Premium and Atrium using Unity Pro

CANopen Field Bus

User manual

04/2015

35008147.12

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents

Safety Information

Important Information

NOTICE

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

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

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

⚠ DANGER

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

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

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.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

At a Glance

Document Scope

This manual presents the CANopen communication on Premium and Atrium PLCs.

Validity Note

This documentation is valid for Unity Pro 10.0 or later.

Product Related Information

⚠ 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.

Chapter 1
General

Subject of this Chapter

This chapter describes the main technical characteristics for CANopen communication.

What Is in This Chapter?

This chapter contains the following topics:

Principles

Introduction

The CAN communication bus was originally developed for onboard automobile systems, and is now used in a wide range of areas, such as:

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

The strong points of the CAN system are:

- Its bus allocation system,
- Its error detection capability,
- The reliability of its 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.

Peer to Peer Communication

The communications on the bus operate on a peer to peer basis, whereby each device can, at any time, 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.

CAN Identifiers

The explicit exchanges of the CAN PDUs at link level use extended-format (29 bit) identifiers (CAN V2.0B standard).

11 bit identifiers (CAN V2.0A standard) may also be used for sending requests, but this type of identifier cannot be received.

General Architecture of the CANopen Field Bus

At a Glance

The CANopen architecture consists of:

- a bus master (TSX CPP 110 PCMCIA card),
- slave devices.

NOTE: It is also possible to connect several TSX CPP 110 PCMCIA cards on the bus, with one as master and all the others in **listen** mode. The cards in **listen** mode enable the Premium PLCs to which they are connected to know the status of the bus and the bus slaves at all times.

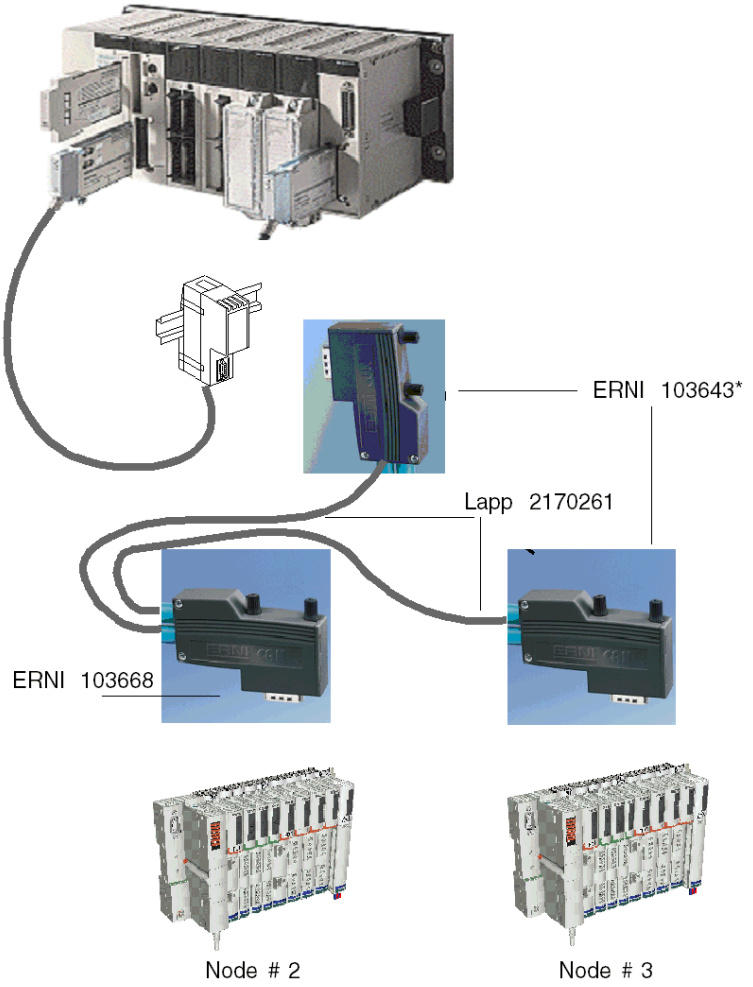
CANopen allows 128 devices (namely the bus master abd 127 remote slaves).

Transmission speed strictly depends on the bus length and the type of cable used.

NOTE: To configure a CANopen bus, use the Sycon software, version 2.8 or higher.

Example of a Bus

The figure below shows an example of a CANopen bus that will be described in detail at the end of this document in the example for implementing a CANopen bus.



*: integrated line termination impedance (120 Ohms).

For more details, consult the CANopen Hardware Setup Manual available on telemecanique.com

Transmission Speed and Cable Length

At a Glance

CANopen allows 127 devices (the bus master and 126 remote slaves). Transmission speed depends strictly on the type of used cable.

In the CAN protocol frame priority is managed by collision between dominant and recessive levels of the line. This collision must be resolved during transmission of a bit, which limits the signal propagation delay between 2 nodes.

The following tables specify the maximum trunk cable length based on the CANopen cable provided by Schneider Electric (TSXCANCA•••, TSXCANCB••• and TSXCANCD•••).

Maximum Cable Length

Consequently, the maximum distance between the 2 most distant nodes of a CAN bus depends on the speed and is provided in the following table:

Speed in bit/s	Maximum Cable Length
1 Mbit/s	20 m (65 ft)
800 kbit/s	40 m (131 ft)
500 kbit/s	100 m (328 ft)
250 kbit/s	250 m (820 ft)
125 kbit/s	500 m (1640 ft)
50 kbit/s	1000 m (3280 ft)
20 kbit/s	2500 m (8202 ft)
10 kbit/s	5000 m (16404 ft)

According to the Schneider Electric network strategy, the speeds 1 Mbit/s, 800 kbit/s, 500 kbit/s, 250 kbit/s and 125 kbit/s are recommended for automation solutions at machine and installation level.

NOTE: The maximum length assumes a reasonable device internal propagation delay and bit sampling point. Devices that present long internal propagation delays will effectively reduce the maximum cable length that could otherwise be realized.

The cable lengths of the above table may include a drop cable if it is at the physical end of the trunk cable.

Repeaters Reducing Cable Length

The above values specify the maximum cable length without any repeater. As repeaters add a propagation delay in the bus, this delay reduces the maximum length of the bus. A propagation delay of 5 ns leads to a length reduction of 1 m (3 ft).

Example. A repeater with a propagation delay of 150 ns reduces the maximum cable length by 30 m (98 ft).

Maximum Cable Length vs. Number of Nodes

In addition to the length limitations based on the transmission speed, the maximum cable length is also influenced by the load resistance.

In any case, the maximum number of nodes that may be connected on the same segment is restricted to 64. To connect more nodes to 1 segment, use a repeater.

The table below shows the influence by the number of nodes on the cable length:

Number of Nodes	Maximum Cable Length
2	229 m (751.31 ft)
16	210 m (688.97 ft)
32	195 m (639.76 ft)
64	170 m (557.74 ft)

Electrical Isolation of CANopen Devices

In documents about CANopen you will often find the value of 40 m (131 ft) maximum value at a transmission speed of 1 Mbit/s. This length is calculated without electrical isolation as used in the Schneider Electric CANopen devices.

With such electrical isolation the minimum network length calculated is 4 m (13 ft) at a transmission speed of 1 Mbit/s. However, the experience shows that 20 m (65 ft) are the practical length that could be shortened by drops or other influences.

Drop Cable Limitations

Overview

A drop cable creates a signal reflection on the transmission line characteristic of the trunk cable. In order to limit reflections, drop cables should be as short as possible.

Maximum Drop Cable Length

Respect the values listed in the following table:

Transmission Rate	Lmax	ΣLmax	TAP Distance	ΣLGmax
1 Mbit/s	0.3 m (0.98 ft)	0.6 m (0.98 ft)		1.5 m (4.92 ft)
800 kbit/s	3 m (9.84 ft)	6 m (19.68 ft)	3.6 m (11.81 ft)	15 m (49.21 ft)
500 kbit/s	5 m (16.4 ft)	10 m (32.8 ft)	6 m (19.68 ft)	30 m (98.42 ft)
250 kbit/s	5 m (16.4 ft)	10 m (32.8 ft)	6 m (19.68 ft)	60 m (196.84 ft)
125 kbit/s	5 m (16.4 ft)	10 m (32.8 ft)	6 m (19.68 ft)	120 m (393.69 ft)
50 kbit/s	60 m (196.84 ft)	120 m (393.69 ft)	72 m (236.21 ft)	300 m (984.24 ft)
20 kbit/s	150 m (492.12 ft)	300 m (984.24 ft)	180 m (590.54 ft)	750 m (2460.62 ft)
10 kbit/s	300 m (984.24 ft)	600 m (1968.49 ft)	360 m (1181.09 ft)	1500 m (4921.24 ft)

Lmax is the maximum length of 1 drop cable.

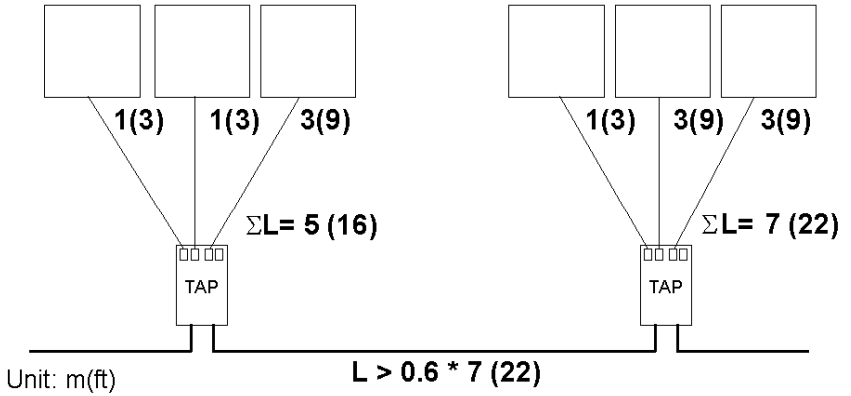
Σ**Lmax** is the maximum value of the sum of drop cables on the same TAP.

TAP distance is the minimum distance necessary between 2 TAPs, can be calculated for each TAP (must be greater than 60% of the largest of the 2 ΣLmax).

Σ**LGmax** is the maximum value of the sum of drop cables on the network.

Calculation Example

The figure below provides an example of a TAP distance calculation with 2 junction boxes and 6 devices:



The TAP distance in the above example is calculated as follows:

Step	Description	Result
1	Calculating the sum of lengths of drop cables for each tap junction.	5 m (16 ft) and 7 m (22 ft)
2	Keeping the longest length.	7 m (22 ft)
3	Calculating the minimum cable length between the 2 TAPs.	60% of 7 m (22 ft)

Respect the TAP distance even if a device is in between.

Installation Phase Overview

Introduction

The software installation of the application-specific modules is carried out from the various Unity Pro editors:

- in offline mode
- in online mode

If you do not have a processor to connect to, Unity Pro allows you to carry out an initial test using the simulator. In this case the installation *(see page 20)* is different.

The following order of installation phases is recommended but it is possible to change the order of certain phases (for example, starting with the configuration phase).

Installation Phases with Processor

The following table shows the various phases of installation with the processor:

Phase	Description	Mode
Declaration of variables	Declaration of IODDT-type variables for the application-specific modules and variables of the project.	Offline (1)
Programming	Project programming.	Offline (1)
Configuration	Declaration of modules.	Offline
	Module channel configuration.	
	Entry of configuration parameters.	
Association	Association of IODDTs with the channels configured (variable editor).	Offline (1)
Generation	Project generation (analysis and editing of links).	Offline
Transfer	Transfer project to PLC.	Online
Adjustment/Debugging	Project debugging from debug screens, animation tables.	Online
	Modifying the program and adjustment parameters.	
Documentation	Building documentation file and printing miscellaneous information relating to the project.	Online (1)
Operation/Diagnostic	Displaying miscellaneous information necessary for supervisory control of the project.	Online
	Diagnostic of project and modules.	
Key:		
(1)	These various phases can also be performed in the other mode.	

Implementation Phases with Simulator

The following table shows the various phases of installation with the simulator.

Phase	Description	Mode
Declaration of variables	Declaration of IODDT-type variables for the application-specific modules and variables of the project.	Offline (1)
Programming	Project programming.	Offline (1)
Configuration	Declaration of modules.	Offline
	Module channel configuration.	
	Entry of configuration parameters.	
Association	Association of IODDTs with the modules configured (variable editor).	Offline (1)
Generation	Project generation (analysis and editing of links).	Offline
Transfer	Transfer project to simulator.	Online
Simulation	Program simulation without inputs/outputs.	Online
Adjustment/Debugging	Project debugging from debug screens, animation tables.	Online
	Modifying the program and adjustment parameters.	
Key:		
(1)	These various phases can also be performed in the other mode.	

NOTE: The simulator is only used for the discrete or analog modules.

Chapter 2
Overview of the PCMCIA TSX CPP 110 card

Subject of this Chapter

This chapter describes the main technical characteristics of TSX CPP 110 PCMCIA cards.

What Is in This Chapter?

This chapter contains the following sections:

Section 2.1
Description of the TSX CPP 110 card

Subject of this Section

This section describes the physical characteristics of the PCMCIA TSX CPP 110 card and its connections.

What Is in This Section?

This section contains the following topics:

Information Concerning the TSX CPP 110 Card

At a Glance

The CANopen communication card TSX CPP 110 is used to implement a CANopen architecture. This card is master of the bus and enables the connection of devices in compliance with the CANopen standard:

- Implicit exchange of **P**rocess **D**ata **O**bjects using `%MW` words.
- Explicit exchange of **S**ervice **D**ata **O**bjects via `READ_VAR` and `WRITE_VAR` function blocks.
- Compatibility with standardized device profiles and communication on CANopen (2.0A and 2.0B).

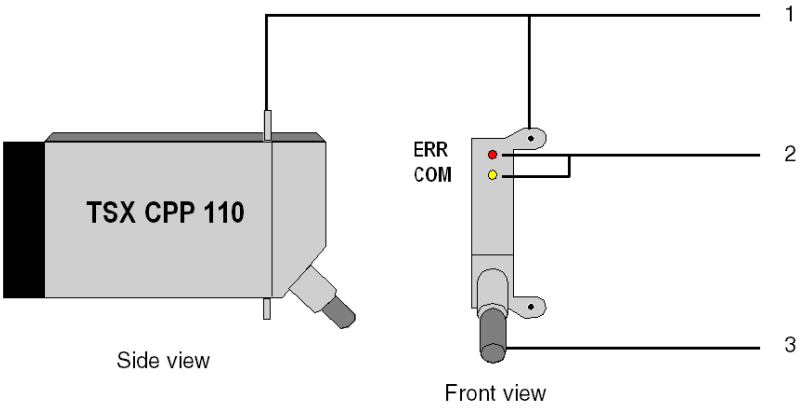
NOTE: The TSX CPP 110 card authorizes a maximum configuration of 64 KB.

Physical Description

The TSX CPP 110 card is a type III PCMCIA card which is inserted into the PCMCIA communication slot on the processor.

Type III CANopen PCMCIA cards (TSX CPP 110) can be used in the PCMCIA slots of all PLCs that you can program using Unity Pro software.

This module is made up of the following elements:



Description

This table describes the elements in the previous figure.

Number	Description
1	Fixing brackets, on the top and bottom of the card, which are used to attach it to the processor.
2	LEDs, which are used to diagnose the operations of the communication card *(see page 67)*.
3	Bus cable, this 50 cm cable is equipped at its end with a CANopen industrial TAP.

Mounting the TSX CPP 110 Card

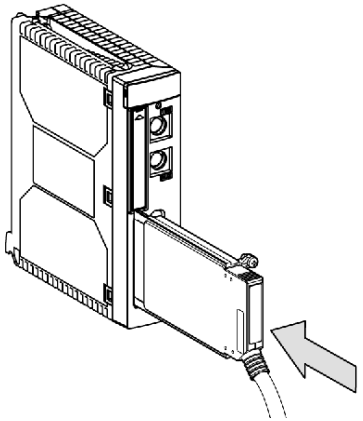
TSX CPP 110 PCMCIA Card

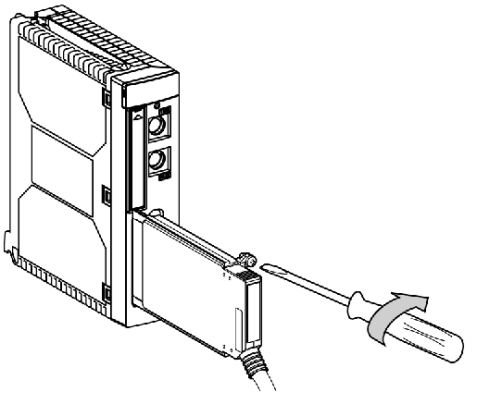
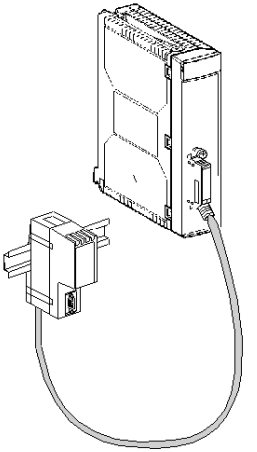
The PCMCIA TSX CPP 110 card with its industrial TAP operate the link between the Premium CPU and a CANopen network.

NOTE: The Modbus TSX SCY 2160• communication module cannot be used.

⚠ CAUTION
PCMCIA CARD DAMAGE To insert or disengage a PCMCIA card the PLC must be switched off. **Failure to follow these instructions can result in injury or equipment damage.**

The following table describes the procedure for installing a TSX CPP 110 card.

Step	Action	Illustration
1	Switch off the PLC.	
2	Insert the CANopen PCMCIA type III card in the PCMCIA slot of the Premium CPU.	

Step	Action	Illustration
3	Screw the card in securely to ensure it operates correctly.	
4	Fix the TAP on a DIN rail.	
5	Switch the PLC on again.	

Section 2.2
Technical Specifications

Subject of this Section

This section describes the main technical characteristics of the TSX CPP 110 card.

What Is in This Section?

This section contains the following topics:

Standards and Characteristics

Standards

The TSX CPP 110 communication card complies with the following international standards:

International Standards	ISO IS 11898, CAN High Speed Transceiver and Data Link Layer
US Standards	UL508
Radiation Standards	EC label, FCC-B (50082-1)

Electrical Characteristics

- Logical V DC supply: 5 V provided by the rack supply
- Power Consumption: 3W

Environment Characteristics

- Storage Temperature: -25 °C to 70 °C
- Operating Temperature: 0° C to 70 °C
- Storage Hydrometry: 30% to 95% without condensation
- Operating Hydrometry: 5% to 95% without condensation

CANopen Characteristics

Standards

The TSX CPP 110 communication card is compliant with the standard DS301 V4.01.

Specific Features

- The user may select a PDO content mapping compliant with the standard DS301 V4.01.
- The TSX CPP 110 card supports the "heartbeat" function (DS 301V4.01).
- The TSX CPP 110 card is normally the network management master (NMT_MASTER) on the bus (this function can be disabled via SyCon).
- The TSX CPP 110 card normally produces the synchronization (SYNC) variable (this function can be disabled via SyCon).
- The node ID of the TSX CPP 110 card may not be used for data transfer. It is only used for the "Heartbeat" function.

Processors Supporting the TSX CPP 110 Card

At a Glance

All Premium and Atrium processors support the CANopen PCMCIA card.

The card is implemented using Unity Pro software.

The general configuration of a CANopen bus is defined using the Sycon software, version 2.8 or higher (TLX LFBCM).

NOTE: It is mandatory for the PCMCIA card to be installed in the slot located in the processor module. As a result, only one CANopen bus is available for each PLC CPU.

Types of Processors and Capacities

The following table gives details of the processors supporting the TSX CPP 110 CANopen PCMCIA card, and their maximum storage capacities.

Processor	Maximum size of TSX CPP 110 configuration data located in the processor(1)	Maximum size of input/output data for the CANopen node configuration	
		MAST task	FAST task
TSX P57 0•• TSX P57 1•• TSX P57 1•••	12 Kb	384 %MW (192+192)	48 %MW (24+24)
TSX P57 2•• TSX P57 2••• TSX PCI 57 204	16 Kb	512 %MW (256+256)	64 %MW (32+32)
TSX P57 3•• TSX P57 3••• TSX PCI 57 354	32 Kb	1024 %MW (512+512)	128 %MW (64+64)
TSX P57 4•• TSX P57 4••• TSX P57 5•• TSX P57 5••• TSX P57 6•••	64 Kb	3584 %MW (1792+1792)	512 %MW (2) (256+256)
Key	(1): This maximum size can be exceeded if you load the configuration into the card using the Sycon software *(see page 43)*. The maximum size of configuration data authorized by the Sycon software is 64 Kb. (2): The maximum size of input/output data for the CANopen node configuration (FAST task) is 1024 %MW (512+512) for TSX P57 554, TSX P57 5634 and TSX P57 6634 with V3.0 firmware version.		

NOTE: The configuration fill rate is given in the word %KWm.1.2 *(see page 98)*.

Chapter 3
Software Setup

Subject of this Chapter

This chapter describes the different possibilities for configuration, supervisory control and diagnostics in a CANopen application.

What Is in This Chapter?

This chapter contains the following sections:

Section 3.1
General

Subject of this Section

This section describes the software installation of a TSX CPP 110 PCMCIA card.

What Is in This Section?

This section contains the following topics:

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 it is to be integrated (rack, supply, processor, modules or devices, etc.) then ensure the necessary software is implemented.

Its software implementation shall be performed from the different Unity Pro editors:

- Either in offline mode,
- Or in online mode: here, modifications are limited to certain parameters.

The bus is configured using Sycon software.

Implementation Principle

The following table shows the different implementation phases.

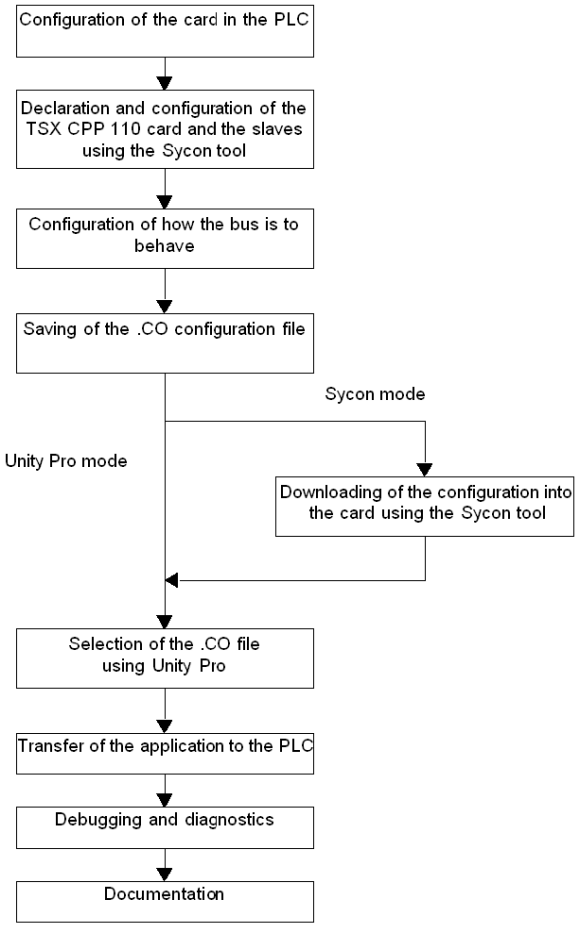
Mode	Phase	Description
Offline	Declaration of the TSX CPP 110 PCMCIA card	The card must be installed in the type III PCMCIA slot of the processor.
	Configuration	• Configuration parameterization. • Declaration of the bus configuration by the **Sycon** software and generation of the **.CO* configuration file. • Selection of the **.CO* configuration file using Unity Pro.
Offline or online	Symbolization	Symbolization of the variables associated with the CANopen card.
	Programming	Programming the specific functions: • Bit objects and associated words, • Specific instructions.
Online	Transfer	Transferring the application to the PLC A transfer of the application to the PLC or cold start of the application result starts and configures the TSX CPP 110 card.
	Debug Diagnostics	Different resources are available for debugging the application, controlling inputs/outputs and diagnosing faults: • Language objects or IODDTs, • The Unity Pro debugging screen, • Signaling by LED.
Offline or online	Documentation	Printing the various information relating to the configuration of the TSX CPP 110 card.

NOTE: The order shown above is for information purposes only. Unity Pro software makes it possible to use the editors in the order you wish, and interactively (however, the data or program editor cannot be used without having first performed the configuration).

Implementation Method

Overview

The following flowchart shows the implementation methodology for a TSX CPP 110 card.



Recommenda-tions

When the configuration of the CANopen bus is too large to allow the change to Unity Pro mode, you should carry out the following checks:

- See if you can remain in Unity Pro mode using a more powerful Premium processor *(see page 30)*.
- If it is not possible to select a more powerful processor, change to Sycon mode.

CANopen Topological Addressing

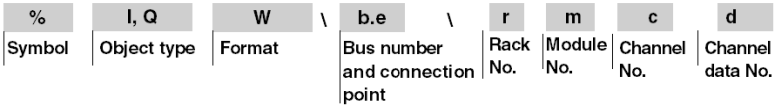
At a Glance

Topological addressing (which depends on the geographical position of the element to be addressed) is available on CANopen buses.

However, this differs slightly from Fipio topological addressing.

Illustration

Addressing is defined in the following way:



Syntax

The table below describes the different elements that make up addressing.

Family	Element	Values	Meaning
Symbol	%	-	Indicates an IEC object
Object type	I Q	- -	Input objects. Output objects. This information is exchanged automatically at each cycle of the task with which the CANopen card is associated (MAST or FAST).
Format (size)	W	16 bit	16 bit WORD-type word.
Module/channel address and connection point	b	3 to 999	Bus number.
	e	1 to 127	Connection point number (CANopen slave number).
Rack No.	r	0	Virtual rack number, always 0.
Module No.	m	0	Virtual module number, always 0.
Channel No.	c	0	Virtual channel number, always 0.
Channel data No.	d	0 to 59	Slave data number. Maximum 59, as a slave can only have a maximum of 60 input and output words (the sum of all input/output words must be less than 60).

Section 3.2
Configuration

Subject of this Section

This section describes the configuration of a TSX CPP 110 PCMCIA card.

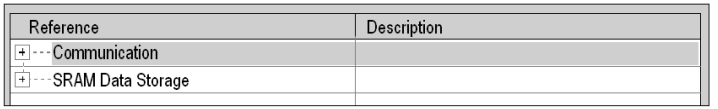
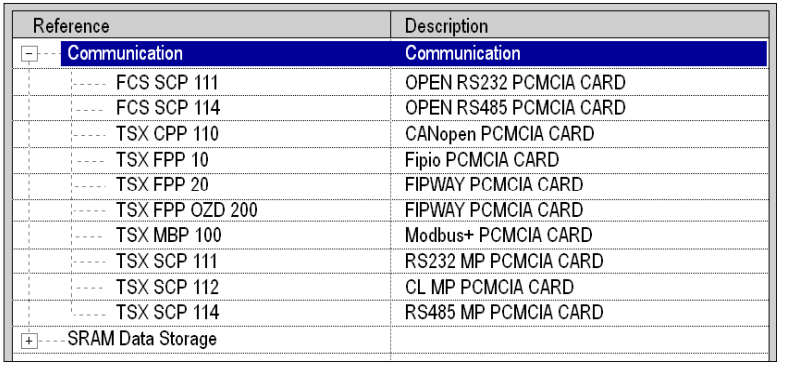
What Is in This Section?

This section contains the following topics:

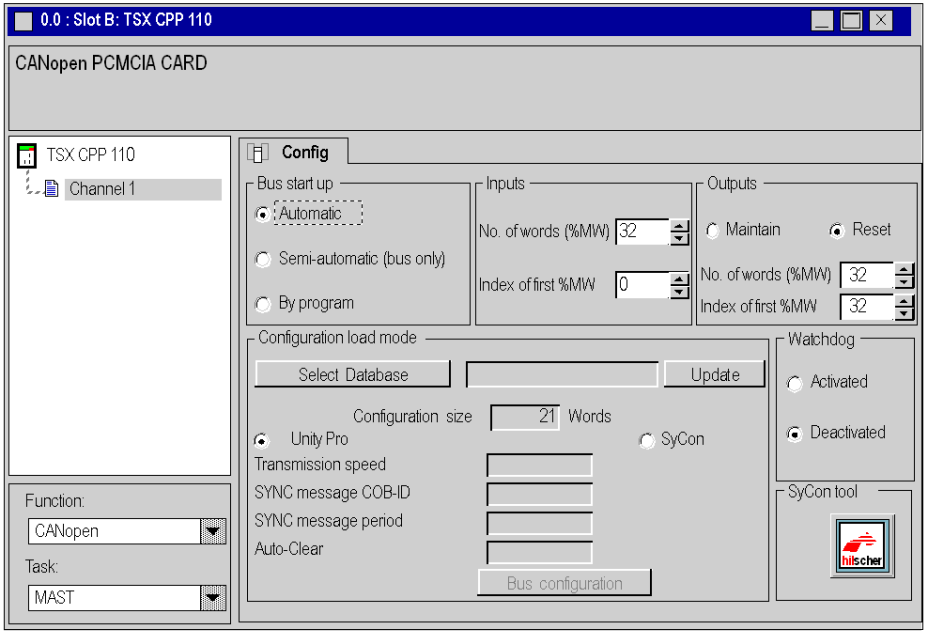
How to Access the Configuration Screen

Procedure

This operation is used to declare a TSX CPP 110 card in the processor's type III PCMCIA slot. The example below describes the procedure.

Step	Action
1	Open the hardware configuration editor from within the application browser.
2	Double-click the PCMCIA card slot positioned at the bottom of the processor. **Result:**the following list appears. 
3	Open up the **Communication** line by clicking this line **+** to obtain a list of communication cards available for this slot. **Result:** the following list appears. 
4	Select the TSX CPP 110 card and then confirm with **OK**. **Result**: the software displays the new X-Bus configuration editor, and you can now access the PCMCIA card configuration screen.

Step	Action
5	Double-click the PCMCIA card slot to obtain the PCMCIA card configuration window. **Result:**



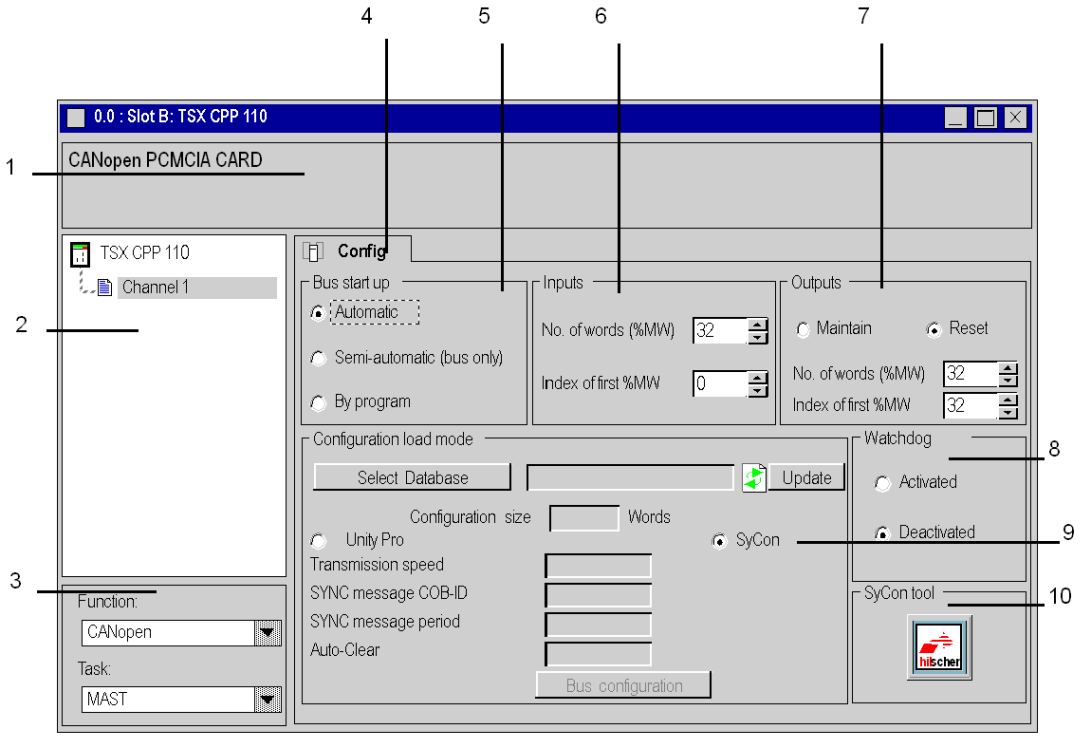
Configuration Screen of the TSX CPP 110 PCMCIA Card

At a Glance

This screen is used to declare the communication channel and to configure the parameters for a CANopen link.

Illustration

The screen dedicated to CANopen communication is as follows:



Elements and Functions

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

Zone	Number	Function
Module	1	This field comprises the abbreviated title of the PCMCIA card.
Channel	2	This zone allows you to select the communication channel to be configured. Click on the card number to display the tabs: ● **Description** which shows the characteristics of the communication card. ● **I/O Objects** *(see Unity Pro, Operating Modes)* which is used to presymbolize the input/output objects. ● **Fault** which shows the card faults (only accessible in online mode).

General parameters	3	This zone is composed of:
		● The communication function choice, in this case CANopen,
		● A drop-down list allowing the CANopen bus to be associated with a task of the application. This drop-down list is made up of three options that set the rate of update for the storage areas associated with the I/O:
		  ● MAST: MAST task rate
		  ● FAST: FAST task rate

Tab	4	The tab in the foreground indicates the type of screen displayed. In our case it is the configuration screen.
Config	5	This zone is used to select how the bus is to behave at start-up.
	6	This zone is used to configure the address (PLC internal memory) to which inputs from the CANopen devices will periodically be copied.
	7	This zone is used to configure the fallback mode for bus device outputs as well as the address (PLC internal memory) where the outputs from CANopen devices will periodically be read.
	8	This zone is used to activate or deactivate the CANopen bus watchdog. The watchdog is activated by default. It is triggered when the PCMCIA card can no longer manage the bus correctly. When it is triggered, it makes all the slaves' outputs change to zero.
	9	This zone is used to configure the bus: ● Selection of the Sycon configuration file (*.CO) *(see page 48)* ● Unity Pro or Sycon Configuration *(see page 43)* **NOTE:** Please, do not select **Sycon** radio button, this option is no longer available. Select only the **Unity Pro** button.
	10	This button is used to start the **Sycon** software, if it is installed on the PC.

⚠ WARNING

UNEXPECTED APPLICATION BEHAVIOR

Before deactivating the watchdog, ensure that, if the PCMCIA card does not manage the CANOpen bus, then the devices behavior remain acceptable.

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

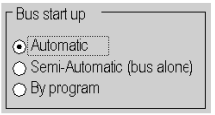
Description of the I/O Data and the Behavior of the Bus on Start-Up

At a Glance

The configuration screen allows you to configure the bus behavior at start-up as well as the inputs and outputs of slave devices on the bus.

Bus Start-Up

This figure illustrates the configuration zone of the bus start-up.



The bus start-up can be performed in three ways:

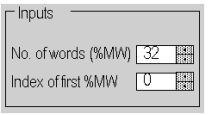
- **Automatic**: the bus configuration, communication management and the updating of slave I/Os are activated at start-up, without application intervention.
- **Semi-Automatic (bus alone)**: the bus configuration and communication management are activated at start-up but management of I/Os must be confirmed by the application using the corresponding language objects *(see page 93)*.
- **By program**: the bus start-up should be fully managed by the application using the corresponding language objects *(see page 93)*.

NOTE: in **Automatic** and **Semi-automatic** mode, if the **"Stop bus when slave watchdog is triggered"**, is activated, then in case of a fault you must perform a cold start of the PLC application in order to restart the CANopen bus.

NOTE: whatever the startup mode configured, the option **"Stop bus when slave watchdog is triggered"** option stops the bus when a slave disappears. If a fault occurs on a slave before the bus starts, the startup is performed nevertheless.

Inputs

This figure illustrates the input configuration zone.



To configure the inputs of the slaves on the bus, it is necessary to indicate a memory zone to which they will be copied periodically. To define this zone, indicate:

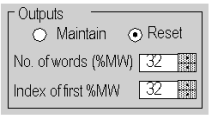
- **A number of words**: this is the number of input words corresponding to the size of input data configured by the **Sycon** software.
- **Address of the first %MW**: this is the address of the first word in the input memory zone.

NOTE: The %MW words contain the input values of the slaves on the bus. When the Unity Pro load mode is chosen, it is possible to recognize words associated with slaves by pressing the **Bus configuration** button on the configuration screen. In Sycon load mode, only the Sycon software allows you to recognize slaves associated with %MW words. These %MW words are used directly by the application as inputs.

NOTE: As for Fipio bus, if the %SW8 word is used (%SW8.0 bit set to 1 for the master task and %SW8.1 bit set to 1 for the fast task), the input acquisition phase of the bus devices is inhibited. The values of these inputs remain in the status prior to the setting to 1 of the bit.

Outputs

This figure illustrates the output configuration zone.



To configure the outputs, it is necessary to indicate, as for the inputs, the word table that will contain the value of the bus outputs, but also the type of fallback needed when there is a slave fault:

- Maintain,
- Reset.

NOTE: The %MW words contain output values of the slaves on the bus. When the Unity Pro load mode is chosen, it is possible to recognize words associated with slaves by pressing the **Bus configuration** button on the configuration screen. In Sycon load mode, only the Sycon software allows you to recognize slaves associated with %MW words. These %MW words are used directly by the application as outputs.

NOTE: The word tables are found in the PLC internal memory. Any crossover between these two areas is prohibited.

NOTE: If the number of input or output words are different to those determined in the configuration *(see page 43)* file (file name **.CO*), the Unity Pro software signals this when the configuration is confirmed.

NOTE: The maximum authorized size of the memory zone reserved for I/Os depends on the processor type and associated task *(see page 30)*.

NOTE: As for Fipio bus, if the %SW9 word is used (%SW9.0 bit set to 1 for the master task and %SW9.1 bit set to 1 for the fast task), the outputs of the devices on the bus are maintained in the state they were in before the bit was set to 1.

How to Select the Configuration Loading Mode

At a Glance

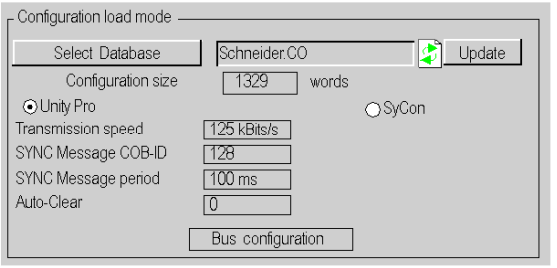
The configuration screen can be used to select the configuration loading mode. The two possible scenarios are as follows:

- Loading by the Unity Pro software,
- Loading by the Sycon software.

For each scenario, it is vital to select the configuration database *(see page 48)* created with the Sycon software.

Illustration

The following figure shows the area of the configuration screen used to select the configuration loading mode.



Description

The following table shows the different possible choices.

Zone	Description
Select Database	This area is used to select the database that corresponds to the configuration of the bus managed by the PCMCIA TSX CPP 110 card. This configuration is performed using the Sycon software, which generates the **.CO* file that must be selected *(see page 48)*.
Update	When you click this button, the selected **.CO* file is reloaded and evaluated. **Note**: This action must be performed after each modification Sycon makes to the selected **.CO* file.
Unity Pro	When you select this button, the configuration of the bus is loaded with the PLC application. When the application is too large (memory size greater than that authorized for the processor), the Unity Pro software does not authorize this selection, and you must change the processor.
Sycon	This option is no longer available, select only the **Unity Pro** button.

Zone	Description
Transmission speed	When the Unity Pro loading mode has been selected, this area displays the transmission speed on the bus defined in Sycon.
SYNC Message COB-ID	When the Unity Pro loading mode has been selected, this area displays the SYNC Message COB-ID selected in Sycon.
SYNC Message period	When the Unity Pro loading mode has been selected, this area displays the period of the bus defined in Sycon.
Auto-Clear	When the Unity Pro loading mode has been selected, this area displays the **Auto-clear on** or **Auto-clear off** mode selected in Sycon.
Bus configuration	When the Unity Pro loading mode has been selected, this button can be used to access the configuration of the slaves on the bus.

How to Load a Configuration using X-Way

General

When the CANopen bus is configured in load mode by Sycon, it is possible to download the configuration onto the TSX CPP 110 card using the X-Way driver.

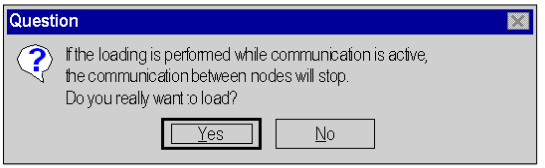
This download can be performed over an Ethernet network, or simply on a Uni-Telway bus.

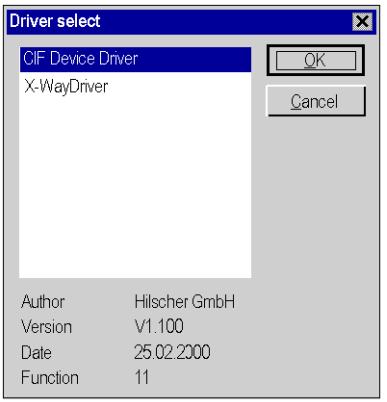
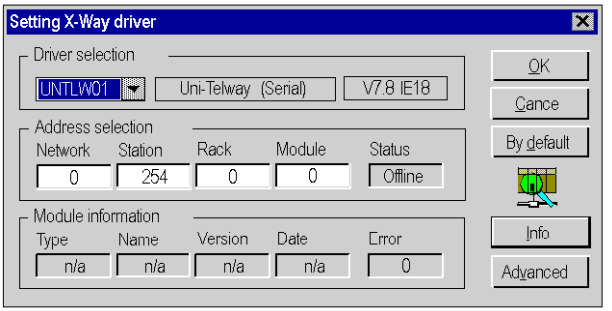
NOTE: MAKE SURE you always put the PLC in STOP mode during the downloading process.

NOTE: When the choice of X-Way communication driver has been validated, you must exit the Sycon software to change the driver.

Procedure

This table describes the steps to be carried out to load the configuration of a CANopen card using the X-Way communication drivers.

Step	Action
1	Connect to the PLC, which contains the TSX CPP 110 card with the help of the Unity Pro software.
2	Switch this PLC to STOP mode.
3	Launch the Sycon software.
4	Load or create the desired configuration with the help of the Sycon software.
5	Select the command **Online** →**Load**. **Result**: a message appears, which indicates that while the configuration is loading, communication between the slaves will be stopped. 

Step	Action
6	Click on **Yes** to indicate that you accept this halt in inter-slave communication. **Result**: a selection window for the **X-Way** or **CIF** driver appears. 
7	Select the **X-Way** driver then click **OK**. **Result**: the **Setting X-Way driver** window appears. 
8	Select the required driver (Uni-Telway, XIP etc.) from the **Driver selection** group box.
9	Enter the PLC address (Network, Station, Module, Rack) then click on **OK** to start the loading process. **Result**: during loading, a window indicates the progress of the data transfer. When the transfer is complete, this window disappears and makes way for the main bus configuration screen.

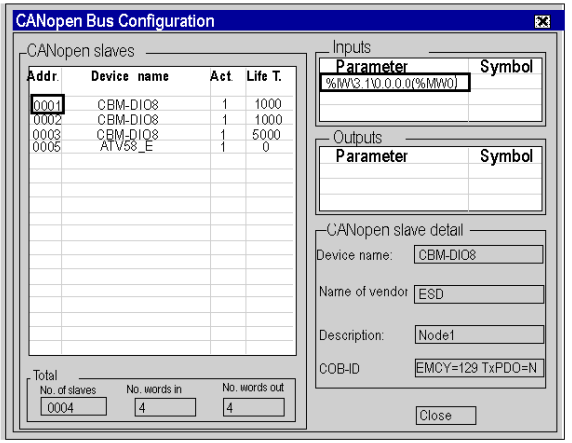
Slave Configuration Screen

At a Glance

With Unity Pro software, it is possible to access bus slave configuration. The information on the screen is almost identical to the debugging screen information *(see page 64)*.

Illustration

The following diagram presents the screen which displays slave configuration.



Operation

- Click on a slave from the list of CANopen slaves.
- The input and output slave parameters appear in the **Inputs** and **Outputs** areas. A clearer explanation is given with the display of the CANopen topological address *(see page 35)* and reserved memory area in the PLC.
- Information about slave appears in the CANopen slave detail zone.

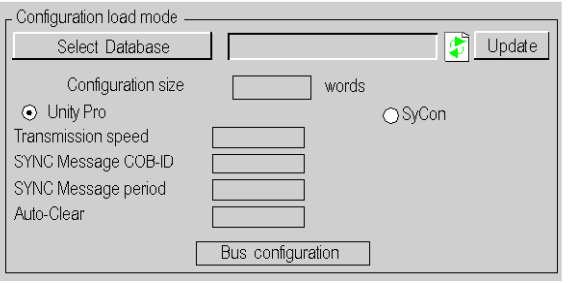
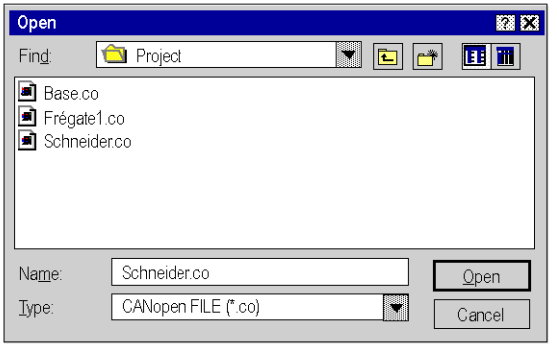
How to Configure a CANopen PCMCIA Card

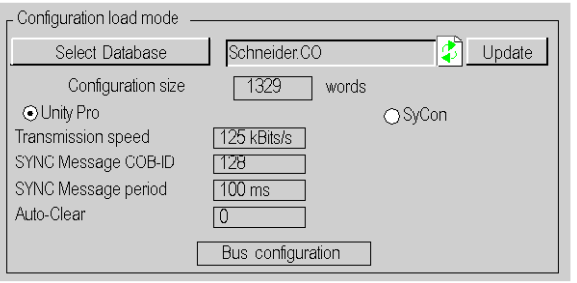
At a Glance

In order to configure a TSX CPP 110 card, some procedures are essential or require special explanation. Details are given in the following procedures.

How to Select a Configuration File

This table describes the steps in the procedure to select a CANopen configuration.

Step	Action
1	Click **Select Database**:  **Result**: a screen like this appears: 

Step	Action
2	Select the required *.*CO* file and then click on **OK**. **Result**: if the number of words reserved for inputs and outputs corresponds to the selected configuration, the configuration appears in the Unity Pro configuration screen.  Otherwise the incorrect values appear in red in the configuration screen and it is impossible to go to the bus configuration until these values have been rectified. When the *.CO* file passes the maximum capacity for configuration data *(see page 30)*, an error message appears. You must then use a more powerful Premium processor.

Procedure for Configuring a CANopen Card

The following table describes the procedure for configuring a CANopen TSX CPP 110 PCMCIA card.

Step	Action
1	Select the type of bus start-up.
2	Click on the **Sycon Tool** button in order to start the **Sycon** configuration software.
3	Using the **Sycon** software, configure your CANopen bus according to the devices planned for the bus.
4	Save the configuration in a file with the .CO extension.
5	Go back to Unity Pro.
6	Select a configuration file *(see page 48)*.
7	Reserve the PLC memory zones to be associated with the inputs and outputs.
8	Select the **Unity Pro** radio button.
9	Click on the Watchdog **Enabled** button.
10	Confirm the configuration. **Result**: the software tells you the number of words needed on input and output to configure the selected bus. If the specified sizes are not the optimum sizes, you can adjust them.

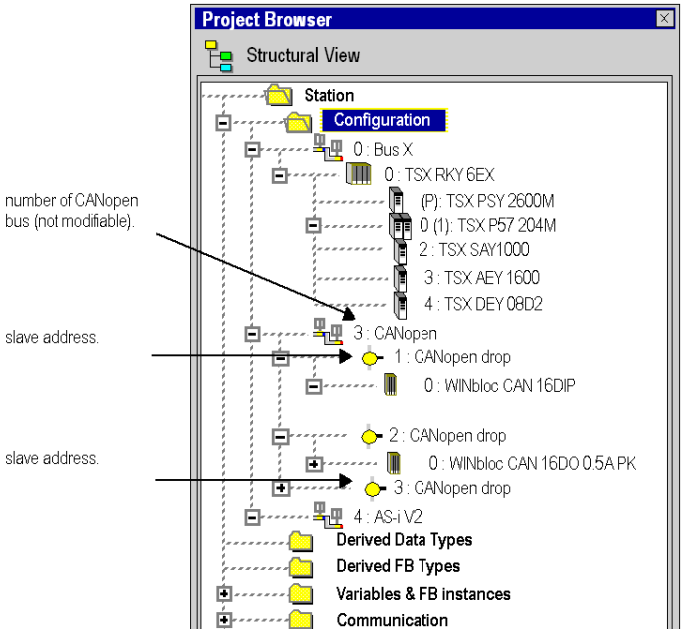
CANopen Bus Displayed in the Project Browser

At a Glance

When you declare a TSX CPP 110 card in a PLC processor, the CANopen bus is represented in the *Configuration* directory of the project browser. The number of the CANopen bus is calculated automatically by Unity Pro. This value cannot be modified.

After having loaded the **.CO* configuration file created with the Sycon software and validated the configuration, the CANopen slaves appear on the CANopen bus of the project browser. Each slave appears with its address number. By displaying the CANopen bus and slaves, you can view their topological addressing.

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



Configuration File of the TSX CPP 110 PCMCIA Card

At a Glance

From the Processor module, a folder describing the configuration of the application for the TSX CPP 110 PCMCIA card is available in the documentation editor.

Illustration

It is presented as follows:

	TSX P57 354M [RACK 0 POSITION 0]		
Module Identification			
Product Reference:	TSX P57 354M	**Designation:**	TSX P57 354M PROCESSOR
Address:	000	**Symbol:**	
Channel Parameters: 0			
Task/Channel Assignment:	MAST		
Type of Channel:	Terminal Port	Channel Symbol:	
Application-Specific Function:	Uni-Telway LINK	Channel Symbol:	
Transmission Speed	19200 bits/s	Timeout:	30 ms
Type of Module:	Master	Parity:	odd
Number of Slaves:	8		
Channel Parameters: 1			
Task/Channel Assignment:	MAST		
Type of Submodule:	TSX CPP 110 CANopen PCMCIA CARD		
Type of Channel:	PCMCIA Port	Channel Symbol:	
Application-Specific Function:	CANopen		
Inputs	Address of first %MW	32	Length: 424
Outputs	Address of first %MW	1056	Length: 102
	Output fallback strategy: Reset to zero		

Configuration mode:	Automatic	TSX CPP 110 watchdog:		active
Load mode:	Unity Pro	CANopen configuration file:		E:\BG1.CO
CANopen Bus Configuration:	Transmission speed :	1 MBits/s	COB-ID message Sync:	128
	Auto-Clear:	off	Sync message period:	100 ms

CANopen slave configuration

Addr.	Type	Act./Guard poll
1	AMM 09000	1 / 1
2	Profile 401 standard EDS	1 / 0
3	ADM 37010	1 / 0

CANopen slave language objects:

Addr.	Inputs	Symbol	Outputs	Symbol
1.	%MW32		%MW1056	
	%MW33			
	%MW34			
	%MW35		%MW1057	
	%MW36		%MW1058	

Section 3.3
Programming

Subject of this Section

This section describes the tools available to program the operation of and obtain information on a CANopen bus managed by PCMCIA card TSX CPP 110.

It is possible to program the operation of the CANopen bus using UNI-TE requests:

- Send and receive SDO messages on the bus,
- Access the link layer by sending PDUs.

It is also possible to monitor the bus and its operation:

- Identification of the master,
- Send diagnostics requests on bus devices.

These requests are sent to the CANopen master (TSX CPP 110 PCMCIA card) to be processed.

What Is in This Section?

This section contains the following topics:

Access to the CANopen SDOs

At a Glance

The READ_VAR *(see Unity Pro, Communication, Block Library)* and WRITE_VAR *(see Unity Pro, Communication, Block Library)* communication functions are used to access the transfer of SDO-type CAN open data. The parameters of these functions determine the action that is performed.

These services are based on CANopen standard CMS standardized message handling. See the documentation for the CANopen slaves for information about the SDO formats used.

⚠ CAUTION

UNEXPECTED APPLICATION BEHAVIOR

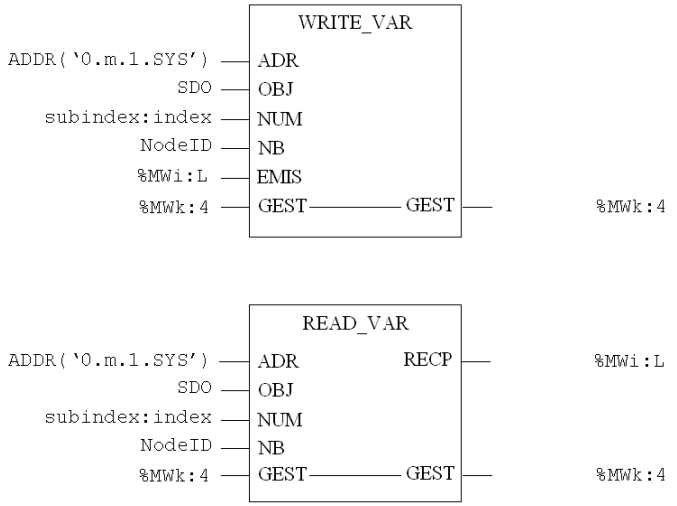
Do not send or receive SDOs simultaneously.

Failure to follow these instructions can result in injury or equipment damage.

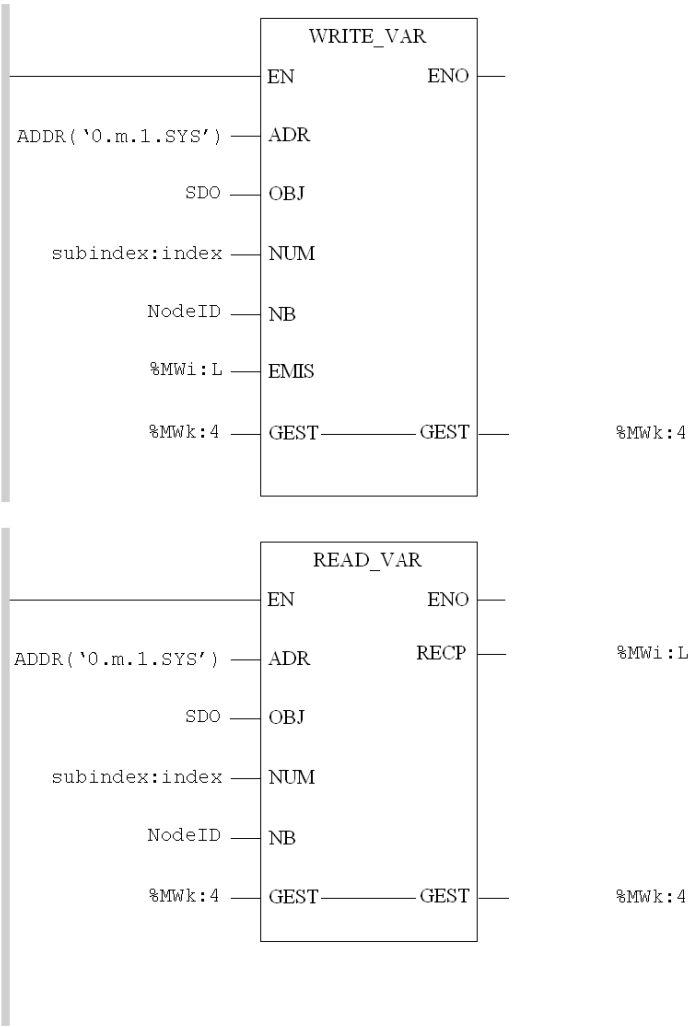
If SDOs are send and receive simultaneously, the following SDO may not be processed. If this happens, perform a cold restart using the processor RESET button to reset the application and return to the normal operating mode.

Representation

FBD Representation:



LD Representation:



ST Representation:

The communication function syntax is as follows:

```
WRITE_VAR(ADDR('0.m.1.SYS'), 'SDO', index:subindex, NodeID, %MWi:L,
%MWk:4)
```

```
READ_VAR(ADDR('0.m.1.SYS'), 'SDO', index:subindex, NodeID, %MWk:4,
%MWi:L)
```

Parameter Description of the WRITE_VAR Function

The communication function syntax is as follows:

The following table describes the various parameters of the function.

Parameter	Description
`ADDR('0.m.1.SYS')`	Address of the exchange destination entity: ● m: processor slot in the rack (0 or 1) ● 1: channel (always 1) ● SYS: UNI-TE server of the PCMCIA card
`'SDO'`	SDO object type (always **SDO** in capitals)
`subindex:index`	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. **Example**: if you use the double word %MD0: ● %MW0: contains the index, ● %MW1: contains the sub-index.
`NodeID`	Word or value identifying the destination device on the CANopen bus
`%MWi:L`	Table of words containing the data to be sent (minimum length = 1)
`%MWk:4`	Exchange management parameters: four words identifying the address of the data used to control the function called `WRITE_VAR`

Parameter Description of the READ_VAR Function

The communication function syntax is as follows:

The following table describes the various parameters of the function.

Parameter	Description
`ADDR('0.m.1.SYS')`	Address of the exchange destination entity: ● m: processor slot in the rack (0 or 1) ● 1: channel (always 1) ● SYS: UNI-TE server of the PCMCIA card
`'SDO'`	SDO object type (always **SDO** in capitals)
`subindex:index`	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. **Example**: if you use the double word %MD0: ● %MW0: contains the index, ● %MW1: contains the sub-index.
`NodeID`	Word or value identifying the destination device on the CANopen bus
`%MWk:4`	Exchange management parameters: four words identifying the address of the data used to control the function called `READ_VAR`
`%MWi:L`	Table of words containing the data to be received (minimum length = 1)

Management Parameters

The management parameters are grouped together in the form of an array of 4 integers. The values contained the this array can be used to manage communication functions. They are detected by the processor that ran the function.

The following table gives details of the %MWk words.

Word number	Most significant byte	Least significant byte	Data managed by
%MWk	Exchange number	Activity bit	The system
%MWk+1	Operation report	Communication report	
%MWk+2	Timeout		You
%MWk+3	Byte length: ● for a WRITE_VAR, initialize this word with the number of bytes to be sent, at each time, before sending the write_var. ● for a READ_VAR, when the request is terminated, this word contains the number of characters received in the word table of received data.		

NOTE: A function can detect a parameter error before activating the exchange. The activity bit remains at 0 and the communication report is initialized with the values corresponding to the detected error.

The following table explains the meaning of the values of the %MWk+1 word:

Word number	Operation report	Communication report
%MWk+1	These codes are for CANopen services on a remote application. 16#00: Positive result 16#01: Request not processed 16#02: Incorrect response 16#03: Reserved	These codes are for FBs READ_VAR, WRITE_VAR & SEND_REQ on a remote application. This report is only significant when activity bit goes from 1 to 0. 16#00: Correct message exchange
	16#00	16#01: Exchange stopped due to timeout 16#02: Exchange stopped due to user demand (CANCEL) 16#03: Incorrect address format 16#04: Incorrect destination format 16#05: Incorrect management parameter format 16#06: Incorrect specific parameter 16#07: Problem sending to destination 16#08: Reserved 16#09: Insufficient receive buffer size 16#0A: Insufficient send buffer size 16#0B: No system resources: the number of simultaneous communication EFs exceeds the maximum that can be managed by the processor 16#0C: Incorrect exchange number 16#0D: No telegram received 16#0E: Incorrect length 16#0F: Telegram service not configured 16#10: Network module missing 16#11: Request missing 16#12: Application server already active 16#13: UNI-TE transaction number incorrect

Word number	Operation report	Communication report
%MWk+1	16#01: No resources toward the processor 16#02: No line resources 15#03: No device or a device without resources**Note**: This code is only managed by PCMCIA cards TSX FPP 10 and TSX FPP 20. 16#04: Detected line error 16#05: Detected length error 16#06: Inoperative communication channel 16#07: Detected addressing error 16#08: Detected application error 16#0B: No system resources: the number of simultaneous communication EFs exceeds the maximum that can be managed by the processor 16#0C: Communication function not active 16#0D: Destination missing 16#0F: Inter-station routing problem or channel not configured 16#11: Incorrect address format 16#12: No destination resource 16#14: Inoperative connection (for example: Ethernet TCP/IP) 16#15: No resource on local channel 16#16: Access not authorized (for example: Ethernet TCP/IP) 16#17: Inconsistent network configuration (for example: Ethernet TCP/IP) 16#18: Connection temporally unavailable 16#21: Application server stopped 16#30: Detected transmission error	16#FF: Message refused

IDENTIFICATION Request

At a Glance

This request enables the master of the CANopen (TSX CPP 110 PCMCIA card) bus to be identified.

This request is carried out using the communication function `SEND_REQ` *(see Unity Pro, Communication, Block Library)*.

Syntax

The communication function syntax is as follows:

```
SEND_REQ(ADDR('0.m.1.SYS'), 16#0F, %MWi:L, %MWk:4, %MWj:L)
```

The following table describes the different function parameters.

Parameter	Description
`ADDR('0.m.1.SYS')`	Address of the exchange destination entity. ● m: Processor slot number, 0 or 1 ● 1: channel (always 1) ● SYS: access to UNI-TE server from the PCMCIA card
`16#0F`	Request code
`%MWi:L`	Not used for the `IDENTIFICATION` function (length is 1)
`%MWk:4`	Exchange management parameters: four words identifying the address of the data used to control the function called `IDENTIFICATION`
`%MWj:L`	Table of words containing the card identification information. Length L must be 12.

%MWj:L

The following table gives details of `%MWj:L` words.

Word number	Most significant byte	Least significant byte
%MWj	Product code: ● 16#05: Premium	16#FF
%MWj+1	Length of identification string: 16#0C	Number of BCD coded version (Version 1.0 coded 16#10)
%MWj+2	‘S’	‘T’
%MWj+3	’ ’	‘X’
%MWj+4	‘P’	‘C’
%MWj+5	’ ’	‘P’
%MWj+6	’1’	’1’

Word number	Most significant byte	Least significant byte
%MWj+7	16#00	'0'
%MWj+8	Indicator lamp status. The COM LED is coded on the first two bits and the ERR LED on the following two according to the following sequences: ● 0, 0: Off, ● 0, 1: Blinking, ● 1, 0: Permanently on.	PCMCIA card state: ● 0: Absent, ● 1: Self-test, ● 2: Failure, ● 3: Ready, ● 4: Waiting, ● 5: Not configured.
%MWj+9	Product type: 16#02	Functional type: 16#2E
%MWj+10	Error type ● bit 0: Card in Test/Debug mode, ● bit 1: Bus error or inactive bus, ● bit 2: Absent connection unit, ● bit 3: Self-testing or inaccessible card, ● bit 4: Reserved, ● bit 5: Card different to the one configured, ● bit 6: Absent card, ● bit 7: Error on at least one slave.	Catalog reference: 16#01
%MWj+11	-	16#00

Management Parameters

The following table gives details of the `%MWk:4` words.

Word number	Most significant byte	Least significant byte	Data managed by
%MWk	Exchange number	Activity bit	The system
%MWk+1	Operation report, positive 16#3F report	Communication report	
%MWk+2	Timeout		You
%MWk+3	Length: initialization to 0 is obligatory before the function is sent		

Diagnostics Command

At a Glance

Diagnostic commands are sent by the SEND_REQ *(see Unity Pro, Communication, Block Library)* function block. In this case the SEND_REQ function is used to:

- Obtain diagnostics for slaves on the bus,
- Obtain the version of a PCMCIA CAN open card,
- Obtain state variables,
- Obtain the history of bus errors.

Syntax

The syntax of the communication function is the following:

```
SEND_REQ (ADDR('0.m.1.SYS', 16#0031, %MWi:3, %Mwk:4, %MWj:L)
```

The following table describes the different function parameters.

Parameter	Description
ADDR('0.m.1.SYS')	Address of the exchange destination entity. ● m: processor slot in the rack (0 or 1) ● 1: channel (always 1) ● SYS: UNI-TE server of the PCMCIA card
16#0031	Request code

%MWi:3	Parameters of the request:
	● %MWi: diagnostic object type:
	  ● 1 ... 127: Diagnostics of slaves 1 to 127
	  ● 128: CANopen card version
	  ● 129: Status of the CANopen card, the response is equivalent to the contents of the %IW0.m.1.i *(see page 90)* status words.
	  ● 130: History of messaging errors
	● %MWi+1: Starting address in the diagnostics table (Value by default 0). To enable partial access to the diagnostics table, specify a starting word in the table (Start offset)
	● %MWi+2: Length in bytes of the diagnostics to be read, this length is generally two times the length of the responses table

%MWk:4	Exchange management parameters: four words identifying the address of the data used to control the SEND_REQ function.
%MWj:L	Reception table contains diagnostics data *(see page 68)*.

Management Parameters

The following table gives details of the %MWk:4 words.

Word number	Most significant byte	Least significant byte	Data managed by
%MWk	Exchange number	Activity bit	The system
%MWk+1	Operation report: ● positive response: 16#61 ● incorrect response: 16#FD	Communication report	
%MWk+2	Timeout		You
%MWk+3	Length: number of response bytes (initialization compulsory in order to activate the function if the number of bytes sent is 6)		

Section 3.4
Debug

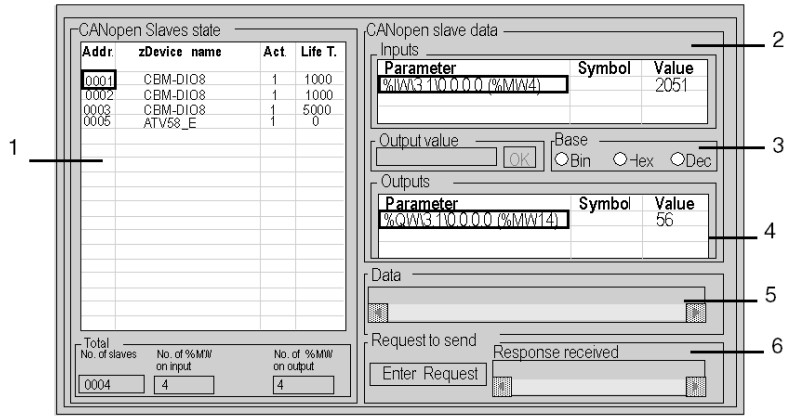
Description of the Debug Screen

At a Glance

The Debug function or the ability to double click on the TSX CPP 110 PCMCIA card in the Unity Pro software is only available in online mode.

Illustration

The figure below is an example of a debug screen.



Description

The table below shows the different zones of the debug screen:

Number	Element	Function
1	CANopen slave status	This zone displays all the CANopen bus slaves. A faulty slave is displayed in red; when this fault disappears, it is displayed in blue. Otherwise, it is displayed in black. Selecting a slave updates zones 2, 4 and 5. **Act.** : indicates whether the slave was activated in the Sycon configuration (1=activated, 0=deactivated) **Life T.**: Life Time.
2	Inputs	When a slave is selected, this zone contains the list of words, which are associated with it on input. A clearer explanation is given with the display of the CANopen topological address *(see page 35)* (%IW\3.1\0.0.0.0) and reserved memory area in the PLC (%MW4).
3	Output value	When an output word in zone 4 is selected, its value can be modified by entering a new value then clicking on the **OK** button.
4	Outputs	When a slave is selected, this zone contains the list of words, which are associated with it on output. A clearer explanation is given with the display of the CANopen topological address *(see page 35)* (%QW\3.1\0.0.0.0) and reserved memory area in the PLC (%MW14).
5	Information on	When a slave is selected (click in zone 1), this zone contains its last diagnostic message, and to obtain information on the TSX CPP 110 card simply click on the table header.
6	Request to send	This zone is used to send a SDO request. Parameter syntax is identical to that used to transfer SDOs using READ_VAR and WRITE_VAR requests *(see page 54)*. Pressing the **Enter Request** button makes the request input fields appear.

Section 3.5
Diagnostics

Subject of this Section

This section describes the equipment and software diagnostics tools available for the TSX CPP 110 PCMCIA card.

What Is in This Section?

This section contains the following topics:

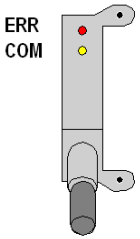
Diagnostics Using the Status LEDs on the TSX CPP 110 PCMCIA Card

At a Glance

LEDs on the card allow you to see the status of the card and of the CANopen bus. In normal operation, the ERR LED is off and the COM LED is on permanently.

Illustration

The following figure indicates the positioning of the two LEDs, ERR and COM.



Diagnostics

Depending on the status of the LEDs, the diagnostics are as follows:

Status display LEDs		**Meaning**
ERR (red)	**COM** (yellow)	
Off	Off	Card has no power supply or configuration transfer in progress
	Blinking (Irregular)	No configuration on the card
	Blinking (Regular)	Card configured and ready, bus not active or no CANopen firmware
	On	Bus configured and active, no error
Perma-nently on	Off	Error detected, bus controller stopped
	Blinking	Card configured and ready, but unable to communicate with a remote peripheral device (e.g. CAN bus cable disconnected), or all the configured peripheral devices report an error. Error on card, configuration error, or error on synchronization between the card and the PLC (for more information, consult the module status diagnostics data)
	On	Bus configured and active, at least one bus subscriber cannot be reached or signals an error

Diagnostics Data

At a Glance

During diagnostics, the first data to be used are implicit exchange %IW words *(see page 90)*.

Elsewhere there are diagnostics data which can be consulted by writing some program lines onto the PLC.

The communication function SEND_REQ enables diagnostics to be carried out *(see page 62)*:

- On the slave of its choice (1 to 127, one request per slave),
- On the PCMCIA card version (128),
- On the card state (129),

And is used to obtain the history of messaging error messages (130).

NOTE: Code 129 enables the same information to be received as is contained in the implicit exchange input words *(see page 91)*.

The information provided comes from the PCMCIA card and is updated periodically.

The request reception table contains the information described in the following paragraphs.

NOTE: Information is given in byte tables. Whilst taking into account the possibility of requesting all or part of this table, it is necessary to pay attention to the most significant and least significant words of the `%MWi:L` table.

NOTE: Diagnostic information complies with the CAN standard. You can find reference information on the following site: `http://www.can-cia.de`.

Slave Diagnostics

The following table describes the information received following a request for diagnostics on a slave (code 1 to 127).

Byte rank	Description
0	Device status bits: ● Bit 0: No response ● Bit 1: Overflow of error messages history table ● Bit 2: Parameters error ● Bit 3: Monitoring active device ● Bit 4 to bit 6: Reserved ● Bit 7: Deactivated
1 and 2	Complementary information read at bus start-up on the standard object 16#1000 (CAN standardization)
3 and 4	Number of profile read at bus start-up on the standard object 16#1000 (CAN standardization)
5	Slave status: ● 1: Disconnected ● 2: Connection in progress ● 3: In preparation ● 4: Ready ● 5: Operating ● 127: Being prepared or missing
6	Error code *(see page 103)* (code of the last error generated by the slave)
7	Number of blocks of urgent information on the slave (0 to 5). **Note**: These blocks are added to the end of this table. Details of a typical block are given in the following paragraph.

Description of an Information Block on the Slave

The following table describes a typical information block.

Byte rank	Description
0 and 1	Error code
2	Value of error register, object 16#1001 of the slave (CAN standardization)
3 to 6	Register value of the specific manufacturer state, object 16#1002 (CAN normalization)
7	Reserved

Diagnostics on a Card Version

The following table describes the information received following a request for diagnostics on a PCMCIA card (code 128).

Byte rank	Description
0 to 7	Firmware version (character string)
8 to 10	Date of creation of the firmware version (BCD coded, DD.MM.YY)
11 to 13	Date of manufacture (BCD coded, DD.MM.YY)
14 to 17	Serial number (BCD coded)
18 to 25	Protocol name (ASCII, without end of string character, e.g.: "CANopen")

History of Messaging Errors

The following table describes the information received following a request for a history report (code 130).

Byte rank	Description
0 and 1	Number of error blocks **Note**: These blocks are added to the end of this table. Details of a typical block are given in the following paragraph.
2 to 49	Contents of the error block (maximum 8 blocks)

Description of a History Report Information Block

The following table describes a typical information block.

Byte rank	Description
0	Service code
1	ID concerned
2 and 3	Messaging error code *(see page 100)*
4 and 5	Detail on error code *(see page 102)*

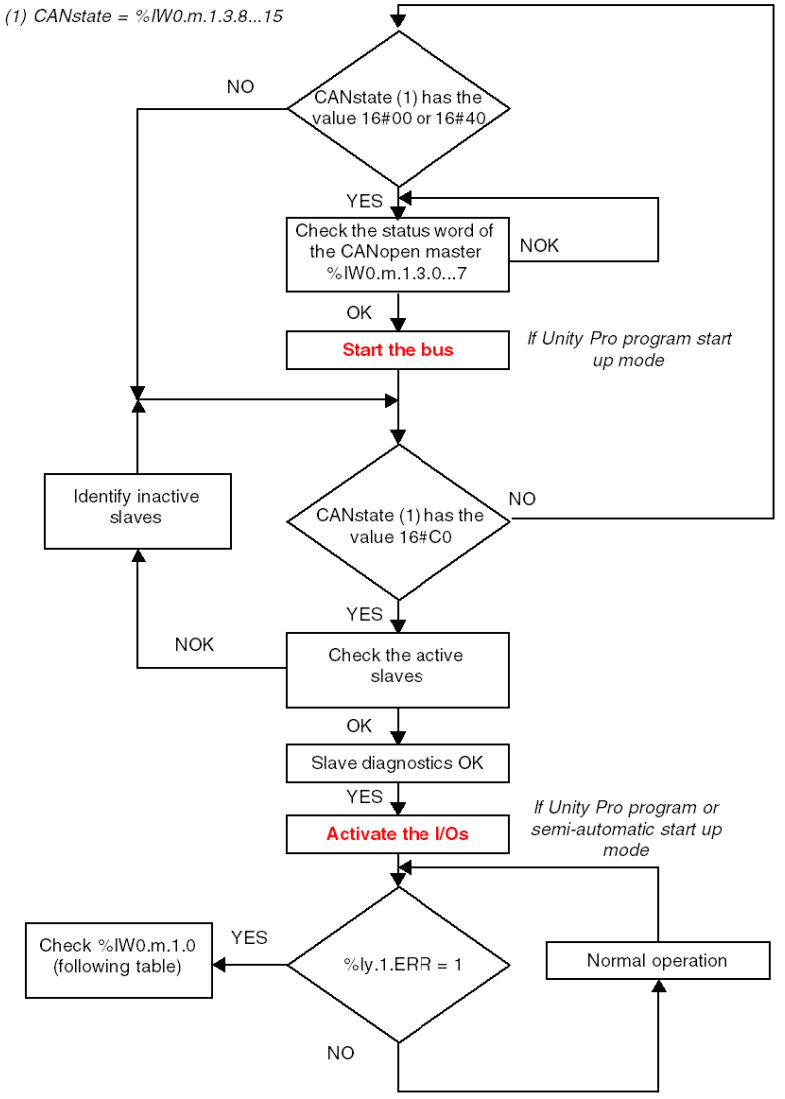
How to perform a diagnostic

At a Glance

You can start by using the PCMCIA card LEDs to search for faults on the CANopen bus. 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 *(see page 75)* provided by the PLC.

Procedure

The following table indicates the different phases of the procedure.



How to check %IW0.m.1.0

This table describes the actions to be carried out in order to obtain an accurate diagnostic with the help of bits x8 to x15 of %IW0.m.1.0.

If	Then
bit 8 =1	it is a configuration error Check the detail of the error codes *(see page 94)* in the words: ● %IW0.m.1.1 ● %IW0.m.1.2
bit 9 =1	it is a PDO transfer error Contact Schneider technical support
bit 10 =1	it is an SDO transfer error Check the detail of the error codes *(see page 94)* in the words: ● %IW0.m.1.1 ● %IW0.m.1.2 Check the historical report of messaging errors *(see page 103)*.

bit 11 =1	it is a PCMCIA card error.
	● Check the detail of the error codes in %IW0.m.1.1.
	● Check the contents of %IW0.m.1.3:
	  ● bit 0: parameter error, the source of the error is indicated in %IW0.m.1.4
	  ● bit 1: the outputs are at zero after the failure of a slave (Autoclear ON), the source of the error is indicated in %IW0.m.1.4
	  ● bit 3: serious error, the card is not active on the bus
	  ● bit 7: faulty connection between the card and the connection unit.

bit 12 =1	it is a bus error (bus not started or transmission error detected). ● Check the number of bus errors counter %IW0.m.1.5; if it is other than zero, check the line. ● Check the number of bus stops counter, if it is increasing, check the line and restart the bus. **Note**: In non-automatic start up mode, the start up bit is %QW0.m.1.0.
bit 13 =1	it is an error on a slave: communication error or I/Os not activated. ● Determine the last source of error contained in %IW0.m.1.4 ● Determine all the active slaves on the bus by consulting the %IW0.m.1.16 to %IW0.m.1.23 bus status words. ● Perform a diagnostic of the faulty slaves using a diagnostic request *(see page 62)*. **Note**: In non-automatic start up mode, the I/O start up bit is %QW0.m.1.1.
bit 14 =1	it is an output error: the outputs are positioned in fallback conditions. ● Check that the PLC is in RUN mode ● Check that the task associated with the module is active ● Test the %IW0.m.1.0.12 bit (Bus error) and the %IW0.m.1.0.13 bit (Slave error). **Note**: In non-automatic start up mode, test the %QW0.m.1.0 and %QW0.m.1.1 bits.
bit 15 =1	a new diagnostic is available for one or more slaves. ● Determine these slaves by using the status words %IW0.m.1.16 to %IW0.m.1.23 ● Carry out a diagnostic *(see page 62)* for the slave(s) concerned

Chapter 4
CANopen Language Objects

Subject of this Chapter

This chapter describes the language objects associated with the CANopen communication channel.

What Is in This Chapter?

This chapter contains the following sections:

Section 4.1
Language objects and IODDTs for CANopen communication

Aim of this Section

This section provides an overview of the general points concerning IODDTs and language objects for CANopen communication.

What Is in This Section?

This section contains the following topics:

Introduction to the Language Objects for CANopen Communication

General

The IODDTs are predefined by the manufacturer. They contain input/output language objects belonging to a channel of an application-specific module.

CANopen communication has two associated IODDTs:

- `T_COM_STS_GEN`, which applies to communication protocols except Fipio and Ethernet,
- `T_COM_CPP110` which is specific to CANopen communication.

NOTE: IODDT variables can be created in two different ways:
- Using the I/O objects *(see Unity Pro, Operating Modes)* tab,
- Using the Data Editor. *(see Unity Pro, Operating Modes)*

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 object:
- **Implicit exchange objects**, which are automatically exchanged on each cycle of the task associated with the module,
- **Explicit exchange objects**, which are exchanged on the application's request, 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.

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.

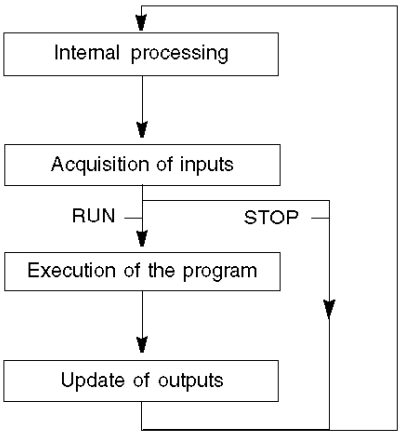
The outputs (`%Q` and `%QW`) are updated at the end of the task, only when the PLC is in RUN mode.

NOTE: When the task occurs in STOP mode, either of the following are possible, 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

Introduction

Explicit exchanges are performed at the user program's request using these instructions:

- READ_STS *(see Unity Pro, I/O Management, Block Library)* (read status words)
- WRITE_CMD *(see Unity Pro, I/O Management, Block Library)* (write command words)
- WRITE_PARAM *(see Unity Pro, I/O Management, Block Library)* (write adjustment parameters)
- READ_PARAM *(see Unity Pro, I/O Management, Block Library)* (read adjustment parameters)
- SAVE_PARAM *(see Unity Pro, I/O Management, Block Library)* (save adjustment parameters)
- RESTORE_PARAM *(see Unity Pro, I/O Management, Block Library)* (restore adjustment parameters)

These exchanges apply to a set of %MW objects of the same type (status, commands or parameters) that belong to a channel.

These objects can:

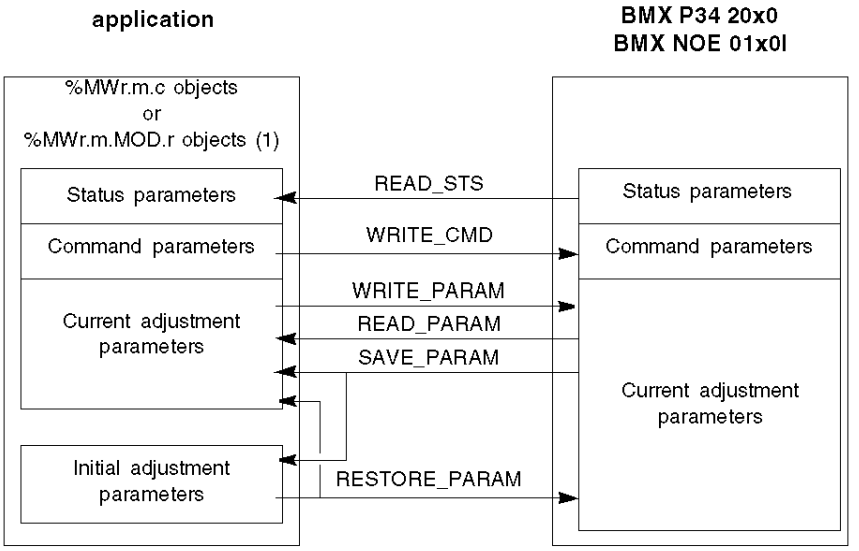
- provide information about the module (for example, type of error detected in a channel)
- have command control of the module (for example, switch command)
- define the module's operating modes (save and restore adjustment parameters in the process of application)

NOTE: To avoid several simultaneous explicit exchanges for the same channel, it is necessary to test the value of the word EXCH_STS (`%MWr.m.c.0`) of the IODDT associated to the channel before calling any EF addressing this channel.

NOTE: Explicit exchanges are not supported when M340 analog and digital I/O modules are configured through an M340 Ethernet RIO adapter module in a Quantum EIO configuration. You cannot set up a module's parameters from the PLC application during operation.

General Principle for Using Explicit Instructions

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



(1) Only with READ_STS and WRITE_CMD instructions.

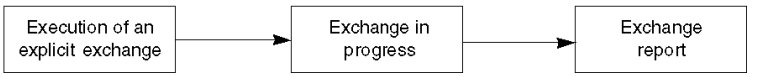
Managing Exchanges

During an explicit exchange, check performance to see that the 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 *(see page 83)*
- the exchange report *(see page 84)*

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 (`%MWr.m.c.0`) of the IODDT associated to the channel before calling any EF addressing 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. IODDTs use two words to manage exchanges:

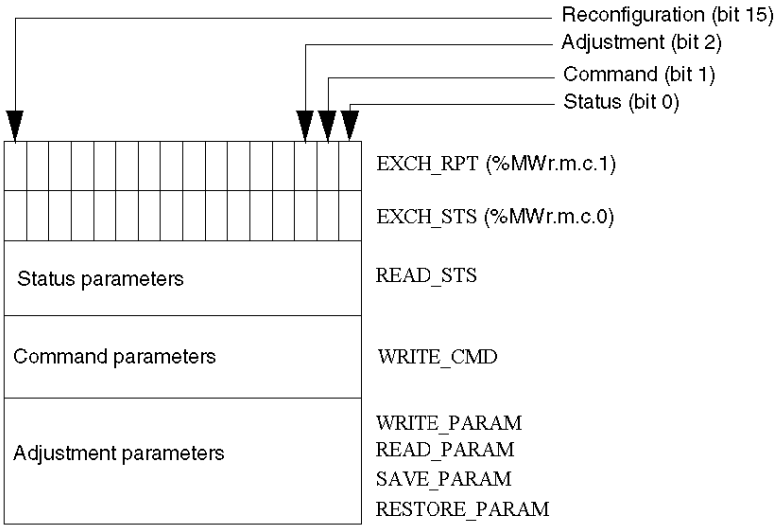
- `EXCH_STS` (`%MWr.m.c.0`): exchange in progress
- `EXCH_RPT` (`%MWr.m.c.1`): report

NOTE:
Depending on the localization of the module, the management of the explicit exchanges (`%MW0.0.MOD.0.0` for example) will not be detected by the application:

- For in-rack modules, explicit exchanges are done immediately on the local PLC Bus and are finished before the end of the execution task. So, the `READ_STS`, for example, is finished when the `%MW0.0.mod.0.0` bit is checked by the application.
- For remote bus (Fipio for example), explicit exchanges are not synchronous with the execution task, so the detection is possible by the application.

Illustration

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



Description of Significant Bits

Each bit of the words EXCH_STS (%MWr.m.c.0) and EXCH_RPT (%MWr.m.c.1) is associated with a type of parameter:

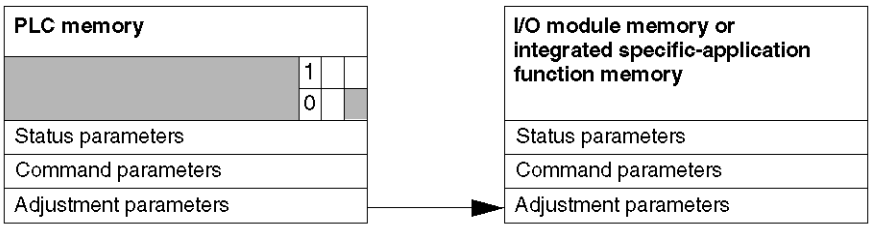
- Rank 0 bits are associated with the status parameters:
  - The STS_IN_PROGR bit (%MWr.m.c.0.0) indicates whether a read request for the status words is in progress.
  - The STS_ERR bit (%MWr.m.c.1.0) specifies whether a read request for the status words is accepted by the module channel.
- Rank 1 bits are associated with the command parameters:
  - The CMD_IN_PROGR bit (%MWr.m.c.0.1) indicates whether command parameters are being sent to the module channel.
  - The CMD_ERR bit (%MWr.m.c.1.1) specifies whether the command parameters are accepted by the module channel.
- Rank 2 bits are associated with the adjustment parameters:
  - The ADJ_IN_PROGR bit (%MWr.m.c.0.2) indicates whether the adjustment parameters are being exchanged with the module channel (via WRITE_PARAM, READ_PARAM, SAVE_PARAM, RESTORE_PARAM).
  - The ADJ_ERR bit (%MWr.m.c.1.2) specifies whether the adjustment parameters are accepted by the module. If the exchange is correctly executed, the bit is set to 0.
- Rank 15 bits indicate a reconfiguration on channel **c** of the module from the console (modification of the configuration parameters + cold start-up of the channel).
- The *r*, *m* and *c* bits indicates the following elements:
  - the **r** bit represents the rack number.
  - The **m** bit represents the position of the module in the rack.
  - The **c** bit represents the channel number in the module.

NOTE: r represents the rack number, **m** the position of the module in the rack, while **c** represents the channel number in the module.

NOTE: Exchange and report words also exist at module level EXCH_STS (%MWr.m.MOD) and EXCH_RPT (%MWr.m.MOD.1) as per IODDT type T_GEN_MOD.

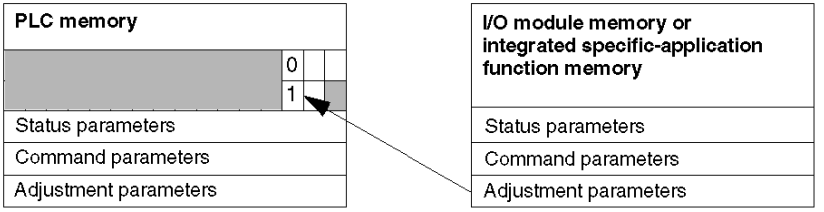
Example

Phase 1: Sending data by using the WRITE_PARAM instruction



When the instruction is scanned by the PLC, the **Exchange in progress** bit is set to 1 in %MWr.m.c.

Phase 2: Analysis of the data by the I/O module and report.



When the data is exchanged between the PLC memory and the module, acknowledgement by the module is managed by the ADJ_ERR bit (%MWr.m.c.1.2).

This bit makes the following reports:

- **0:** correct exchange
- **1:** incorrect exchange)

NOTE: There is no adjustment parameter at module level.

Execution Indicators for an Explicit Exchange: EXCH_STS

The table below shows the control bits of the explicit exchanges: 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	Command parameters exchange in progress	%MWr.m.c.0.1
ADJ_IN_PROGR	BOOL	R	Adjust parameters exchange in progress	%MWr.m.c.0.2
RECONF_IN_PROGR	BOOL	R	Reconfiguration of the module in progress	%MWr.m.c.0.15

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 shows the report bits: `EXCH_RPT` (`%MWr.m.c.1`)

Standard Symbol	Type	Access	Meaning	Address
STS_ERR	BOOL	R	Error detected while reading channel status words (1 = detected error)	%MWr.m.c.1.0
CMD_ERR	BOOL	R	Error detected during a command parameter exchange (1 = detected error)	%MWr.m.c.1.1
ADJ_ERR	BOOL	R	Error dectected during an adjust parameter exchange (1 = detected error)	%MWr.m.c.1.2
RECONF_ERR	BOOL	R	Error detected during reconfiguration of the channel (1 = detected error)	%MWr.m.c.1.15

Counting Module Use

The following table describes the steps realized between a couting module and the system after a power-on.

Step	Action
1	Power on.
2	The system sends the configuration parameters.
3	The system sends the adjust parameters by WRITE_PARAM method. **Note:** When the operation is finished, the bit %MWr.m.c.0.2 switches to 0.

If, in the begining of your application, you use a WRITE_PARAM command, wait until the bit %MWr.m.c.0.2 switches to 0.

Section 4.2
Language Objects and Generic IODDT Applicable to Communication Protocols

About this Section

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

What Is in This Section?

This section contains the following topics:

Details of IODDT Implicit Exchange Objects of Type T_COM_STS_GEN

Introduction

The following table presents the IODDT implicit exchange objects of type `T_COM_STS_GEN` applicable to all communication protocols except Fipio and Ethernet.

Error Bit

The table below presents the meaning of the detected 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

Introduction

This section presents the T_COM_STS_GEN type IODDT explicit exchange objects applicable to all communication protocols except Fipio and Ethernet. 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 inoperative.	%MWr.m.c.2.1
BLK	BOOL	R	Terminal block not connected.	%MWr.m.c.2.2
TO_ERR	BOOL	R	Time out exceeded anomaly.	%MWr.m.c.2.3
INTERNAL_FLT	BOOL	R	Internal detected 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	Interruption of the communication with the PLC.	%MWr.m.c.2.6
APPLI_FLT	BOOL	R	Application detected error (adjustment or configuration).	%MWr.m.c.2.7

Section 4.3
Language Objects of the CANopen Specific IODDT

Subject of this Section

This section describes the implicit and explicit language objects of the CANopen specific IODDT, `T_COM_CPP110`.

What Is in This Section?

This section contains the following topics:

Details of T_COM_CPP110 Type Implicit Exchange Objects of the IODDT

At a Glance

The tables below describe all the language objects with implicit exchange of the IODDT of type `T_COM_CPP110` for a CANopen communication with the PCMCIA TSX CPP 110 card.

Error Bit

The table below shows the different bit objects for implicit exchange.

Standard symbol	Type	Access	Meaning	Address
CH_ERROR	EBOOL	R	Communication channel error bit.	%I0.m.0.ERR

Communication Status Bits

The table below presents the error word for management of the CANopen bus. This is detailed bit by bit.

Standard symbol	Type	Access	Meaning	Address
CAN_FLT	BOOL	R	Channel error, logical **OR** between all the bits that follow except bit 15.	%IW0.m.1.0.0
SOFT_CONF	BOOL	R	Incorrectly configured	%IW0.m.1.0.8
PDO_EXCH_FLT	BOOL	R	Error in I/O exchange (PDO)	%IW0.m.1.0.9
MSG_EXCH_FLT	BOOL	R	Messaging exchange error (SDO)	%IW0.m.1.0.10
CARD_FLT	BOOL	R	Card error (card missing or not ready)	%IW0.m.1.0.11
BUS_FLT	BOOL	R	Bus fault (at least one **bus error** type event has been generated)	%IW0.m.1.0.12
SLAVE_FLT_B	BOOL	R	Slave fault (a communication error has been detected with one or more slaves)	%IW0.m.1.0.13
OUTP_FLT	BOOL	R	Output error (outputs are positioned at fallback conditions)	%IW0.m.1.0.14
SLAVE_DIAG	BOOL	R	New slave diagnostics available (a new diagnostic is available for one or more slaves)	%IW0.m.1.0.15

CANopen Master Status Bits

The table below presents the error word of the CANopen bus master `MAST_STS` (`%IW0.m.1.3`). This is detailed bit by bit.

Standard symbol	Type	Access	Meaning	Address
PARAM_FLT	BOOL	R	Parameters error	%IW0.m.1.3.0
SLAVE_OUTP_FLT	BOOL	R	Indicates that the outputs are at zero following the failure of a slave, Autoclear ON.	%IW0.m.1.3.1
NO_BUS_EXCH	BOOL	R	No exchanges on the bus (no slave is communicating)	%IW0.m.1.3.2
CARD_NO_ACT	BOOL	R	Serious error, the card is not active on the bus	%IW0.m.1.3.3
FEW_BUS_ERR	BOOL	R	One or more **bus error** events have been detected	%IW0.m.1.3.4
CARD_ACC_PROH	BOOL	R	The processor has not yet authorized access to the card.	%IW0.m.1.3.5
TIMEOUT_MSG	BOOL	R	Timeout exceeded when sending CAN messages	%IW0.m.1.3.6
CNX_FLT	BOOL	R	Faulty connection between the card and the connection unit.	%IW0.m.1.3.7
-	-	R	Bits 8 to 15 of `MAST_STS` constitute a byte whose value indicates the operating mode. ● 16#00: offline mode ● 16#40: bus in STOP mode ● 16#80: outputs are in security mode (set to zero) ● 16#C0: bus in RUN mode	%IW0.m.1.3.8 to %IW0.m.1.3.15

Status Word for Bus Devices

The table below presents the status word for the CANopen bus devices.

Standard symbol	Type	Access	Meaning	Address
SLAVE_STS	INT	R	The least significant byte (bits 0 to 7) contains the address of the slave which has generated the last error. The most significant byte (bits x8 to x15) contains the last error code.	%IW0.m.1.4

Bus Status Words

The table below presents the status words for the CANopen bus. These words are broken down into bits that each represent a bus slave.

Standard symbol	Type	Access	Meaning	Address
SLAVE_ACTIV_0	BOOL	R	Slave 0 active on the bus.	%IW0.m.1.8.0
SLAVE_ACTIV_1	BOOL	R	Slave 1 active on the bus.	%IW0.m.1.8.1
...	...	...	...	...
SLAVE_ACTIV_15	BOOL	R	Slave 15 active on the bus.	%IW0.m.1.8.15
SLAVE_ACTIV_16	BOOL	R	Slave 16 active on the bus.	%IW0.m.1.9.0
...	...	...	...	...
SLAVE_ACTIV_31	BOOL	R	Slave 31 active on the bus.	%IW0.m.1.9.15
SLAVE_ACTIV_32	BOOL	R	Slave 32 active on the bus.	%IW0.m.1.10.0
...	...	...	...	...
SLAVE_ACTIV_47	BOOL	R	Slave 47 active on the bus.	%IW0.m.1.10.15
SLAVE_ACTIV_48	BOOL	R	Slave 48 active on the bus.	%IW0.m.1.11.0
...	...	...	...	...
SLAVE_ACTIV_63	BOOL	R	Slave 63 active on the bus.	%IW0.m.1.11.15
SLAVE_ACTIV_64	BOOL	R	Slave 64 active on the bus.	%IW0.m.1.12.0
...	...	...	...	...
SLAVE_ACTIV_79	BOOL	R	Slave 79 active on the bus.	%IW0.m.1.12.15
SLAVE_ACTIV_80	BOOL	R	Slave 80 active on the bus.	%IW0.m.1.13.0
...	...	...	...	...
SLAVE_ACTIV_95	BOOL	R	Slave 95 active on the bus.	%IW0.m.1.13.15
SLAVE_ACTIV_96	BOOL	R	Slave 96 active on the bus.	%IW0.m.1.14.0
...	...	...	...	...
SLAVE_ACTIV_111	BOOL	R	Slave 111 active on the bus.	%IW0.m.1.14.15
SLAVE_ACTIV_112	BOOL	R	Slave 112 active on the bus.	%IW0.m.1.15.0
...	...	...	...	...
SLAVE_ACTIV_127	BOOL	R	Slave 127 active on the bus.	%IW0.m.1.15.15

Output Word

The table below presents the output word %QW0.m.1.0 of the CANopen PCMCIA card. It is detailed bit by bit.

Standard symbol	Type	Access	Meaning	Address
ACT_BUS_CONF	BOOL	RW	This bit is only used when the bus startup is managed by the application: ● 1: activates the bus configuration ● 0: deactivates the bus configuration	%QW0.m.1.0.0
ACT_DATA_TR	BOOL	RW	This bit is used when the startup is semi-automatic or managed by the application ● 1: activates data transfer on the bus ● 0: deactivates data transfer on the bus	%QW0.m.1.0.1
INIT_ERR_BIT	BOOL	RW	Initializes the error bits: ● I/O errors, ● messaging errors, ● history errors.	%QW0.m.1.0.2
INIT_CARD	BOOL	RW	Initializes the PCMCIA card. This bit warm starts the card and is only used when the bus start-up is managed by the application.	%QW0.m.1.0.3

NOTE: Command bits 2 and 3 are not automatically reset to zero by the application.

Details of T_COM_CPP110 Type Implicit Exchange Objects Not Belonging to the IODDT

At a Glance

The table below describes the CANopen language objects that do not belong to a specific IODDT but which can be used directly on the basis of their number.

Input Word Objects

The table below shows the different input word objects for implicit exchange that do not belong to an IODDT.

Object (1)	Function	Meaning
%IW0.m.1.1	Error word	This word contains a module error code *(see page 100)* (last configuration or I/O error)
%IW0.m.1.2	Error word	This word contains a detailed module error code *(see page 102)* (last configuration or I/O error)
%IW0.m.1.5	Counter status word	Counter of the number of bus errors
%IW0.m.1.6	Counter status word	Counter of the number of bus stops
%IWy.1.7	Counter status word	Counter of the number of timeouts on CAN messages
%IW0.m.1.16 to %IW0.m.1.23	Bus status words	Diagnostics available on the bus, each bit set to 1 corresponds to a device for which diagnostics are available (8 words of 16 bits, so 128 bits, the master and 127 slaves)

Explicit Exchange Language Objects of the T_COM_CPP110 IODDT

At a Glance

The table below describes all IODDT explicit exchange language objects of type `T_COM_CPP110` for CANopen communication with the PCMCIA TSX CPP 110 card.

Example of declaration of a variable: `IODDT_VAR1` of type `T_COM_CPP110`

Exchange Management Words

The table below presents two explicit exchange management bits belonging to the `T_COM_CPP110` type variables.

Standard symbol	Type	Access	Meaning	Address
STS_IN_PROGR	BOOL	R	Explicit exchange in progress	%MW0.m.1.1.0
STS_ERR	BOOL	R	Error during previous explicit exchange	%MW0.m.1.1.1

Status Word of the TSX CPP 110 Card

The table below presents the status word of the TSX CPP 110 card. This is detailed bit by bit. Reading is performed by a `READ_STS` (`IODDT_VAR1`).

Standard symbol	Type	Access	Meaning	Address
COM_FLT	BOOL	R	Bus has a fault or is not initialized (in start-up mode managed by the application)	%MW0.m.1.2.0
FEW_SLAVE_FLT	BOOL	R	Slave has an error, one or more slaves have errors or are not in RUN mode	%MW0.m.1.2.1
CABL_FLT	BOOL	R	The connection unit has an error or its cabling is faulty	%MW0.m.1.2.2
CARD_MISS	BOOL	R	The PCMCIA card is: ● missing from its slot, or ● not ready, or ● has a serious error	%MW0.m.1.2.3
CARD_NO_ACC	BOOL	R	The PCMCIA card is: ● initializing, so not ready, or ● has an error, or ● cannot be accessed	%MW0.m.1.2.4
CARD_NO_REC	BOOL	R	The card or protocol type is not recognized	%MW0.m.1.2.5
IO_EXCH_FLT	BOOL	R	Error in I/O exchanges	%MW0.m.1.2.6
CONF_FLT	BOOL	R	Configuration or parametering error	%MW0.m.1.2.7

Section 4.4
The IODDT Type T_GEN_MOD Applicable to All Modules

Details of the Language Objects of the T_GEN_MOD-Type IODDT

At a Glance

All the modules of Premium 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.
- Not all bits are 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	%Ir.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

Standard symbol	Type	Access	Meaning	Address
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

Section 4.5
CANopen Configuration Language Objects

Language Objects Associated With Configuration

At a Glance

This page describes all the configuration language objects for CANopen communication with the PCMCIA card TSX CPP 110 that can be displayed by the application program.

Internal Constants

The following table describes the internal constants:

Object	Type	Access	Meaning
%KW0.m.1.0	INT	R	Constant value used by the system Least significant byte: 16#00 Most significant byte: 16#37
%KW0.m.1.1	INT	R	Configuration bits ● Output fallback mode when PLC changes to STOP mode: Bit 0 = 0: Reset Bit 0 = 1: Maintain ● Bit 1 = 0: Load configuration via the terminal Bit 1 = 1: Use Flash EEPROM configuration ● Bus control at start-up: Bit 2 = 0: Automatic Bit 2 = 1: Via the application ● I/O control at start-up: Bit 3 = 0: Automatic Bit 3 = 1: Via the application ● Data exchange synchronization Bit 4 = 0: MAST task Bit 4 = 1: FAST task ● Bit 5 reserved ● CANopen PCMCIA card watchdog Bit 6 = 0: Activated Bit 6 = 1: Deactivated ● Bits 7 to 15 Reserved
%KW0.m.1.2	INT	R	Configuration bits Size of the bus configuration in memory (in number of bytes)
%KW0.m.1.3	INT	R	Configuration bits Size of **input image** zone in memory (in number of words)

Object	Type	Access	Meaning
%KW0.m.1.4	INT	R	Configuration bits Size of **output image** zone in memory (in number of words)
%KW0.m.1.5	INT	R	Configuration bits Address of the start of the **input image** zone (%MW)
%KW0.m.1.6	INT	R	Configuration bits Address of the start of the **output image** zone (%MW)

Section 4.6
CANopen error codes

CANopen Error Codes

At a Glance

The following tables describe the different error codes that can arise in a CANopen configuration.

The explanations given in each table allow you to program the application in order to more easily detect and correct any future abnormal operation.

Module Error Codes

The following table describes the error codes located in the words `%IW0.m.1.1` (in the Description column) and `%IW0.m.1.2` (in the Details column).

Code	Description	Details
0	No error, operating correctly	
	Standard errors	
100	Invalid address (NULL pointer)	-
101	Invalid value	Value
102	Invalid object ID	ID
103	Invalid driver status	Status code
104	Size of read memory zone invalid	Memory size
105	Size of data to be written invalid	Size of data
106	Timeout	Timeout counter for an SDO transfer or loading status when downloading a configuration
107	Synchronization error	-
108	Stopped by the user	-
	Resource errors	
200	Memory overflow	-
201	Memory resources saturated	-
	Messaging errors	
300	Invalid source address	Value of address
301	Invalid destination address	Value of address
302	Invalid service	Service code
303	Invalid service class for a segment ID	Specified value

Code	Description	Details
304	Primitive function of service invalid	Function code
305	ID of service called invalid	Specified value
306	Invalid communication gate	Port number
307	Invalid bus device ID	Value of ID
308	Invalid SDO index	Value of index
309	Invalid SDO sub-index	Value of sub-index
310	Remote error when executing a service	Error code
311	Invalid COB-ID	Value of COB-ID
312	Invalid type of transfer on the link layer	Code of the transfer requested: ● 101: send ● 102: receive ● 103: send and receive
	PCMCIA card errors	
600	Card missing	-
601	Card different to TSX CPP 110 detected	-
602	Card not ready to communicate	-
603	Card no longer in RUN mode	-
	PCMCIA card communication errors	
700	Error when sending a message to the card	-
701	Error when receiving a message coming from the card	-
702	Error when sending an output PDO to the card	-
703	Error when receiving an input PDO from the card	-
	Configuration errors	
800	Incorrect size of bus configuration data	Configuration data size
801	Size of input image data invalid	Size, in number of words, fixed at card start-up
802	Size of output image data invalid	Size, in number of words, fixed at card start-up
803	Overlap of memory areas reserved for inputs and outputs	Type of overlap: ● 1: the start of the input area covers the end of the output area ● 2: the start of the output area covers the end of the input area
804	Data loading zone not found	Type of zone: ● 1: global data ● 2: bus parameters ● 3: synchronization mode

Code	Description	Details
805	Invalid configuration data checksum (inconsistency of bus configuration data in Sycon mode)	Checksum of the bus configuration data located in the card's flash memory
806	Negative report on downloading of configuration	Most significant byte: card error code *(see page 102)*. Least significant byte: ● 16#00: no loading in progress ● 16#01: load request ● 16#02: loading in progress ● 16#03: loading completed ● 16#11: transfer to the PC requested ● 16#12: transfer to the PC in progress

Code 806 Details

The following table describes the error codes, situated in the most significant byte of word `%IW0.m.1.2` and for the value 806 of the word `%IW0.m.1.1`.

Code	Description
48	Timeout
52	Unknown zone code
53	Maximum memory size exceeded
55	Incorrect parameter
57	Sequence error in downloading
59	Incomplete data downloaded
60	Duplicate address
61	Size of PDO address table too big
62	Size of the bus device parameter zone too big
63	Mode of PDO transmission unknown
64	Size of PDO data too big
65	Transmission speed unknown
66	Synchro COB-ID outside limits
67	Synchro message Timer Preset value outside limits
68	Size of input data + offset greater than the maximum size of the input image zone
69	Size of output data + offset greater than the maximum size of the output image zone
70	Inconsistency between the configuration of the PDOs and the PDO address table
71	Invalid length of PDO address table
72	Length of download data invalid

Code	Description
73	Urgent message COB-ID outside limits
74	Bus device monitoring message COB-ID outside limits
75	PDO length indicator outside limits
76	Size of SDO data too big

History Codes

The following table describes the error codes, located in the fifth and sixth byte of the diagnostics history table.

Code	Description
3	Service rejected by the device
17	No response from the device
51	Length of the receive memory zone too long
53	Length of fragmented protocol data greater than the size of the buffer memory
54	Unknown function requested by the card driver
55	Bus device address outside limits
57	Sequence error during a fragmented transfer. The action is canceled.
200	card not configured

Slave Diagnostics Codes

The following table describes the error codes, located in the seventh byte of a slave's diagnostics table.

Code	Description
30	Error in slave monitoring
31	Change of status of a device on the bus, device unavailable
32	Sequence error in bus monitoring protocol
33	No response for a configured PDO
34	No response when configuring device
35	Profile of device configured different to profile of device present on the bus
36	Type of device configured different to type of device present on the bus
37	Unknown SDO response
38	Frame received longer than 8 bytes
39	Device not scanned or stopped (for example in Autoclear mode)

Sycon Loading Error Codes

The following table describes the error codes that can occur while loading configuration or firmware onto the card using the X-WAY driver.

Code	Description
0	No error, operating correctly
Standard errors	
8001	Driver inoperative
8002	Unknown event code from the driver
8003	Command code not recognized by the driver
8004	Command refused
8005	Another command is still active
8006	Command sent to an invalid device
Allocation errors	
8010	No device assigned
8011	Device already assigned
Communication errors	
8020	Sending of a service request when no device is connected
8021	Initialization of a connection when there is already one
8022	Time-out
8030	Driver status read error
8031	Error following the sending of a request on the network
8032	Outbox still busy
8033	Network response error
8034	No response available from the inbox
8035	Input/output data transfer error
Driver initialization errors	
8080	Parametering errors
8081	General driver initialization error
Multitask processing errors	
-1	Work task not created
-2	Invalid task pointer or synchronized object
-3	No synchronization event has been created

Chapter 5
Examples of CANopen bus installation

Subject of this Chapter

This chapter uses an example to describe installation of the CANopen bus with the help of the SyCon tool (V2.8) and Unity Pro software.

What Is in This Chapter?

This chapter contains the following sections:

Section 5.1
Example Description

Example Description

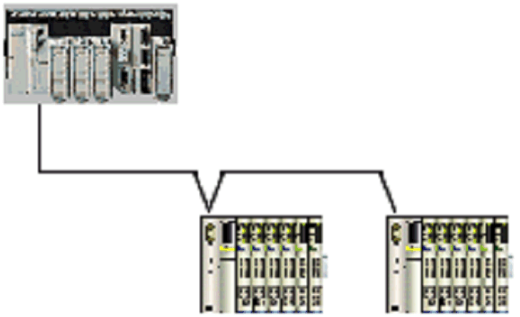
At a Glance

This example is given for educational purposes. It allows you to follow the different stages in the configuration of a CANopen structure made up of:

- A **TSX CPP 110** master module installed in a premium PLC,
- With two **Advantys STB** slave devices:
  - Node # 2: a CANopen STBNCO2212 interface, an STBPDT3100 power supply module, an STBDDI3420 4 input module and an STBDDO3410 4 output module.
  - Node # 3: configuration identical to that for node 2 with in addition, an STBAVI1270 module with two analog inputs and an STBAVO1250 module with two analog outputs.

Illustration

The illustration below is a diagram of the structure used in the example.



Software Required

Implementation of this example requires the following software:

- **Advantys V 1.1**, to configure the STB islands.
- **Sycon V2.8**, to configure the CANopen bus.
- **Unity Pro V2.0**, to configure the PLC.

Section 5.2
Hardware Implementation

Subject of this Section

This section presents implementation of the hardware for the CANopen example.

What Is in This Section?

This section contains the following topics:

Hardware Configuration of Advantys Islands

At a Glance

The first phase of implementation consists of configuring the hardware for the CANopen slaves. Once you have assembled the different elements of the STBs previously described and connected the power supplies, you must follow the steps below.

How to Set Transmission Speed

In this example, we will see how to configure transmission speed of 250 Kbit/s for each STB.

Step	Action
1	Switch the first STB off.
2	Set the top switch to 4. **Note**: 0 = 10 Kbits/s, 1 = 20 Kbits/s, 2 = 50 Kbits/s, 3 = 125 Kbits/s, 4 = 250 Kbits/s, 5 = 500 Kbits/s, 6 = 800 Kbits/s and 7 = 1bits/s.
3	Set the bottom switch to "Baud rate" (position greater than 9).
4	Switch the STB on. **Result**: the STB is configured to work at a speed of 250 Kbit/s.
5	Repeat the same steps for the second STB.

How to Set the CANopen address of the STB

In this second example, we will see how to configure the address of each STB (2 for the first and 3 for the second).

Step	Action
1	Switch the STB off.
2	Set the top switch to 0 (tens figure).
3	Set the bottom switch to 2 (units figure). **Note**: For the second STB, select 3.
4	Switch the STB on. **Result**: the STB is configured with the address defined on the switches.

How to Load the STB Configuration

In this example, we will see how to load a configuration into an STB using automatic transmission without a SIM card.

Step	Action
1	Check that the STB is powered-up.
2	Remove the SIM card if one is present.
3	Press the **Reset** button for more then 5 seconds. **Result**: the STB carries out its start-up and initialization procedure, the hardware configuration is scanned (STB modules), then stored in flash memory. **Note**: If the SIM card is present, the STB tries to load the configuration it contains; if the SIM card is not present, the STB tries to load the configuration contained in flash memory. Pressing the **Reset** button allows flash memory to be updated with the physical configuration actually present. You must always carry out an initialization when the configuration has changed or when you do not know what flash memory contains.

How to Carry Out a Visual Check

The STBs are ready to dialog with a CANopen TSX CPP 110 card, and a last check must be carried out:

Step	Action
1	Make sure that the **RUN** and **PWR** LEDs of the CANopen module (NCO) are lit up.
2	Make sure that the **CANRUN** LED of the CANopen module (NCO) is flashing.
3	Make sure that the **IN** and **OUT** LEDs of the power supply module (PDT) are lit up.
4	Make sure that the **RDY** LED of the I/O modules is lit up for each I/O module. **Result**: the visual diagnostics performed with the help of the LEDs allows us to conclude that the STBs have been correctly configured and are ready to be implemented on a CANopen bus.

Configuration Fault

When the configuration present in flash memory is different from the actual configuration (physical configuration), the LEDs:

- Of the CANopen module (NCO) **RUN** and **PWR** are lit up, the **CANRUN** green LED flashes, and the **ERR** and **CANERR** red LEDS flash.
- Of the power supply module (PDT) **IN** and **OUT** are lit up.
- Of the I/O modules **RDY** and **OUT** are flashing for each module that is not present in the flash memory configuration and are permanently lit for the others.

Module Fault

Certain modules cans display faulty situations (for example, absence of output power supply on a DDO3230 module). In this case, **RDY** is lit up and **ERR** flashes on the faulty module.

Hardware Configuration of the Master

At a Glance

The CANopen bus is managed by a Premium PLC in which a TSX CPP 110 card has been installed and configured.

For more information on the TSX CPP 110 card, please refer to the hardware chapter *(see page 22)* of this document.

Mounting Procedure

The following table describes the procedure for physically installing a TSX CPP 110 card.

Step	Action
1	Check that the PLC power supply is switched off.
2	Install and secure the modules of the base rack.
3	Insert the TSX CPP 110 card into its slot on the processor *(see page 25)*.
4	Wire the PLC power supply and switch the power on. **Note**: Once the PLC's power supply has been correctly connected, the software can now be configured.

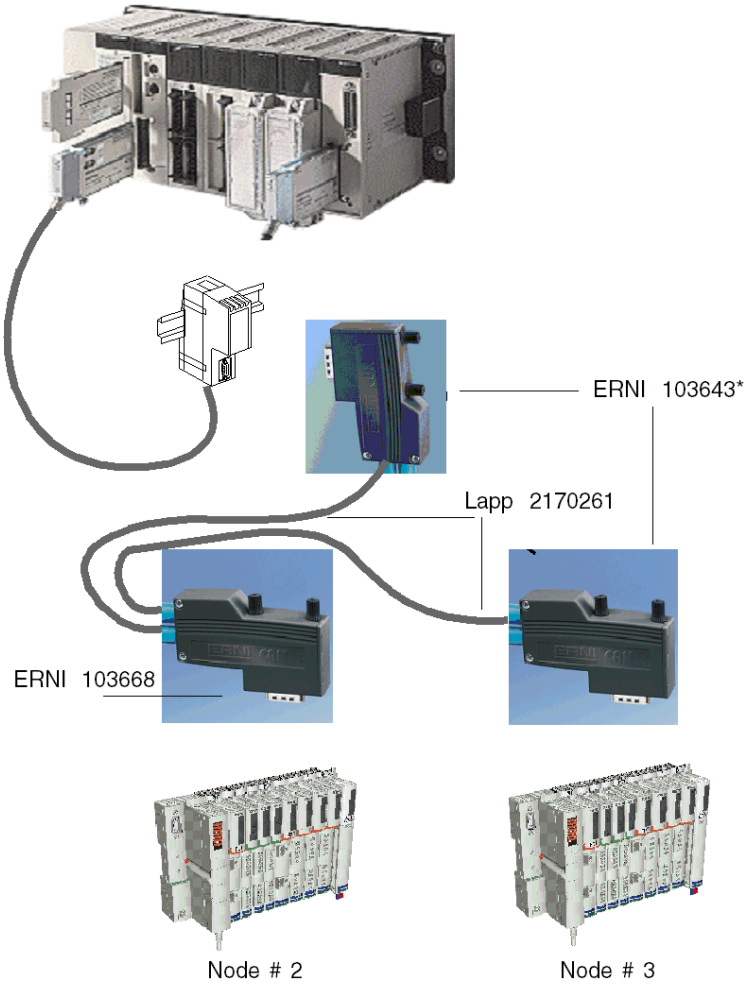
Hardware Configuration of the Bus

At a Glance

Once the hardware configuration for the slaves and CANopen master has been completed, you must now connect these different devices.

Wiring Examples for the Bus

The figure below represents a wiring solution.



* : integrated line termination impedence (120 Ohms).

Some References

Wire example from the **Selectron** company:

- DCA 701 (item code 44170014)

For additional information, see the Internet site: `http://www.selectron.ch`.

Wire example from the **Lapp** company:

- UNITRONIC BUS CAN 2170261: 120 Ohms, double shielded twisted pair cable.

For additional information, see the Internet site: `http://www.lappcable.com/products`.

Example of connectors from the **ERNI** company:

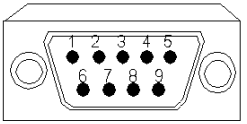
- 1 chaining connector, reference 103668 (connected to node 2).
- 2 bus end connectors, reference 103643 (one of which is connected to the TSX CPP 110 connection device and the other to node 3).

For additional information, see the Internet site: `http://www.erni.com`.

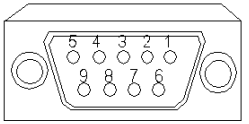
Connector Diagram

The connectors used are 9-pin SUB D connectors:

Male connector



Female connector



Description of the Pins

This table provides the function of each pin.

Element	Description
1	Reserved
2	CAN_L
3	CAN_GND
4	Reserved
5	NC
6	NC
7	CAN_H
8	Reserved
9	NC

NOTE: Make sure that you correctly connect the cable shielding to the connector.

For more details, consult the CANopen Hardware Setup Manual available on telemacanique.com.

Section 5.3
Software Implementation

Subject of this Section

This section presents implementation of the software for the CANopen example.

The steps to be carried out are as follows:

- Creation of the STB Advantys configuration and generation of EDS files for each node.
- Creation of the CANopen configuration using Sycon (use of EDS files previously created to configure each node).
- Creation of the PLC application using Unity Pro and transfer to the Premium/Atrium

What Is in This Section?

This section contains the following topics:

Advantys Software Configuration

At a Glance

Advantys software is used to create and configure Advantys nodes, and then save EDS files.

Its key functions are as follows:

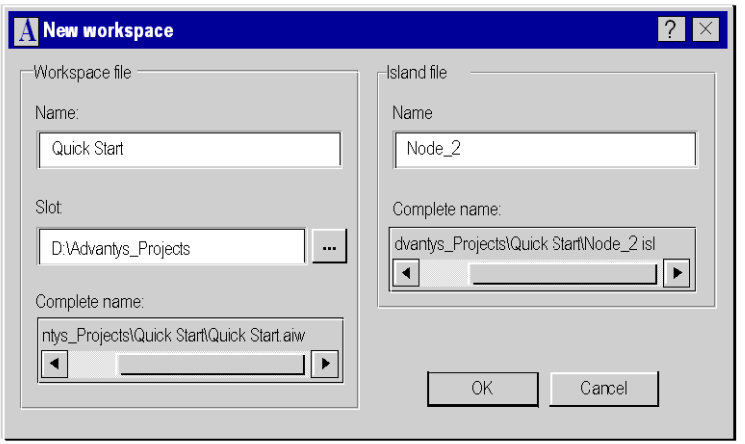
- Modification of default parameters for each of the I/O modules (output fallback mode, for example).
- Loading the configuration onto the SIM card (if one is present),
- Generation of EDS files.

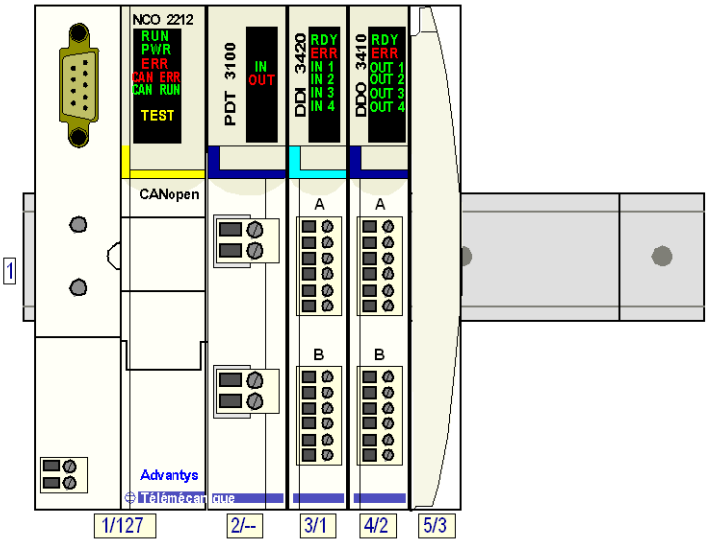
NOTE: In our example, only the last function is necessary because we do not use the SIM card and we are we are entirely satisfied with the default configuration *(see page 109)*.

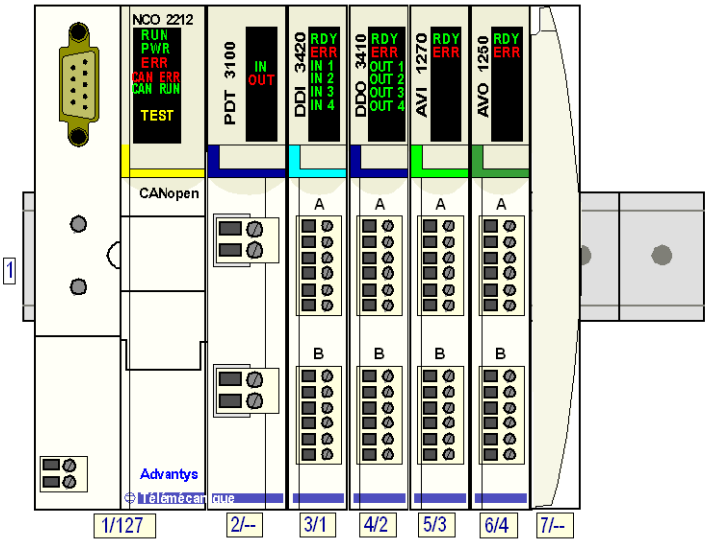
NOTE: The Advantys software is not mandatory as you can use the generic Sycon EDS files. Nevertheless, this possibility requires in-depth knowledge of the Sycon software.

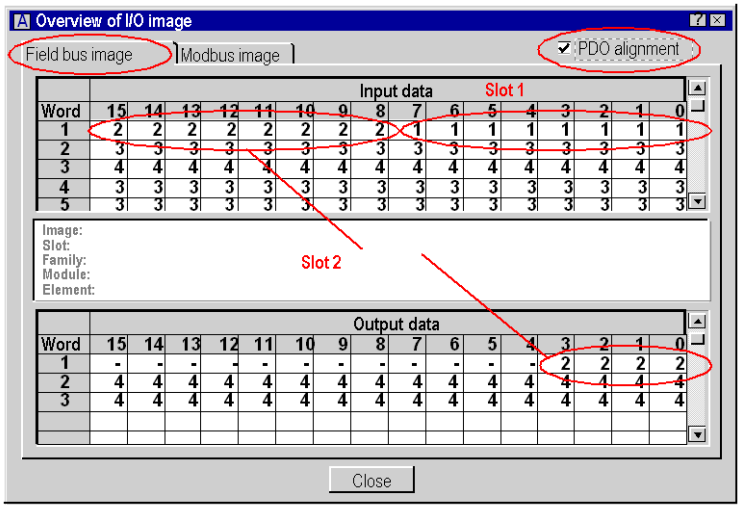
How to Create EDS Files

The table below describes the procedure for creating EDS files that can be used with Sycon.

Step	Action
1	Open the Advantys software and create a new workspace (**File →New workspace...**) by indicating: ● The name of the project (Quick start), ● The save path (`D:\Advantys_Projetcs`), ● The node name (Node_2) **Result**:  the name of the project file will be **Quick start.aiw** and the file name of the Advantys STB will be **Node_2.isl**. **Note**: All nodes for the same bus must be declared in the same project directory.

Step	Action
2	Configure **Node_2** by dragging the modules from the hardware catalog located on the right side of the screen. **Result**:  **Note**: Be careful not to forget the STB XMP 1100 bus terminator.
3	Add a new island from the **File** menu, whose name will be **Node_3**.

Step	Action
4	Repeat step **2** for **Node_3**. **Note**: If you have forgotten the power supply, as well as the STB XMP 1100 terminator, you cannot display the configuration in online mode. You must then add them manually. **Result**: 
5	Select **Node_3** in the menu **Island** →**I/O image preview** to display the I/O image.

Step	Action
6	Click on the **Field bus image** tab and check the **PDO alignment** box to display the elements of node 3. **Result**:  **Note**: We see that node 3 has 3 output words and 5 input words. Inputs for slot 1 (DDI module) are located in the least significant byte of word 1. Module I/Os located at slot 2 (DDO module) are located in input word 1 (most significant) and its outputs in the output word 1, etc. The general rules for mapping are explained in the remainder of this document.
7	Repeat step **6** for **Node_2** in order to display the elements of node 2.
8	Select node 2 and create the EDS file by selecting the **File →Export Node_2... option** and naming it **Node_2**. **Note**: In our example, the file will be saved in the following location: `D:\Advantys_Projects\Quick Start\Node_2.eds.`
9	Select node 3 and create the EDS file by selecting the **File →Export Node_3... option** and naming it **Node_3**. **Note**: In our example, the file will be saved in the following location: `D:\Advantys_Projects\Quick Start\Node_3.eds.`

General Rules for Mapping

The main rules for positioning date in the Advantys memory are:

- The first block is the Discrete input block, followed by the analog input block.
- In each block, the I/O bits are classified according to the physical position of the module.
- The I/O bits are sorted by their number on the module. I/O values will appear first, followed by the echo (for outputs only), and then the status bits.
- The analog I/O bits are also sorted by their number on the module. The I/O values are located in the analog block while the status bits are located in the Discrete input block.

Example of input data from node 3

Input data				
Word	Bits 15 to 12	Bits 11 to 8	Bits 7 to 4	Bits 3 to 0
1	Status bits slot 2	Output echo bits slot 2	Status bits slot 1	Input bits slot 1
2	Status byte slot 3		Status byte slot 3	
3	Status byte slot 4		Status byte slot 4	
4	Analog inputs slot 3			
5	Analog inputs slot 4			

Example of output data from node 3

Output data				
Word	Bits 15 to 12	Bits 11 to 8	Bits 7 to 4	Bits 3 to 0
1	-	-	-	Output bits slot 2
2	Analog outputs slot 4			
3	Analog outputs slot 4			

NOTE: the slot is the right-hand figure on the label at the bottom of each module.

Declaration of the CANopen Master Using Sycon and EDS Import

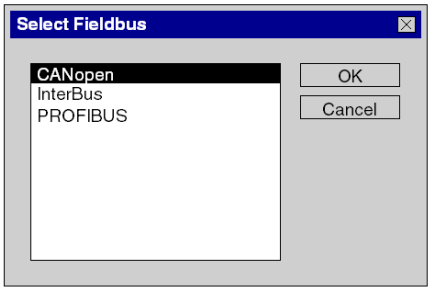
At a Glance

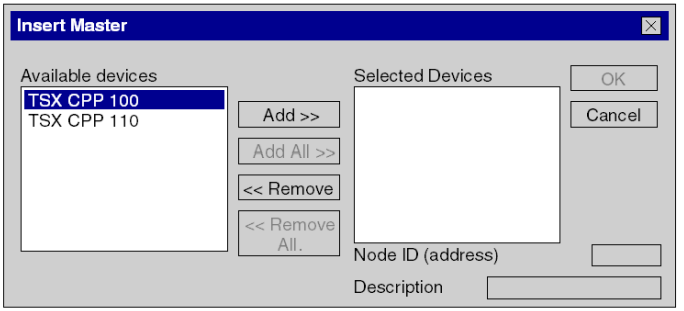
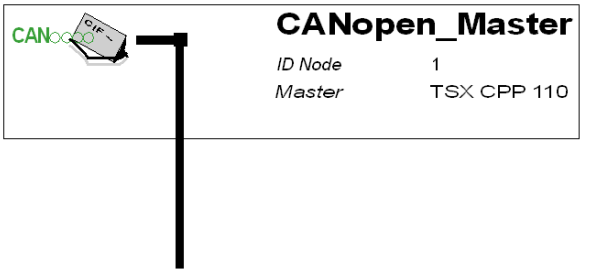
We will now use Sycon software to create the CANopen bus and generate the description of the bus that will be loaded onto the Premium PLC using Unity Pro software.

First, we will declare the bus master, then import the EDS files from nodes 2 and 3, so that they can be recognized by the Sycon software.

How to Declare the Master of the CANopen Bus

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

Step	Action
1	Start the Sycon tool from the Windows **Start** menu, or from the configuration screen of the TSX CPP 110 card (in Unity Pro software), click on the icon  **Result:** the SyCon tool will appear on the screen.
2	Select the command **File →New** **Result:** the following screen appears: 
3	Select **CANopen** then confirm by clicking **Ok**. **Result:** a blank structure will appear on the screen.

Step	Action
4	Select the command **Insert →Master**. **Result:** the following screen appears: 
5	● Select **TSX CPP 110**, ● Click on **Add**, ● Enter a name, which represents the master device, in the **Description** field, **Note:** The name should contain neither spaces nor letters with accents and should consist of no more than 32 characters. ● Confirm by clicking **Ok**. **Result:** the following structure appears: 
6	Save the CANopen project under the name Demo_cfg.co. **Note**: Make sure to remember the place where the .CO file is stored, as you will need to use the it again when loading the file into the configuration of the card running Unity Pro.

How to Import EDS Files

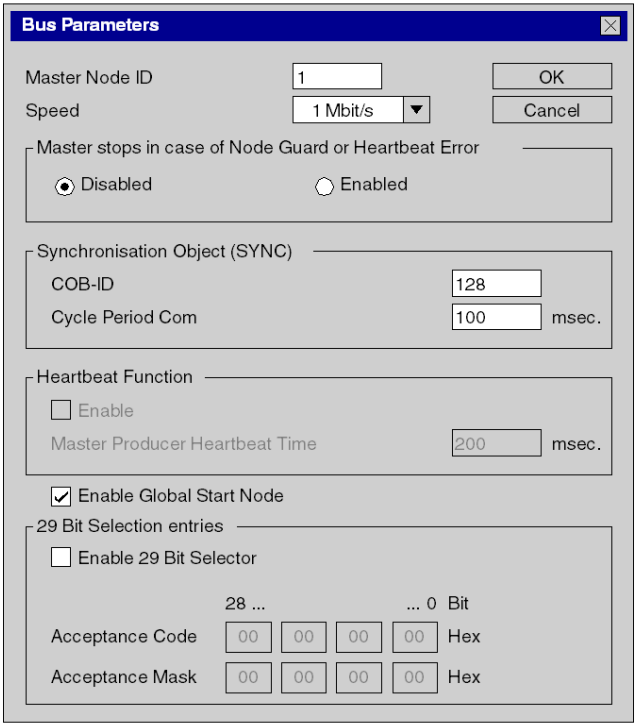
The table below shows the different stages for importing EDS files using Sycon.

Step	Action
1	Launch the Sycon software if it is not already running, in order to declare the bus master *(see page 122)*. **Result:** the SyCon tool will appear on the screen.
2	Select the **File** →**Copy EDS** option to import the EDS files using Sycon.
3	Select the **Node_2.eds** file from the `D:\Advantys_Projects\Quick Start` directory to import the EDS file from node 2. **Note**: Refuse the bitmap import, as this file was not created.
4	Repeat stages **2** and **3** for node 3 with the **Node_3.eds** file in the `D:\Advantys_Projects\Quick Start` directory to import the EDS file from node 3. **Note**: Refuse the bitmap import, as this file was not created. **Result**: the two EDS files are now available for insertion in the two CANopen slaves *(see page 126)*.

CANopen Bus Configuration

Procedure

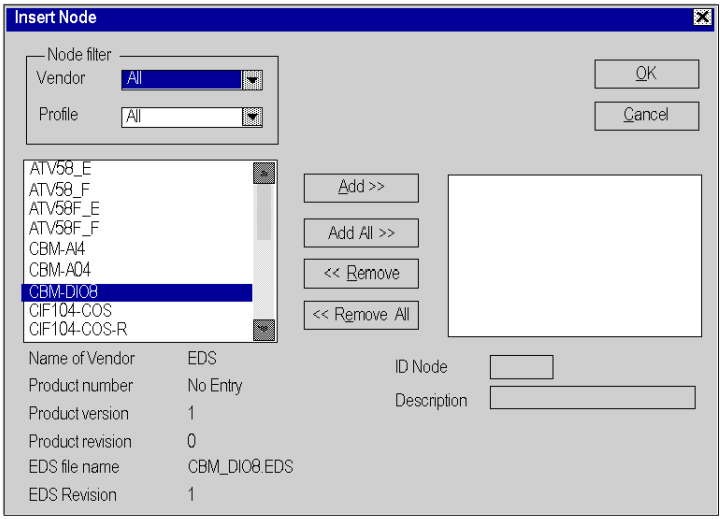
The table below shows the different stages for declaring the CANopen bus.

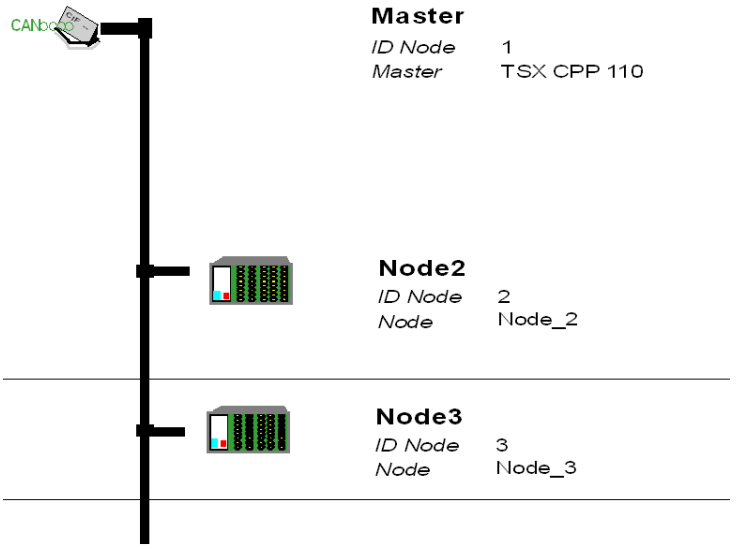
Step	Action
1	Select the command **Parameters →Bus parameters**. **Result:** the following screen appears: 
2	Configure: ● Speed to 250 Kbit/s, ● The value of **SYNC COB-ID** to 128 (default value), ● Cycle Period Com to 100 ms.
3	Select **Disabled** to **Master stops in case of Node Guard or Heartbeat Error**.
4	Select **Enable Global Start Node**.
5	Validate with **Ok**.

Declaration of Nodes 2 and 3

Procedure

The table below shows the different stages to declare nodes 2 and 3.

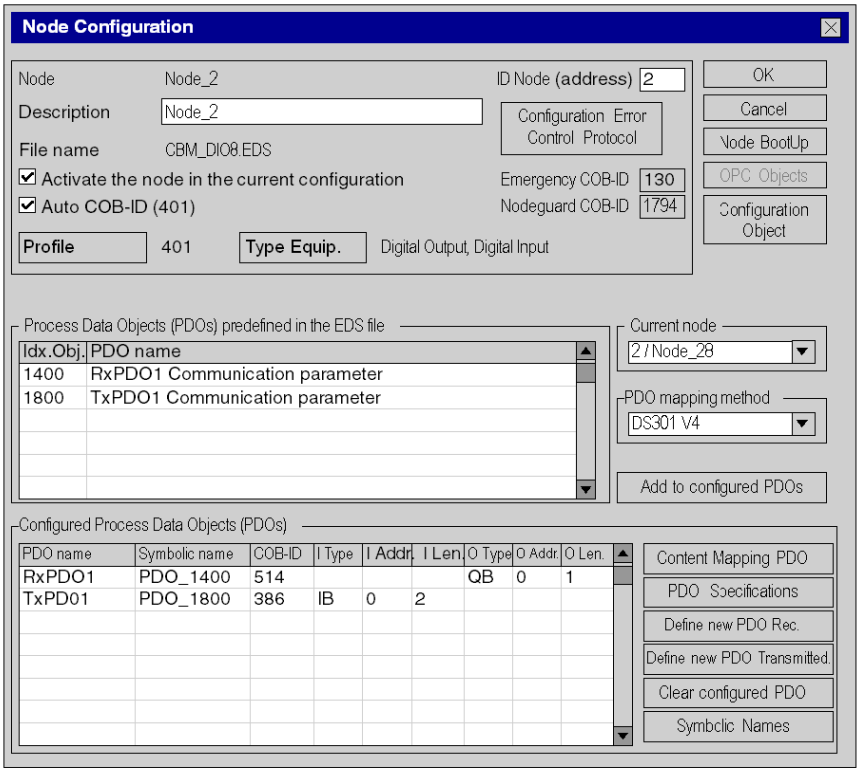
Step	Action
1	Select the command **Insert →Node**. **Result:** a cursor appears:
2	Place the cursor on the bus outside of the text frame that defines the master and click once. **Result:** the following screen appears: 
3	Select **Node_2** in the list of available devices.
4	Click on the **Add** button to insert **Node_2** in the list of devices selected.
5	Select **Node_3** in the list of available devices.

Step	Action
6	Click on the **Add** button to insert **Node_3** in the list of devices selected and confirm with **OK**. **Result:** the following structure appears: 

Configuration of Nodes 2 and 3

How to Configure Node 2

The table below shows the different steps for configuring node 2.

Step	Action
1	Double-click on node 2. **Result:** the configuration screen appears and displays the grid of the two PDOs (Predefined Process Objects) of the node. A reception PDO (RxPDO) used to configure the outputs and a transmission PDO (TxPDO) used to configure the inputs.
2	Double-click on the first line (PDO RxPDO1) to display the window containing the PDO specifications.
3	Confirm with **OK**, as we will use the default configuration. Result: the PDO now appears in the **Configured Process Data Objects (PDOs)** area.
4	Repeat steps **2** and **3** for the second PDO. **Result:** the screen should be configured as follows: 

Step	Action
5	Click OK to confirm the configuration of node 2. **Note**: As the PDOs have been configured, the Sycon software now possesses all the configuration information for node 2 in order to use the corresponding EDS file.

How to Configure Node 3

The table below shows the different steps for configuring node 3.

Step	Action
1	Double-click on node 3. **Result:** the configuration screen appears and displays the grid of the four PDOs (Predefined Process Objects) of the node. Two reception PDOs (RxPDO) that can be used to configure the outputs and two transmission PDOs (TxPDO) that can be used to configure the inputs.
2	Double-click on the first line (PDO RxPDO1) to display the configuration window of the first PDO.
3	Confirm with **OK**, as we will use the default configuration.
4	Repeat steps **2** and **3** for the 3 other PDOs.
5	Click on the **Configure object** button to activate transmission of analog inputs. **Note**: By default, transmission of analog inputs is disabled; it is therefore necessary to activate this functionality.
6	Scroll down to the bottom of the scroll bar in the **Predefined objects specified in the EDS file** area and double-click on the **6423 0 Analog Input Global Enable** line in the list of supported predefined objects.
7	In the **Chosen value** column, enter the value **1**.
8	Click **OK** to confirm the configuration the objects.
9	Click **OK** to confirm the configuration of node 3. **Result**: node 3 is now configured. If you have also configured node 2, you can save the Sycon project, which can now be used in the PLC, under the name `Demo_cfg.co`.

Configuration of the PCMCIA TSX CPP 110 Card

At a Glance

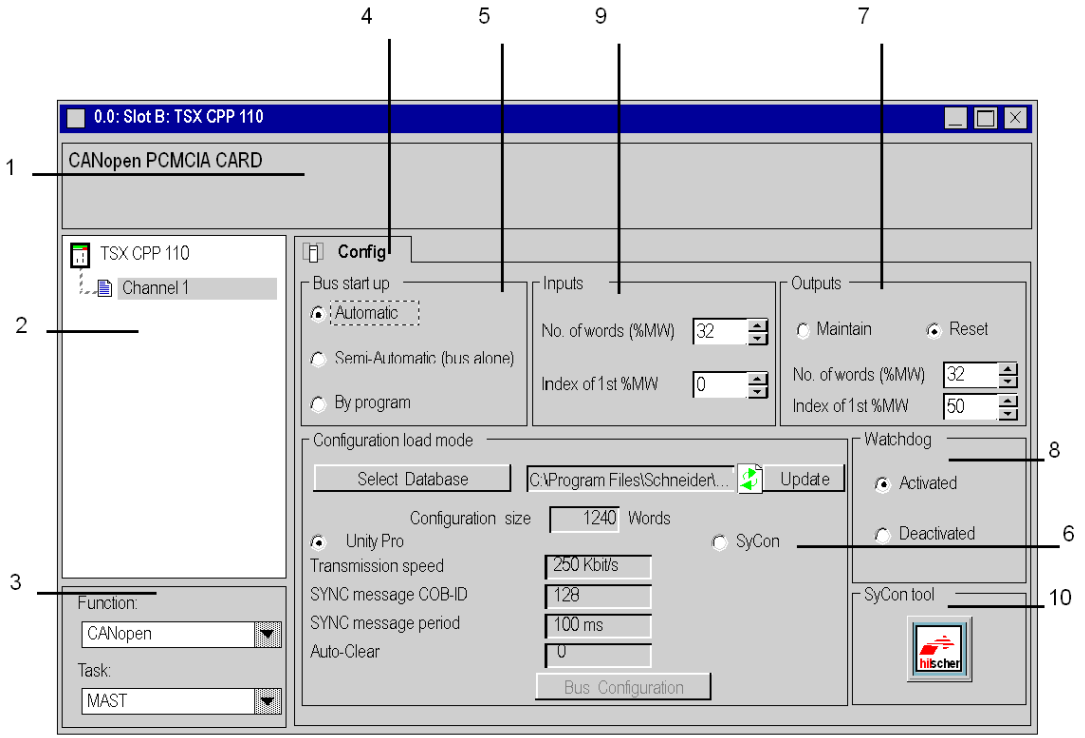
Once you have created the EDS files and the CANopen bus using the Sycon software, you must now declare and then configure the TSX CPP 110 card in the PLC.

To declare the card, please refer to the corresponding paragraph *(see page 37)* in this documentation.

The configuration of the card in this example is shown in the figure below. If you wish to obtain more details concerning the possible configurations, please refer to the corresponding paragraph. *(see page 130)*.

Illustration

The screen below shows the configuration parameters required to implement our example:



Elements and Functions

This table describes the various zones that make up the configuration screen:

Zone	Number	Function
Module	1	This zone comprises the abbreviated title of the PCMCIA card that you declared, which must be a CANopen card.
Channel	2	This zone allows you to select the communication channel to be configured. Click on the channel to obtain the configuration tab.
General parameters	3	In this zone, select the task associated with the I/Os located on the CANopen bus. The selected task will determine the acquisition rate of the inputs and update of the outputs of the CANopen bus slaves. In our case, we will select the master (MAST) task.
Tab	4	The tab in the foreground indicates the type of screen displayed. In our case it is the configuration screen. If you are in online mode, you will be able to access other tabs. In this case, check that you have clicked on the **Config.** tab to obtain the screen shown in the illustration.
Config	5	This zone is used to select how the bus is to behave at start-up. Select **Automatic**.
	6	This zone is used to configure the bus. Click on the Unity Pro button in order to access the configuration using Unity Pro. Select the `Demo_cfg.co` file that you created with the Sycon software. **Result**: the bus parameters are displayed when a .CO file is selected. **Note**: If you modify the .CO file using Sycon, click the **Update** button in order to reload it into the configuration. **Note**: If you wish to display the list of slaves on the bus, click the **Bus configuration** button.
	7	This zone is used to configure the fallback mode for bus device outputs as well as the address (PLC internal memory) where the outputs from CANopen devices will periodically be read. In our example, we select **Reset** and a number of **32 words** starting at the address **50** (%MW50 to %MW81). Later on we will see that these addresses correspond to topological objects (that depend on the slave in which they are located).
	8	This zone is used to activate or deactivate the CANopen bus watchdog. The watchdog is activated by default. It is triggered when the PCMCIA card can no longer manage the bus correctly. When it is triggered, it makes all the slaves' outputs change to zero. In our example, we will leave it activated.
	9	This zone is used to configure the address (PLC internal memory) to which inputs from the CANopen devices will periodically be copied. In our example, we will retain the default values of 32 and 0 to indicate that the inputs will be stored in the words %MW0 to %MW31. Topological addressing is also available as for the outputs.
	10	This button is used to start the **Sycon** software, if it is installed on the PC.

⚠ WARNING

UNEXPECTED APPLICATION BEHAVIOR

Before deactivating the watchdog, ensure that, if the PCMCIA card does not manage the CANOpen bus, then the devices behavior remain acceptable.

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

Confirmation and Generation

Once you have completed the different fields in the configuration screen, you must:

- confirm the configuration
- generate the project.

When these two operations have been carried out, you can save your .STU file and transfer your project to the PLC.

I/O Distribution

The inputs and outputs configurated in the .CO file are distributed in the following manner:

- %MW0 contains inputs for node 2.
- %MW50 contains outputs for node 2.
- %MW1 to %MW5 contain inputs for node 3.
- %MW51 to %MW53 contain outputs for node 3.

I/O Topological Distribution

If you click the **Bus configuration** button in the configuration screen, you can display the two CANopen bus nodes and thus obtain the I/O topological addressing.

For node 2, we have the following inputs:

- %IW\3.2\0.0.0.0 which corresponds to %MW0.

and the following outputs:

- %QW\3.2\0.0.0.0 which corresponds to %MW50.

For node 3, we have the following inputs:

- %IW\3.3\0.0.0.0 which corresponds to %MW1,
- %IW\3.3\0.0.0.1 which corresponds to %MW2,
- %IW\3.3\0.0.0.2 which corresponds to %MW3,
- %IW\3.3\0.0.0.3 which corresponds to %MW4,
- %IW\3.3\0.0.0.4 which corresponds to %MW5.

and the following outputs:

- %QW\3.3\0.0.0.0 which corresponds to %MW51,
- %QW\3.3\0.0.0.1 which corresponds to %MW52,
- %QW\3.3\0.0.0.2 which corresponds to %MW53.

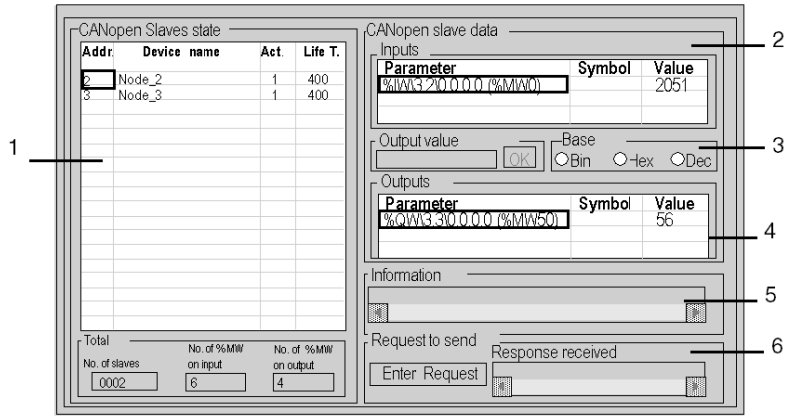
Debug

At a Glance

The debug screen can only be accessed in online mode. It can be used to display bus, slave and CANopen master operation.

Illustration

The figure below is extracted from the debug screen for our example.



Description

The table below shows the different zones of the debug screen:

Number	Element	Function
1	CANopen Slaves state	This zone displays all the slaves (also called nodes) on the CANopen bus. A faulty slave is displayed in red, when the fault disappears, it is displayed in blue. If there is no fault, it is displayed in black. Selecting a slave updates zones 2, 4, 5 and 6. **Act.**: indicates whether the slave was activated in the Sycon configuration (1=activated, 0=deactivated) **Life T.**: Life Time. In our example, we display two slaves (or two nodes); these are the two nodes of our Sycon configuration. **Note**: When a node is displayed in red, you can obtain the causes of the error by clicking on it. The information line is automatically updated with diagnostic information.

Number	Element	Function
2	Inputs	When a slave is selected, this zone contains the list of words, which are associated with it on input. A clearer explanation is given with the display of the CANopen topological address *(see page 35)* (%IW\3.2\0.0.0.0) and reserved memory area in the PLC (%MW0). If node 2 is selected, then the previous illustration is displayed; if node 3 is selected, the screen changes and displays the %IW of node 3.
3	Output value	When an output word in zone 4 is selected, its value can be modified by entering a new value then clicking on the **OK** button.
4	Outputs	When a slave is selected, this zone contains the list of words that are associated with it on output. A clearer explanation is given with the display of the CANopen topological address *(see page 35)* (%QW\3.2\0.0.0.0) and reserved memory area in the PLC (%MW50). If node 2 is selected, then the previous illustration is displayed; if node 3 is selected, the screen changes and displays the %QW of node 3.
5	Information on	When a slave is selected (click in zone 1), this zone contains its last diagnostic message, and to obtain information on the TSX CPP 110 card, simply click on the table header.
6	Request to send	This zone is used to send an SDO request. Parameter syntax is identical to that used to transfer SDOs using READ_VAR and WRITE_VAR requests *(see page 54)*. Pressing the **Enter Request** button makes the request input fields appear.

Appendices

Appendix A
Configuration Example for Devices on the CANopen Bus

Subject of this Chapter

This chapter presents three configuration examples for devices on the CANopen bus.

- An Altivar ATV 58,
- A Lexium drive,
- A situation requiring more than 4 PDOs per device.

What Is in This Chapter?

This chapter contains the following topics:

Configuration of an Altivar Variable Speed Controller

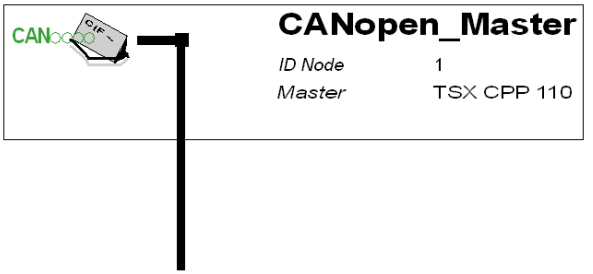
At a Glance

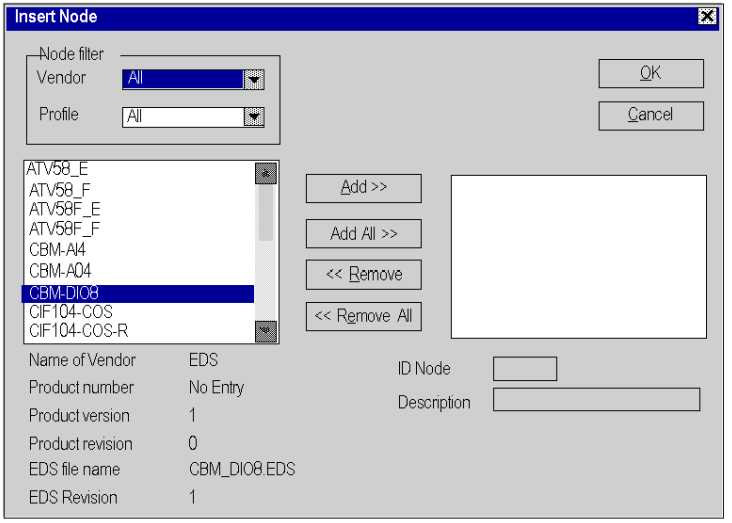
You can configure an Altivar on the CANopen bus very easily. We will see how to carry out this configuration.

Before reading the following paragraphs, we recommend that that you perform the Advantys STB implementation on a CANopen bus *(see page 105)* example described in this manual, or that you read these implementation pages, as we will refer to them on several occasions.

How to Configure an Altivar

The table below shows the different steps for configuring an Altivar.

Step	Action
1	Launch the Sycon software and create a new CANopen project *(see page 122)*. **Result:** you should obtain bus architecture of this type: 
2	Select the command **Insert** →**Node**. **Result:** a cursor appears:

Step	Action
3	Place the cursor on the bus outside of the text frame that defines the master and click once. **Result:** The following screen appears: 
4	Select the Altivar EDS file that you would like to incorporate into the bus and click on **Add**. **Note:** Four Altivars are available: ● ATV58_E: EDS files in English, ● ATV58_F: EDS files in French, ● ATV58F_E: EDS files in English, ● ATV58F_F: EDS files in French,
5	Confirm your selection with **OK**. **Result**: the bus architecture shows a new node made up of a drive. You must now configure the PDOs of this drive.
6	Double click on the drive node to display the PDO window. **Result**: a screen resembling the Advantys STD screen *(see page 140)* appears, allowing you to configure the two PDOs for the drive: ● Transmission PDO: 1 status word and 1 word for current speed, ● Reception PDO: 1 command register word and 1 word for commanded velocity.
7	Double click on each PDO and confirm the default transmission parameters.
8	Check that the node speeds and its address correspond to those that you set in the Altivar. **Note**: If Altivar is the last element of the bus, do not forget to activate the bus end termination by setting the switch located on the Altivar CANopen communication card.
9	Click OK to confirm the node configuration.
10	Save the .CO file.

Step	Action
11	Open your Unity Pro project that contains the master of the CANopen bus (TSX CPP 110 card), declare the card, import the saved .CO file, generate the project and transfer it to the PLC as in the Advantys STD example *(see page 130)*. **Result**: the CANopen bus and your Altivar are operational.

Tests

You can carry our several tests in order to check whether the Altivar and the bus are operating correctly by opening your TSX CPP 110 card debug screen and by performing the following actions:

- You can start the motor by entering the values 6, 7, 15 in the command register (first output word).
- You can set the motor's rotational speed by modifying the value of the second output word.
- If Altivar faults, you must carry out a reinitialization by entering the value 128 in the command register (first output word), and then repeating sequences 6, 7 and 15.

NOTE: Once you have checked that your Altivar was successfully commanded by the modification of words in the PLC automatically transmitted to it on the CANopen bus, you can implement your automation project.

ATV 31

The ATV 31 variable speed controller is also CANopen-compatible. For details concerning its configuration, please refer to the ATV 31 manual, CANopen Communication Profile.

Configuration of a Lexium Drive

At a Glance

You can configure a Lexium drive on the CANopen bus very easily. We will see how to carry out this configuration.

Before reading the following paragraphs, we recommend that that you perform the Advantys STB implementation on a CANopen bus *(see page 105)* example described in this manual, or that you read these implementation pages, as we will refer to them on several occasions.

Connecting the Lexium

The Lexium is connected to the CANopen bus via the AMO 2CA 001 V000 adapter:

- Screwed onto the Lexium, then
- Connected to the CANopen Bus cables.

NOTE: For implementation of any other connectors, please refer to the **Lexium CANopen** manual that you will find on the **Motion Tools 3.0 CD ROM**.

Description of Lexium PDOs

As with Altivar, the PDOs allow implicit management of the Lexium via PLC words (%MW or topological addressing %IW and %QW *(see page 35)*).

The Lexium uses two types of PDOs:

- Predefined PDOs:
  - These PDOs are associated with commands specific to the Lexium.
    Example: the PDO 22 of this appendix is associated with the request 16#6040 for the command word and the request 16#2060 for the **Feed rate or current**.
- Free PDOs:
  - These PDOs are not associated with commands specific to the Lexium. They are configured by the application. We will not describe these PDOs in this example; for additional details, please refer to the Lexium CANopen manual.

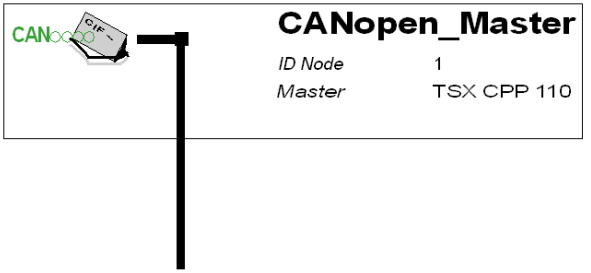
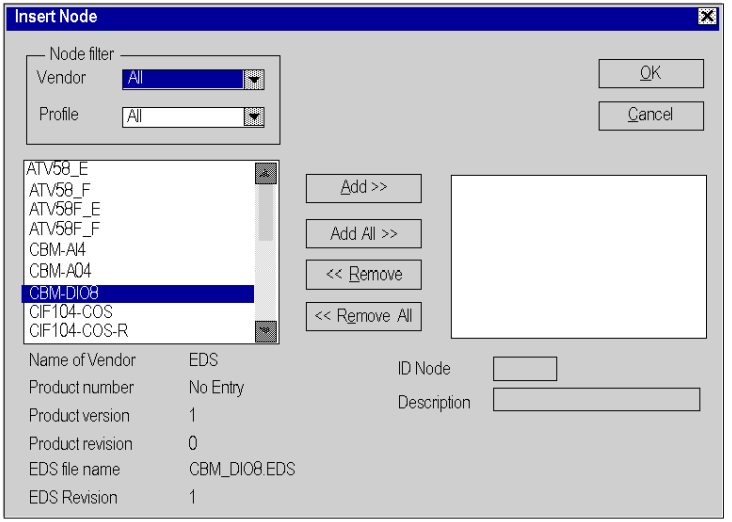
The PDO are exchanged using two predefined channels:

- 3 reception channels (16#2600, 16#2601, 16#2602)
- 3 transmission channels (16#2A00, 16#2A01, 16#2A02).

NOTE: The maximum data size is 8 bytes per channel and the Lexium version must be higher to or equal to version 5.51.

How to Configure a Lexium PDO

The table below shows the different steps for configuring a reception PDO, the Lexium PDO 22, on channel 1. .

Step	Action
1	Launch the Sycon software and create a new CANopen project *(see page 122)*. **Result:** you should obtain bus architecture of this type: 
2	Select the command **Insert** →**Node**. **Result:** a cursor appears:
3	Place the cursor on the bus outside of the text frame that defines the master and click once. **Result:** the following screen appears: 
4	Select the file EDS LEXIUM17D and click on **Add**.

Step	Action
5	Confirm your selection with **OK**. **Result**: the bus architecture shows a new node made up of a Lexium 17D. You must now configure the PDO 22 of this drive.
6	Double-click on the drive node to display the PDO window. **Result**: a screen resembling the Advantys STB screen *(see page 144)* appears, allowing you to configure the two PDOs for the drive:
7	Select the first reception PDO.
8	Click on the **Add to configured PDOs** button and confirm the default transmission parameters by clicking on **OK**. **Result**: a new line appears in the list of configured PDOs.
9	Click on this line, then on the **Content Mapping PDO** button. **Result**: a new window appears.
10	Select the Idx Obj. 2060 (Feed rate or current).
11	Click on the **Add Object** button, then on **OK** to confirm. **Result**: the requests 16#6040 and 16#2060 are associated with the PDO. You must now configure the PDO.
12	Select the first reception PDO again, then click on the **Configure Object** button. **Result**: the PDO configuration window appears.
13	Scroll down through the list of **Predefined objects specified in the EDS file** and select the **1st receive PDO select** line of the predefined object 2600.
14	Click on the **Add to configured obj** button. **Result**: the object appears in the **Objects configured automatically at node start-up** window.
15	Enter 22 in the corresponding box of the **Chosen value** column.
16	Save the .CO file in order to be able to import it into the configuration of the TSX CPP 110 card using Unity Pro.
17	Open your Unity Pro project that contains the master of the CANopen bus (TSX CPP 110 card), declare the card, import the saved .CO file, configure the card with the default values, generate the project and transfer it to the PLC as in the Advantys STB example *(see page 130)*. **Result**: the CANopen bus and your Lexium are operational.

Tests

You can carry our several tests in order to check whether the Lexium and the bus are operating correctly by opening your TSX CPP 110 card debug screen and by performing the following actions:

- If you used the default configuration of the TSX CPP 110 card, the word %MW32 (first output word) corresponds to the Lexium control word; you can modify it and check the consequence on the Lexium.
- Similarly, the double word %MD32 lets you access the speed.

NOTE: Once you have checked that your Lexium was successfully commanded by the modification of words in the PLC automatically transmitted to it on the CANopen bus, you can implement your automation project.

Configuration of More than 4 PDOs per Node

At a Glance

For each node, 4 transmission PDOs and 4 reception PDOs can be configured. This is because the Sycon software automatically generates the COB-IDs to guarantee their unicity and uses an algorithm respecting profile 301. The following paragraph explains this operation.

To be able to use more than 4 PDOs in each direction, you must use a manual procedure like the one described for creating a 5 th PDO.

How the COB-IDs are Assigned

Possible COB-IDs are between the values of 385 and 1407 (16#180 and 16#57F).

You can use the COB-ID of your choice for each PDO; however, the COB-IDs must be unique in this interval of values. The CANopen configuration tools such as Sycon are configured by default to provide a COB-ID that respects these constraints.

The following table describes the algorithm used by the Sycon software to carry out these COB-ID assignments.

PDO	ID	Node 1 (decimal)	Node 2 (decimal)	...	Node 127 (decimal)
1.TXPDO	16#180+ID-Node	385	386	...	511
1.RXPDO	16#200+ID-Node	513	514	...	639
2.TXPDO	16#280+ID-Node	641	642	...	767
2.RXPDO	16#300+ID-Node	769	770	...	895
3.TXPDO	16#380+ID-Node	897	898	...	1023
3.RXPDO	16#400+ID-Node	1025	1026	...	1151
4.TXPDO	16#480+ID-Node	1153	1154	...	1279
4.RXPDO	16#500+ID-Node	1281	1282	...	1407

As the COB-ID determines the priority of the message (the weaker its value, the greater its priority), the consequences are the following:

- The first PDO in the node has greater priority than the second, third and fourth.
- A transmission PDO has priority over a reception PDO in the same node.
- The lower the node number, the greater the priority of the node's PDOs.

How to Create a Number 5 PDO

The table below describes how to create a 5th PDO for a given node.

Step	Action
1	Configure a PDO5 from the node configuration screen. **Result**: a warning window appears to indicate that it is not possible to create a new PDO. This is normal, given what was said previously (a maximum of 4 PDOs per node).
2	In the node configuration screen, disable the Auto **COB-ID** option. **Result**: you can now write a new COB-ID for the PDOs that you want to create and thus define the new priorities in the messages transmitted on the CANopen bus. **Caution**: you must respect the unicity of the COB-IDs. The numbers available are: ● TXPDO: 1664 to 1759, ● RXPDO: 1761 to 1792.
3	Close the node configuration window and save your project.

Glossary

C

CAN

Controller Area Network: field bus originally developed for automobile applications which is now used in many sectors, from industrial to tertiary.

CiA

CAN in Automation: international organization of users and manufacturers of CAN products.

COB

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

CRC

Cyclic Redundancy Checksum: cyclic redundancy checksum indicates that no character has been "deformed" during frame transmission.

CSMA/CA

Carrier Sense, Multiple Access / Collision Avoidance: communication control method on a network featuring the link layer.

D

DIN

Deutsches Institut für Normung: German standardization institute.

DS

Draft Standard: specifications document created by the CIA organization.

E

EDS

Electronic Data Sheet: description file for each CAN device (provided by the manufacturers). To add a CAN device to the bus with Sycon configuration software, select the corresponding EDS. The EDS are available on the website **http://www.can-cia.de** or from the hardware provider.

L

Life Time
Life Time = Life Time factor x Guard Time.

LLC
Logical Link Control.

M

MAC
Medium Access Control.

MDI
Medium Dependent Interface.

MTBF
Mean Time Between Failure: mean time between two failures.

O

OD
Object Dictionary: object dictionary recognized by CAN. A hexadecimal code is given to each objet type, the dictionary regroups all the object's codes.

P

PCMCIA
Personal Computer Memory Card International Association

PDO
Process Data Object: there are RPDOs (Receive PDO) and TPDOs (Transmit PDO).

PDU
Process Data Unit: there are APDUs (Application PDUs). A PDU on the link layer is an APDU encapsulated by headings and bytes, which characterize this link layer.

PMA
Physical Medium Attachment.

S

SDO
Service Data Object: there are SSDOs (Server SDO) and CSDOs (Client SDO).

T

TAP

Transmission Access Point: the bus connection unit.

Index