, 23

## C

CANopen  
connectors, 32  
channel data structure for all modules  
IODDT, 139  
T\_GEN\_MOD, 163  
channel data structure for communication  
protocols  
T\_COM\_CO\_BMX, 132, 143  
T\_COM\_STS\_GEN, 132, 139  
COB-ID, 159  
configuring, 65  
steps of configuration, 62  
configuring the devices  
STB, 85  
TesyS U, 85

configuring the servodrives

ATV31, 85  
ATV61, 85  
ATV71, 85  
IclA, 85  
Lexium 05, 85

## D

debugging, 117  
diagnosing, 33  
diagnostics, 125

## E

emergency objects (EMCY), 159  
error codes, 159  
error control  
heartbeat, 79  
node guarding, 79  
event timer, 100

## I

inhibit time, 100

## N

NMT (network management), 79

## **P**

- parameter settings, 132
- PDO mapping, 104
- PDOs, 100
- performances, 63
- profile, 16
- programming, 99

## **Q**

- quick start, 165

## **R**

- READ\_VAR, 108

## **S**

- SDOs, 105
- standards, 25

## **T**

- T\_COM\_CO\_BMX, 143
- T\_COM\_CPP110, 133
- T\_COM\_STS\_GEN, 132, 133, 139
- T\_GEN\_MOD, 163
- transmission speeds, 23
- transmission type, 100

## **W**

- WRITE\_VAR, 108