



RightWON CANbus/CANopen Protocols - Manual - V1.3

Document No. RWM002015-MA-en
©2012 Vizimax, Inc. All rights reserved.





Copyright

© Copyright Vizimax, Inc., 2012. All rights reserved.

The information contained in this document is the proprietary and/or confidential information of Vizimax, Inc. ("Vizimax") and is subject to all relevant laws protecting intellectual property and confidential information, as well as to the terms of any specific agreement protecting Vizimax's rights in such information. Neither this document nor the information contained herein may be published, reproduced, transmitted or disclosed in whole or in part by any means for any purpose without the express, prior, written authorization of Vizimax. In addition, any use of this document or the information contained herein for any purposes other than those for which it was disclosed is strictly forbidden.

Vizimax may have patents or pending patents applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not constitute a license to these patents, trademarks, copyrights, or other intellectual property.

Document Disclaimer

Vizimax believes the information in this document to be accurate at the time of publication of the document. However, this document may contain errors or omissions. Vizimax makes no warranties regarding this document or its content. Vizimax is not responsible for any loss, liability, or damages of any kind arising from or related to this document or the information contained in it. Information contained herein may be periodically updated or modified without notice in subsequent editions. If you encounter an error in this document, please contact Vizimax.

Any representations or statements in this document concerning Vizimax products are for informational purposes only and do not constitute warranties, either express or implied, regarding such products. Vizimax's standard limited warranty, stated in its sales contract or order confirmation form, is the only warranty offered by Vizimax in relation to the products.

All specifications and designs are subject to change without prior notice. Vizimax reserves the right, at its sole discretion, to modify or replace any part of this document. It is your responsibility to check this document periodically for changes.

Warranty Disclaimer

Vizimax and its suppliers and licensors hereby disclaim all warranties of any kind, express or implied, including, without limitation, the warranties of merchantability, fitness for a particular purpose and non-infringement. Neither Vizimax nor its suppliers and licensors, makes any warranty that Vizimax products will be error-free, or that access to remote units and their connected equipment will be continuous or uninterrupted.

Limitation of Liability

Vizimax products are programmable devices which can be modified by anyone who has been given authorization to access the product and log onto its configuration software. Vizimax cannot oversee modifications to the configuration of Vizimax products unless a prior service agreement has been made between all parties involved. Vizimax does not have any control over who is given permission to access Vizimax products. Vizimax cannot therefore be held responsible for the configuration, automatism and actions that are programmed into any Vizimax product once it has been delivered to the purchaser or third party. Likewise, Vizimax is not responsible for the particular use of Vizimax products in industrial, commercial or other applications, nor is it liable for any harmful effects of such use.

You are responsible for taking precautions as necessary to protect yourself, your computer networks and all connected equipment from any harmful or destructive actions that may arise due to incorrect programming of Vizimax products, or any purposely destructive actions programmed by a person with malicious intent. Vizimax disclaims any responsibility for any harm resulting from the use of the RightWON.

In no event will Vizimax, or its suppliers or licensors, be liable with respect to any subject matter under any contract, negligence, strict liability or other legal or equitable theory for: (i) any special, incidental or consequential damages; (ii) the cost of procurement or substitute products or services; (iii) interruption of use, or the loss or corruption of data.

Vizimax, its contractors, licensors, and their respective directors, officers, employees and agents, are hereby indemnified from any and all claims and expenses, including attorneys' fees, arising out of your use or misuse of Vizimax products. Vizimax shall have no liability for any damage, injury, failure or delay due to matters beyond the reasonable control of Vizimax.

The foregoing shall not apply to the extent prohibited by applicable law.

Trademarks

Vizimax, the Vizimax logo, RightWON, WiseWON and the RightWON icons are trademarks or registered trademarks of Vizimax, Inc. in Canada, the United States and other jurisdictions. All other trademarks, registered trademarks and service marks are the property of their respective owners.

Your use of Vizimax products grants you no right or license to reproduce or otherwise use any Vizimax or third-party trademarks.

Vizimax is a licensed user of the following service marks:





Contents

Introduction	1
1.1. Document scope	1
1.1.1. Other documents.....	1
1.2. Document conventions.....	1
1.3. Safety precautions	1
1.3.1. Warnings ⚠.....	2
1.3.2. Cautions ⚠.....	2
Managing the CANbus – CANopen Protocol.....	3
2.1. Introduction to the CANbus protocol	3
2.1.1. Node	3
2.1.2. Message and message identifier (COB-ID).....	3
2.1.3. Bus arbitration	3
2.1.4. Implementation levels.....	3
2.1.5. CANopen protocol features.....	4
2.1.6. CANopen system support.....	5
2.1.7. Function of PDOs associated with digital inputs/outputs	6
2.1.8. Function of PDOs associated with analog inputs/outputs.....	7
2.1.9. Updating digital and analog outputs	8
2.1.10. Updating variables from the digital and analog inputs	10
2.1.11. Configuring the CANopen controllers.....	11
2.1.12. PDO addressing and assigning additional PDOs to a node	11
Integrating the CANbus Protocol into the RightWON	13
3.1. Configuring CANopen I/Os.....	13
CANbus - CANopen Configuration Tutorial	15
4.1. Creating a new RightWON project.....	15
4.2. Adding and configuring a CANbus link	15
4.3. Adding the CANbus protocol driver	16
4.3.1. CANbus driver setup	17
4.4. Inserting and configuring a port	18
4.5. Creating the TPDOS and RPDOS for data exchange	19
4.6. Configuring the PDO exchange modes.....	21
4.6.1. Digital outputs	21
4.6.2. Analog outputs.....	22
4.6.3. Digital inputs	22
4.6.4. Analog inputs.....	23
4.7. Creating periodic RTR requests	23
4.8. Inserting a variable into a PDO.....	24
4.8.1. Associating variables with digital inputs/outputs.....	24
4.8.2. Associating variables with analog inputs/outputs	26
4.9. Creating a CANopen NMT message	28
4.10. Starting the nodes with a CANopen NMT message.....	29
4.11. Launching the project	30
4.11.1. Preparing the CANopen coupler.....	30
4.11.2. Preparing the RightWON application	30
4.11.3. Preparing the project.....	31

4.11.4. Bringing the project online	32
4.11.5. Operation of the CANbus example	32
4.11.6. Exiting from online mode	33
Advanced CANopen Protocol Management.....	34
5.1. Dictionary management (SDO)	34
5.1.1. Defining SDO messages in the CANbus driver	35
5.1.2. Adding a CANopen slave.....	35
5.1.3. Adding a message ("Insert Slave/Data block")	36
5.2. Example of using SDO requests for initialization	38
5.2.1. CANopen configuration	38
5.2.2. Startup program (pStartup)	39
5.2.3. CANopen management program	39
5.3. Adding more PDOs to a CANopen configuration	40
5.3.1. SDO read/write commands on inputs/outputs.....	40
5.3.2. PDO mapping registers.....	41
5.3.3. Assigning an identifier to a PDO.....	42
5.3.4. PDO definition and assignment sequence	43
5.4. Description of the CAN_PDO_Example project	43
5.4.1. CANopen configuration	43
5.4.2. Definitions	44
5.4.3. Startup program (pStartup)	44
5.4.4. Command recipe	45
5.4.5. CANopen management program	45
5.5. Using the CANSNDMSG and CANRCVMSG functions.....	49

Revision History

Date (yy-mm-dd)	Comments	Author
2011-08-08	V1.3: Initial release	P. Taillefer, J. Warne

Document Applicability

Document version	Product version	Comments
V1.0 and up	1.7 and higher	⚠ Applies to RightWON software version 1.7 and higher.

⚠ Important Notice

It is recommended that users of CANopen systems upgrade to RightWON firmware version 1.7 or higher, since the following problems have been detected in earlier versions:

- When multiple read requests for analog data on TPD02 to TPD04 data are launched simultaneously, the variables may not refresh properly. Periodic or on-demand read requests must be sequenced.
- 4-byte write requests (SDOs) do not execute properly. Commands affecting module initialization, PDO assignments and mapping may not work properly.



Introduction

Thank you for the trust you place in us, and congratulations for choosing the RightWON system from Vizimax! For your satisfaction and the success of your projects, RightWON system has been designed and manufactured to the highest quality and performance standards in the industry.

1.1. Document scope

This document describes the installation and configuration of a CANbus-CANopen protocol in a RightWON system using the RightWON Configuration Suite.

1.1.1. Other documents

For further details on the information in this document, refer to the specific manuals below:

Document No.	Document Name
RWM000010-MA	RightWON Configuration Suite - Manual
RWM000011-MA	RightWON Configuration Suite – Installation Guide
RWM000020-MA	RightWON – Operation Guide
RWM000050-MA	RightWON Satellite – User Manual
RWM000080-MA	RightWON Configuration Suite – Application Guide

1.2. Document conventions

To facilitate the reading of this document the following conventions are used:

- Menu/dialog controls and items are in **bold**, e.g. **Options/Advanced settings...**, as are buttons, e.g. **OK**
- Names of Categories, Users, Sectors and Tags defined by the system integrators are in *italics*, e.g. *John Smith, Generator*
- Application-specific items such as Sector, Tag, Category and User group begin with a capital letter
- [Hyperlinks](#) are in blue
- The ⚠ symbol is used to raise the reader's attention.

1.3. Safety precautions

To ensure the safety of personnel and products, and to prevent the risk of accident, you must strictly follow the cautions and warnings written on product labels, in the manuals and on the RightWON product packaging.

To ensure proper operations of the RightWON product, read this manual in its entirety before proceeding to the other stages of learning, hardware installation, configuration or operation. Make sure that you fully understand the product and all information provided in this manual. For further information or if you require assistance from Vizimax, write to our application engineers or the Technical Support group at support@vizimax.com (certain fees and conditions may apply, depending on the type of service requested).

1.3.1. Warnings

RightWON products are not designed for safety management applications or as security devices. Mishandling of this product could cause critical situations leading to personal, equipment or property damage, network failure, loss of data, electrical shock, serious injury or even death. To prevent such events from occurring:

- Take all possible measures to ensure the security of your systems through the use of appropriate equipment that meets the requirements of the application. This will help preserve the integrity of your systems in the event of product failure or other external factors.
- To prevent the risk of explosion, do not use RightWON products in areas where explosives are stored without taking appropriate measures as defined by the standards and regulations in effect, obtained from the proper local authorities.
- To prevent damage to electronic components, do not expose this product to open flame or submit it to environmental factors that exceed the recommended levels.
- Batteries may explode if they are not handled with care. Do not recharge, disassemble or dispose of in fire. We recommend that you recycle these items by taking them to the appropriate collection service.

1.3.2. Cautions

- Make sure that RightWON products are managed by qualified personnel who have been properly trained to install, configure and troubleshoot them.
- Always configure and operate this product within the recommended technical specifications and operating criteria recommended by Vizimax, as cited in this manual and the other technical documents available.
- Use homologated external emergency devices, including but not limited to: emergency stop, emergency signaling, interlock and safety circuitry.
- Properly connect and secure removable cables and connectors. Loose connections could overheat and catch fire.
- Protect all power supplies and connect to ground on the equipment using an appropriate connection. Failure to protect and/or ground the equipment could lead to fatal electrical shock.
- Take all possible measures to prevent foreign materials from falling into the product interior (liquids, flammable materials, metal objects, etc.).
- Turn the equipment off and disconnect all sources of power before undertaking any procedure whatsoever on the equipment.



Managing the CANbus – CANopen Protocol

The CANbus is a bus used for reliable data exchange between peers on a serial communication link. The following topics are covered in this document:

- 1- [Introduction to the CANbus Protocol](#), defining a node, a CAN message, bus arbitration, levels of implementation, and integration of the CANopen protocol in the CANbus.
- 2- [Integrating the CANbus Protocol into the RightWON](#), including the interrelations between the PLC IEC 61131-3 application, fieldbus manager, network management link and the hardware.
- 3- [CANbus Configuration Tutorial](#), describing the configuration settings and steps for developing a CANbus application.
- 4- [Advanced CANopen Protocol Management](#), including management and use of the dictionary (SDO) to configure input/output modules, adding additional PDOs to the CANopen configuration, and the description of an example project (CAN_PDO_Example).

2.1. Introduction to the CANbus protocol

CANbus is a serial communication protocol that supports real-time control of distributed systems in an efficient manner. The protocol allows subsystems connected to the bus to exchange data in turn. The bus consists of a single bidirectional channel that carries the [messages](#). This eliminates the need to use a point-to-point connection for each subsystem.

2.1.1. Node

A node is a peripheral device connected to the network that is able to communicate using the CANbus protocol. Nodes can be added to a functional network without requiring changes to existing nodes.

2.1.2. Message and message identifier (COB-ID)

Information is carried over the bus by means of a message (bit frame) in a defined format but with variable and limited length. Each message has a unique message identifier that defines its priority and contents (COB-ID: Communication OBject IDentifier). All nodes in a bus receive the message, and each is able to interpret whether it is the intended recipient.

Since several systems compete for bus access, the message identifier is unique across the entire network. To permit [bus arbitration](#), each frame begins with the COB-ID message identifier.

2.1.3. Bus arbitration

As soon as the bus is free, any node connected to the network can send a new message. When multiple nodes transmit a message simultaneously there is contention and data corruption. The systems sending the messages resolve the contention using an arbitration system based on the priority level contained in the message identifier. When contention occurs on the bus, lower priority stations stop transmitting. The messages are retransmitted by the stations when the bus becomes free again.

2.1.4. Implementation levels

The RightWON supports the CAN 2.0 A and CAN 2.0 B specifications of the CAN in Automation organization (www.can-cia.org). CAN implementations designed in accordance with the

CAN 2.0 A or CAN 2.0 B specification can communicate with each other as long as they do not use the extended format.

For further details, consult the site http://en.wikipedia.org/wiki/Controller_area_network.

2.1.5. CANopen protocol features

In the RightWON, the CANbus driver is used to manage CANopen subsystems. CANopen is an application protocol in a layer above the CANbus protocol. CANopen addresses the details and functions specific to the CAN implementation. It offers different ways to access the data in the [object dictionary](#): [Service Data Object](#) and [Process Data Object](#).

The CAN in Automation organization (www.can-cia.org) provides further information on the CANopen protocol. In addition, the following document is an excellent reference for understanding this protocol:

www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf

2.1.5.1. Object Dictionary (OD)

An object dictionary defines, for each input/output of a [node](#), information on the format of the data and how to access it. A dictionary entry is accessed by defining its unique index and sub-index [index, sub-index].

2.1.5.2. Service Data Object (SDO)

The SDO provides access to input/output parameters of a node included in the [object dictionary](#). The CANopen slave responds by sending the data parameters or by confirming a write to the OD. Requests are sent with a lower priority identifier than data messages received from the nodes.

2.1.5.3. Process Data Object (PDO)

The PDO is used to transfer real time data such as control and status information for an I/O module. The PDO accesses the data value in the [OD](#) by defining its [index, subindex]. The request is made through a message (RPDO) that is received by all nodes on the bus. This allows all nodes on the bus to read the requested data at the same time. The destination nodes for the message transmit the requested information (TPDO) over the bus to the network master.

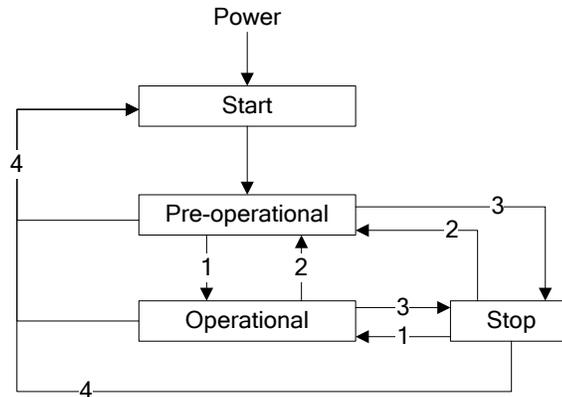
A PDO permits putting one or more fields of the OD into a single short message. A PDO contains from 0 to 8 bytes of data. For example, a PDO can transfer up to 64 binary data values or 4 analog data values with 2 bytes each.

2.1.5.4. CANopen NMT message

NMT messages are used to send commands that change the node status (e.g. start and stop node commands). The commands can be sent to a single node or all nodes on the bus. All CANopen slaves evaluate the incoming NMT commands, but only one NMT master is active and can send NMT commands.

A node can be in four possible states:

- Started: Nodes send a Boot-up message to the network master to indicate that the devices have been registered and are ready to operate. The node then enters pre-operational mode automatically.
- Pre-operational: In this state only SDO messages, error control messages and NMT commands are permitted.
- Operational: All types of messages are permitted (NMT, SDO, PDO, error control).
- Stopped: Only communication through NMT commands is permitted between the node and the network master. Node monitoring is also allowed.



The following NMT commands are available for changing the state:

- 1- **Start:** Initializes one or more nodes.
- 2- **Pre-operational:** The node(s) enter pre-operational state.
- 3- **Stop:** Stops communication between the node(s) and the network master.
- 4- **Reset:** Resets node parameters in the OD to their default values.

See the section [“Creating a CANopen NMT message”](#) for further details.

2.1.5.5. Error control

Error control allows the network master to assess whether the CANopen slaves are operating properly in the network. There are two types of error checking: heartbeat and node monitoring (or “guarding”).

- **Heartbeat:** The CANopen slave periodically sends a message to the network master indicating its presence and heartbeat status. The time period is configured in the OD and the master must receive at least one heartbeat message during this period. The master then evaluates whether the node is functioning properly and is in the correct network state.
- **Node guarding:** Node guarding permits verifying whether the node is still operating in the correct network state. The master requests an error control message from the CANopen slave via a remote CAN frame (“Remote Transmit Request”). The slave responds with a CAN data frame that contains its current NMT state.

2.1.6. CANopen system support

The RightWON supports most CANopen controllers offered by input/output system manufacturers, such as:

- Phoenix Contact CANopen Bus Terminal, IL CAN BK-TC-PAC (32 TPDOs, 32 RPDOs)
- Wago CANopen Fieldbus Coupler, 750-307 (5 RPDOs, 5 TPDOs) and 750-337 (32 TPDOs and 32 RPDOs)
- Wieland Bus coupler unit RICOS TP BC-CANOPEN, 83.039.0120.0 (5 RPDOs, 5 TPDOs)
- Many other products.

By default (without configuration), CANopen controllers that comply with the DS401 specification support 4 PDOs for input to the RightWON (TPDO1 to TPDO4) and 4 PDOs for output from the RightWON (RPDO1 to RPDO4), each with a maximum of 8 bytes. These PDOs are used to transport data between the RightWON and the inputs/outputs. TPDO1 is

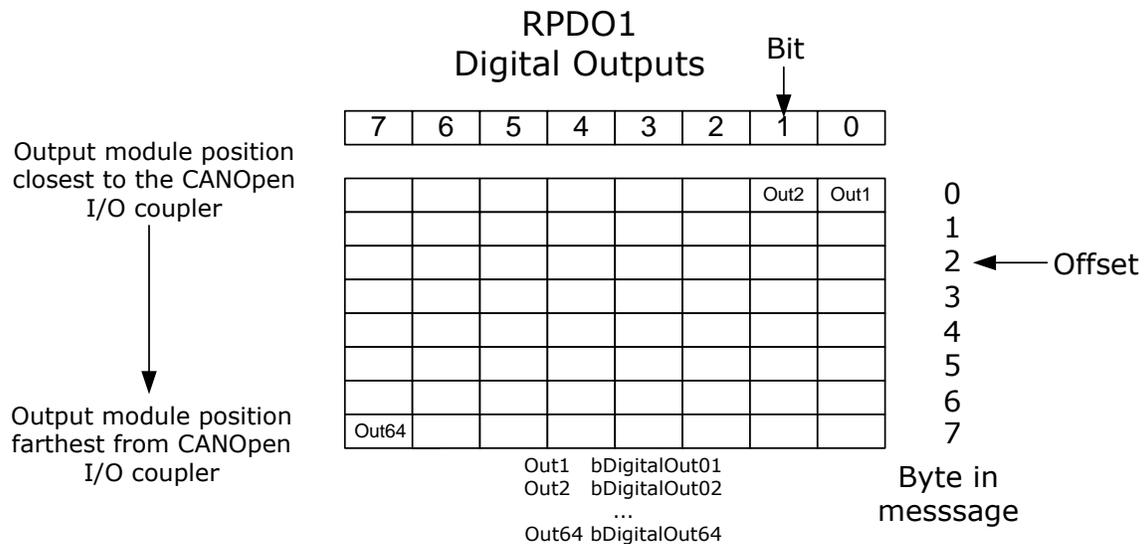
automatically assigned to the first 64 digital inputs found to the right of the controller, while RPDO1 is automatically assigned to the first 64 digital outputs. TPDO2 to TPDO4 are assigned to the first analog inputs (12 inputs, or 4 per block) while RPDO2 to RPDO4 are assigned to the first 12 analog outputs. Note that the concept of transmission (TPDO) and reception (RPDO) is viewed from the CANbus coupler connected to RightWON.

PDO	Identifier (hexadecimal)	Usage
TPDO1	181	Max. 64 digital inputs
TPDO2	281	Max. 4 16-bit analog inputs (1-4)
TPDO3	381	Max. 4 16-bit analog inputs (5-8)
TPDO4	481	Max. 4 16-bit analog inputs (9-12)
RPDO1	201	Max. 64 digital outputs
RPDO2	301	Max. 4 16-bit analog outputs (1-4)
RPDO3	401	Max. 4 16-bit analog outputs (5-8)
RPDO4	501	Max. 4 16-bit analog outputs (9-12)

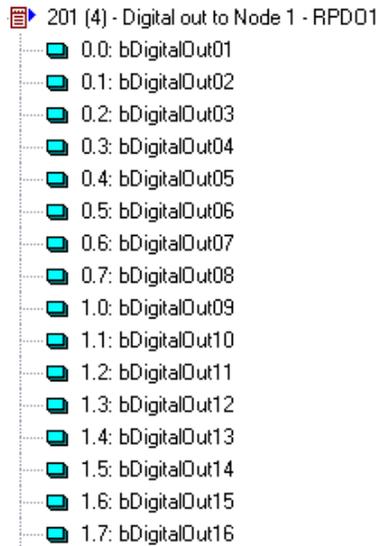
For further details refer to the section ["PDO addressing and assigning additional PDOs to a node"](#).

2.1.7. Function of PDOs associated with digital inputs/outputs

The following figure illustrates the mapping of RPDO1 associated with the digital outputs of a CANopen node. Since the RPDO contains 8 bytes, it can be associated with up to 64 Boolean variables. During automatic configuration of a node, the outputs are assigned in the order they are found to the right of the CANopen coupler. Assignment starts from bit 0 of the byte (0) to bit 7. When more than 8 digital outputs are found, they are assigned to the following bytes until the block is full.



The following figure illustrates the mapping of Boolean variables *bDigitalOut%%* to the first 16 outputs of RPDO1 with COB-ID equal to 201. The corresponding bit of the RPDO is designated by the index *offset.bit*.

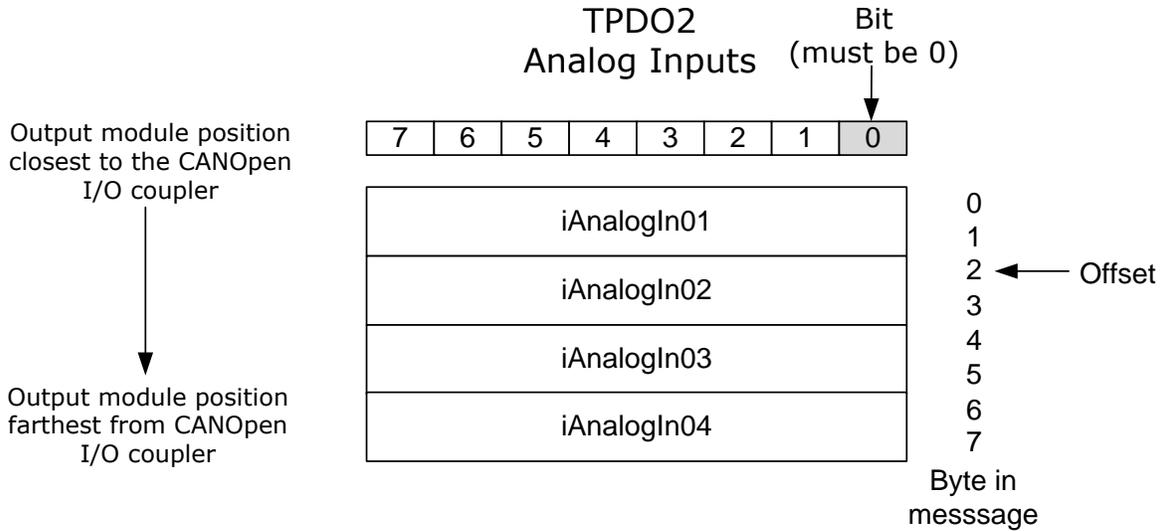


For example, the variable *bDigitalOut11* is associated with offset 1, bit 2 (1.2) of RPDO1.

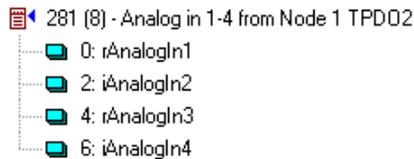
Note that mapping of digital inputs is carried out in the same way as for digital outputs, except that they are assigned to TPDO1 instead of RPDO1.

2.1.8. Function of PDOs associated with analog inputs/outputs

The following figure illustrates the mapping of TPDO2 associated with the analog inputs of a CANopen node. Since the TPDO contains 8 bytes, it can be associated with a maximum of 4 integer variables. During automatic configuration of a node, the 12- or 16-bit analog inputs are assigned in the order they are found to the right of the CANopen coupler. Assignment starts from bytes 0 and 1 (offset 0), then continues with offsets 2, 4 and 6 until the block is full. Additional analog inputs are automatically assigned as required to TPDO3 and TPDO4.



The following figure illustrates 4 variables (real numbers) assigned to the first 4 analog inputs of the CANbus coupler at node 1 (TPDO2). Although the data of the TPDO are divided into four 16-bit signed integers, variables of any type may be linked to the PDO since the CANbus driver converts the data.



In addition, the driver supports data scaling. In the following example, an analog input from a Phoenix Contact IB IL AI 2/SF module converts 4-20mA to a rough number between 0 and 30000 (Signal Min and Max). The variable associated with the input will range from 0 and 100.00 (Range).

Range	<input type="text" value="0"/>	Range	<input type="text" value="100.00"/>
Signal Min:	<input type="text" value="0"/>	Signal Max:	<input type="text" value="30000"/>

2.1.9. Updating digital and analog outputs

The RightWON supports 3 methods for updating digital and analog outputs:

- 1- [Change of value](#)
- 2- [Periodic dispatch](#)
- 3- [Dispatch on demand](#)

2.1.9.1. Update through change of value

In this mode, the entire RPDO is sent by the RightWON to the CANopen coupler when any variable assigned to the RPDO is changed. To prevent congestion of the CANbus due to changes that occur close together, the minimum period defines a wait time in milliseconds before sending the block when a value changes. The maximum period forces a periodic refresh of the outputs even if no value has changed. This update method is commonly used for digital outputs.

The screenshot shows the 'CAN Message' dialog box with the following settings:

- Description: Digital out to Node 1 - RPDO1
- Message ID: 201
- Length: 4 (bytes)
- RTR:
- Mode:
 - Received
 - Sent periodically
 - Sent on request (one shot)
 - Sent on change of data
- Min period: 100 (ms)
- Max period: 60000 (ms)
- Initial data (hexadecimal): [Empty text box]

2.1.9.2. Periodic update

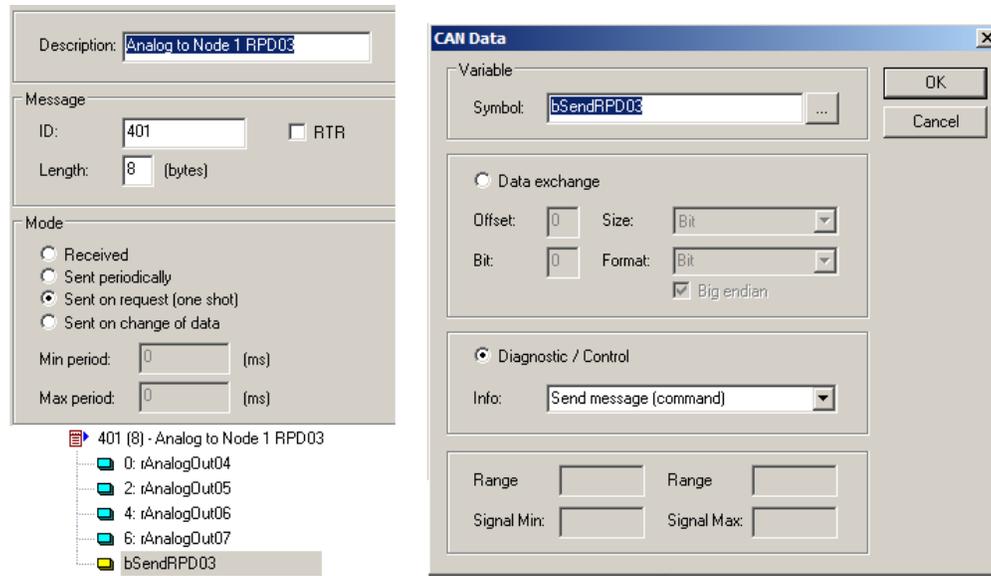
In this mode, the entire RPDO is sent by the RightWON to the CANopen coupler according to the specified period. This is the mode most commonly used to update analog outputs.

The screenshot shows the 'CAN Message' dialog box with the following settings:

- Description: Analog to Node 1 RPDO2
- Message ID: 301
- Length: 8 (bytes)
- RTR:
- Mode:
 - Received
 - Sent periodically
 - Sent on request (one shot)
 - Sent on change of data
- Min period: 1000 (ms)
- Max period: 0 (ms)
- Initial data (hexadecimal): [Empty text box]

2.1.9.3. Update on demand

In this mode, the entire RPDO is sent by the RightWON when the associated request is activated. The driver resets the request to 0 after processing. This request is associated with the data block by configuring it as a diagnostic/control exchange rather than a data exchange.



2.1.10. Updating variables from the digital and analog inputs

The RightWON supports 3 methods for updating variables from the digital and analog inputs:

- 1- [By event](#)
- 2- [By RTR request](#)
- 3- [By SYNC request](#)

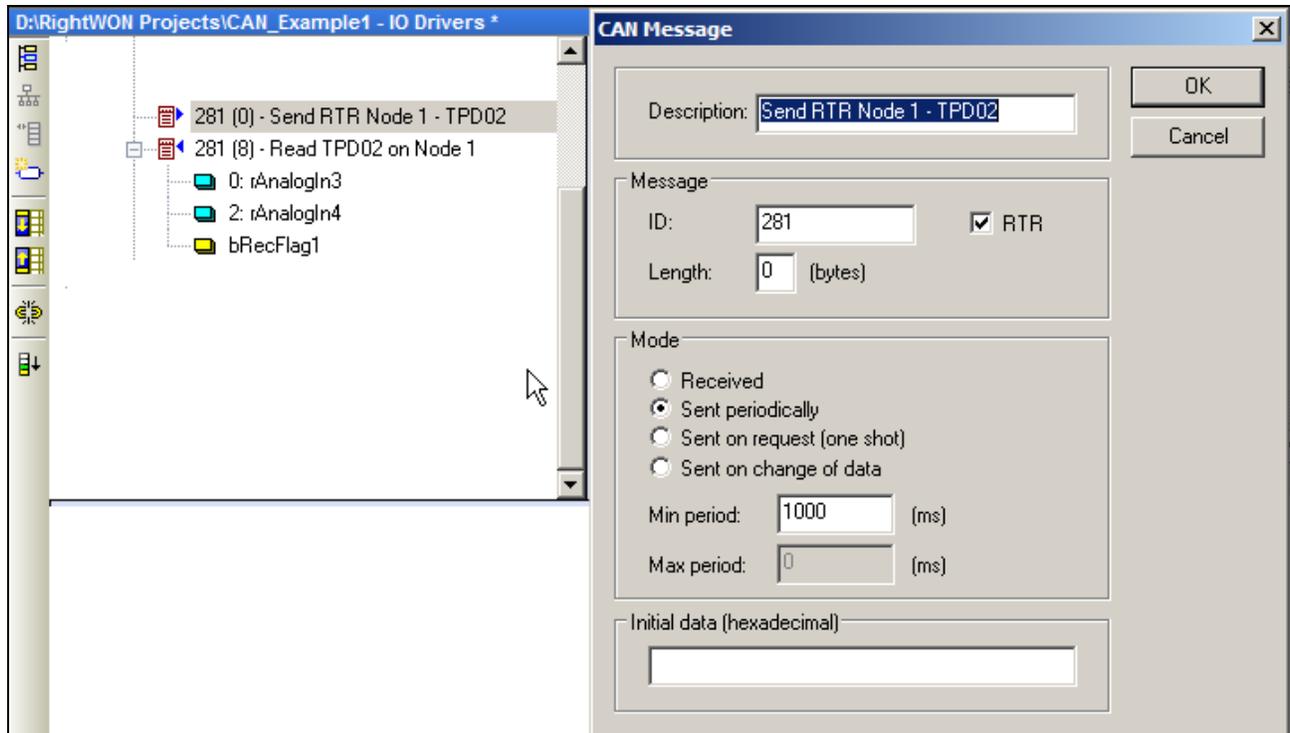
2.1.10.1. Update by event

Update by event is the default method used for digital inputs. Whenever the CANopen coupler detects a change in one of the inputs associated with TPDO1, the block is sent to the CAN controller. Note that a large number of events can overload the communication bus.

The event-based operating mode is configurable bit by bit for each digital input in the system. In addition, this operating mode can be used for analog inputs by defining deadbands and/or limitations. Configuration of event-based operating modes requires advanced programming using SDO requests, or parameter settings carried out using a CANopen configurator (see ["Configuring the CANopen controllers"](#)).

2.1.10.2. Update by RTR request

Update by RTR (Remote Transmission Request) is the method most commonly used to read analog inputs. It involves a query generated by the RightWON requesting the CANopen coupler to send the corresponding TPDO. It is somewhat like a polling request. As in the case of updating outputs, this request can be configured as a [periodic dispatch](#) or a [dispatch on demand](#).



2.1.10.3. Update by SYNC request

Update by SYNC request permits synchronizing data dispatch with the message initiated by the RightWON. A single SYNC message can trigger the automatic sequential delivery of all TPDO objects from the coupler or the simultaneous sampling of inputs. Configuration of this operating mode requires advanced programming using SDO requests, or parameter settings carried out using a CANopen configurator (see "[Configuring the CANopen controllers](#)").

2.1.11. Configuring the CANopen controllers

Configuring extended I/Os beyond 64 digital inputs (TPDO1), 64 digital outputs (RPDO1), 12 analog inputs (TPDO2 to TPDO4) or 12 analog outputs (RPDO2 to RPDO4) requires configuration of the CANopen controller. This configuration permits assigning input/output modules to the PDOs. Several CANopen configuration software programs are available from input/output hardware suppliers. However, some specialized editors offer CANopen configuration and verification software. To this end, the *CANopen ConfigurationStudio* (http://www.ixxat.com/canopen_configurationstudio_en.html) and *CANopen Magic Pro* (<http://www.canopenmagic.com/peak/professional.htm>) are excellent work tools.

2.1.12. PDO addressing and assigning additional PDOs to a node

According to the CANopen protocol and the DS401 specification, the COB-ID message identifier contains 4 function code bits that designate the object type (NMT, TPDO1, RPDO3, TSDO, etc.) and 7 address bits that designate the node. Address 0 designates all nodes.

The PDO identifier consists of the base address added to the node address. PDO base addresses for node 1 are illustrated in the following table. From this table, TPDO1 on node 3 would be equal to 183, and RPDO4 on node 5 would be equal to 505.

PDO	Identifier (hexadecimal)	Usage
TPDO1	181	Max. 64 digital inputs
TPDO2	281	Max. 4 16-bit analog inputs (1-4)
TPDO3	381	Max. 4 16-bit analog inputs (5-8)
TPDO4	481	Max. 4 16-bit analog inputs (9-12)
RPDO1	201	Max. 64 digital outputs
RPDO2	301	Max. 4 16-bit analog outputs (1-4)
RPDO3	401	Max. 4 16-bit analog outputs (5-8)
RPDO4	501	Max. 4 16-bit analog outputs (9-12)

The basic protocol defines only PDOs 1 to 4 for designating data objects. Therefore, any additional data must be assigned to unused identifiers in the network using the CANopen configurator (e.g. the PDOs of another uninstalled node). Following is an example assigning additional PDOs for node 1 using the uninstalled PDOs of node 2:

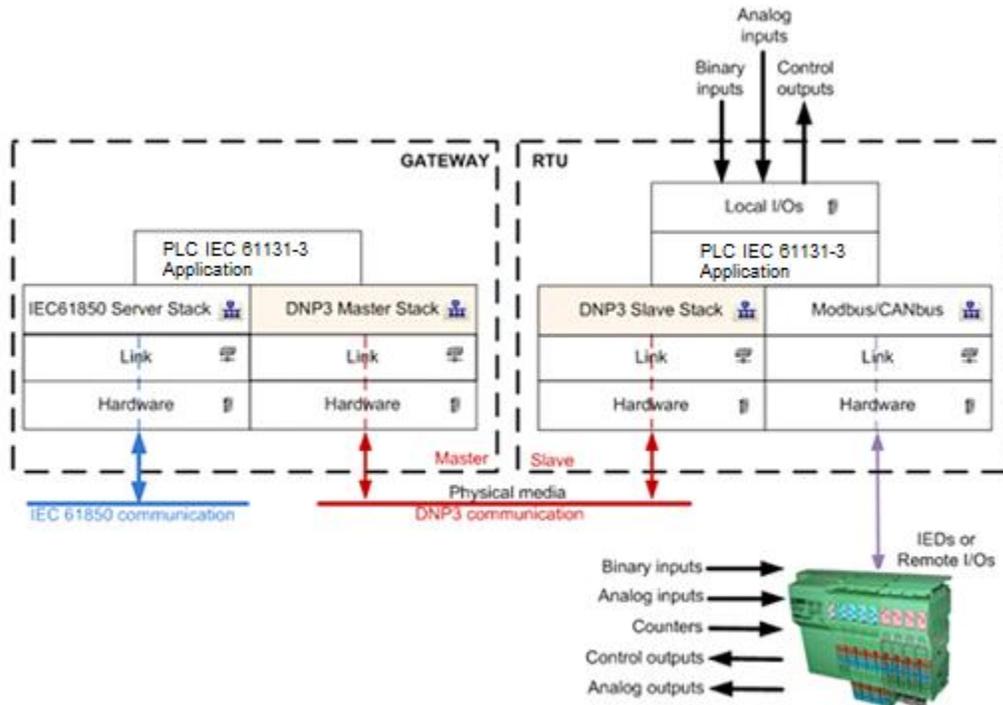
PDO	Identifier (hexadecimal)	Usage
TPDO5	182	Max. 64 digital inputs
TPDO6	282	Max. 4 16-bit analog inputs (1-4)
TPDO7	382	Max. 4 16-bit analog inputs (5-8)
TPDO8	482	Max. 4 16-bit analog inputs (9-12)
RPDO5	202	Max. 64 digital outputs
RPDO6	302	Max. 4 16-bit analog outputs (1-4)
RPDO7	402	Max. 4 16-bit analog outputs (5-8)
RPDO8	502	Max. 4 16-bit analog outputs (9-12)

The section "[Adding more PDOs to a CANopen configuration](#)" describes how to add other PDOs to a node (CANopen coupler).



Integrating the CANbus Protocol into the RightWON

Each RightWON CPU (RWU 010000) has one CANbus port. The CANbus operating mode in a typical application is illustrated in the following figure. All data carried by the CANbus protocol translate into variables in the PLC IEC 61131-3 application. Thus the protocols are managed from the fieldbus manager under the supervision of PLC automation programs. The fieldbus manager exchanges the variables between the CANbus driver and the PLC application.



RightWON units support the CANopen protocol through the [CANbus driver configured with the Drivers I/O configurator](#). The logical link between the physical port in the RightWON hardware configuration and the CANbus driver is managed by the [CANbus communication link](#). This link is configured by the Link manager in the Network Configurator.

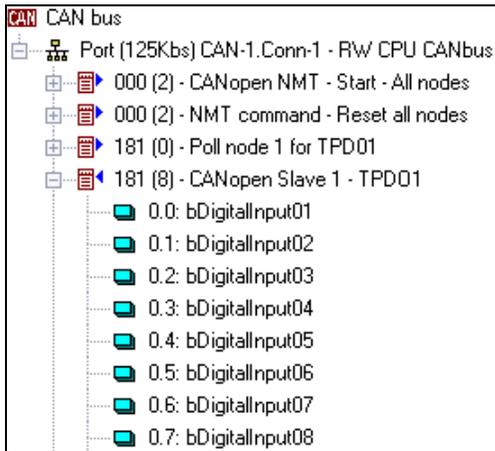
3.1. Configuring CANopen I/Os

RightWON binary and analog variables contain data to be exchanged with the modules of a CANbus node. The variables are assigned to RPDOs and TPDOs through configuration of the CANbus driver.

In hierarchical order, the integrator must define:

- 1- The CANbus communication link with the network manager.
- 2- The CANbus configuration that integrates the CANbus driver with the application loaded in the RightWON.
- 3- The CANbus port used to establish communication with devices connected to the RightWON. This is where network throughput is defined.

- 4- Requests for the exchange of data, commands and control with the nodes.
- 5- If applicable, the variables associated with data and control exchanges at a lower level.



For CANbus-CANopen implementation, refer to "[CANbus Configuration Tutorial](#)".



CANbus - CANopen Configuration Tutorial

To configure the CANbus, carry out the following steps:

- 1- [Create a new RightWON project](#)
- 2- [Add and configure a CANbus link](#) in the network configurator
- 3- [Add the CANbus protocol driver](#) to the RightWON configuration
- 4- [Insert and configure a port](#)
- 5- [Create the TPDOs and RPDOs for data exchange](#) between the RightWON and the CANopen coupler
- 6- [Configure the PDO exchange mode](#) between the RightWON and the CANopen coupler
- 7- [Create periodic RTR requests](#) to return the analog and digital inputs
- 8- [Insert the variables into the PDOs](#)
- 9- [Insert and configure a CANopen NMT message](#)
- 10- [Start the nodes](#) with the pStartup program
- 11- [Launch the project](#)

The CAN_Tutorial project contains a program based on a CANopen coupler with 16 digital inputs/outputs and 4 analog inputs/outputs.

4.1. Creating a new RightWON project

To create a new RightWON project, refer to the manual "RWM000080-MA-en, RightWON - Application Guide". As a minimum, the project must include the RightWON configuration in the fieldbus configurator.

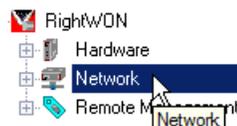
4.2. Adding and configuring a CANbus link

To use the CANopen I/O modules you must add a CANbus link to associate the hardware with the CANbus driver.

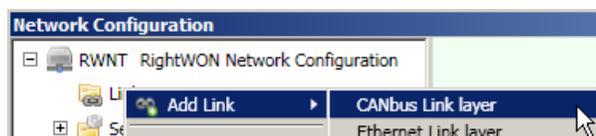


Consult the manual "RWM000010-MA-en, RightWON Configuration Suite - Manual" for more details on the RightWON network configurator.

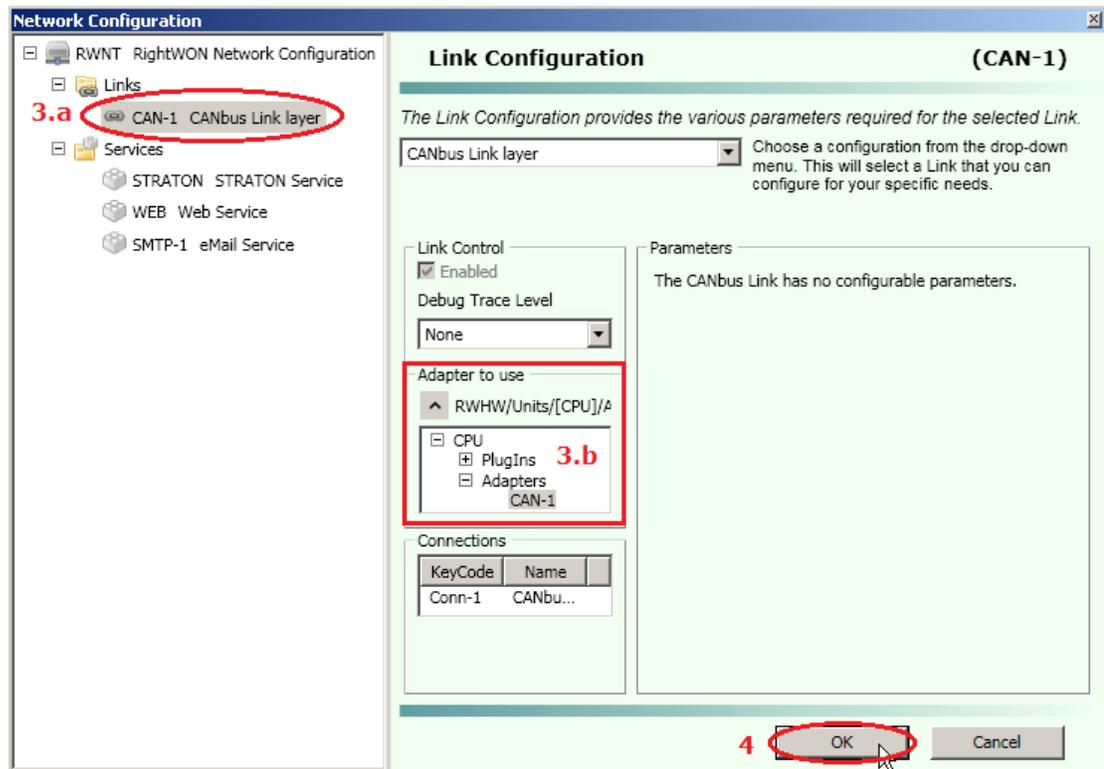
- 1- In the **RightWON** configuration accessed from the **IO Drivers** window, double-click on **Network**.



- 2- In the network configurator, right-click on **Links**. Select **Add Link**, then click on **CANbus Link Layer**.



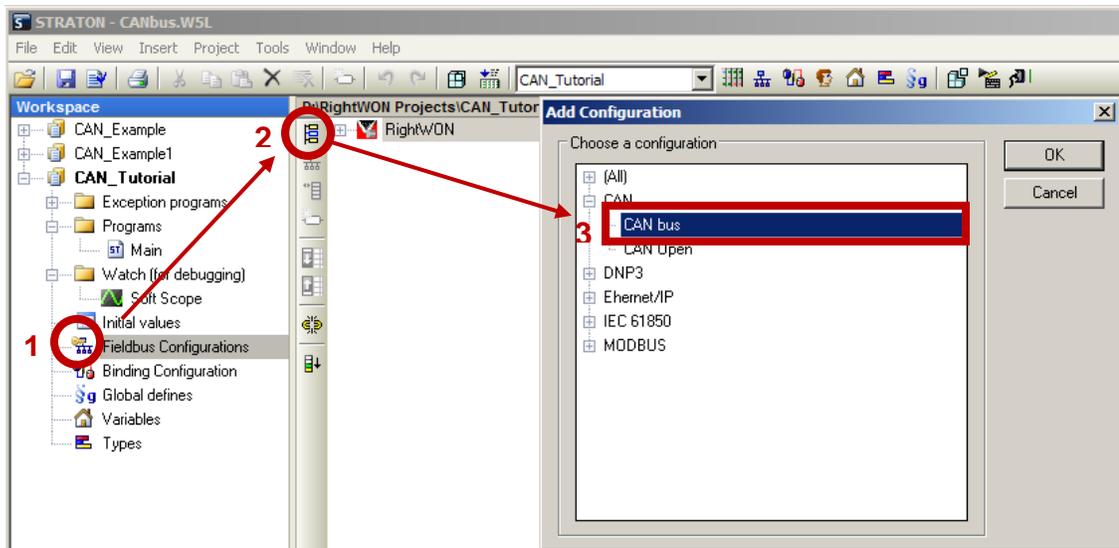
- 3- If the project has only one CPU without an MCU, the association between CANbus link and hardware is carried out automatically. If the project has one CPU with one or two MCUs, the CAN link must be paired with the adapter on the CPU or the MCU that is used for communicating with the CANbus IEDs. To do this, carry out the following steps:
 - a. In the navigation area, select **CAN-1** under the **Links** section.
 - b. Next, in the work area, select the **CAN-1** adapter in the **Adapter to use** section.
- 4- Click **OK**.



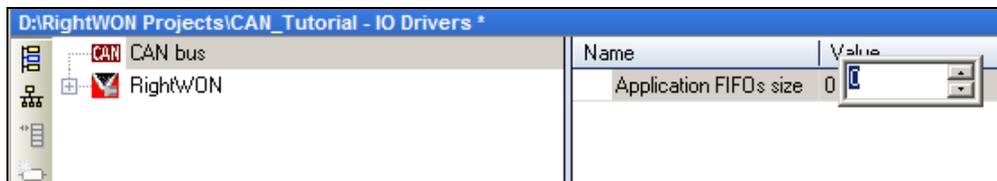
4.3. Adding the CANbus protocol driver

To add the CANbus protocol to the RightWON configuration, carry out the following steps:

- 1- In the work area, double-click on **Fieldbus Configurations**.
- 2- In the **IO Drivers** window, click on the **Insert Configuration** icon.
- 3- Select **CANbus** from the list, then click **OK**.



- 4- In the **IO Drivers** window, double-click on **CANbus** to edit the driver parameters (see “CANbus driver setup”). This configuration is not required in the tutorial.

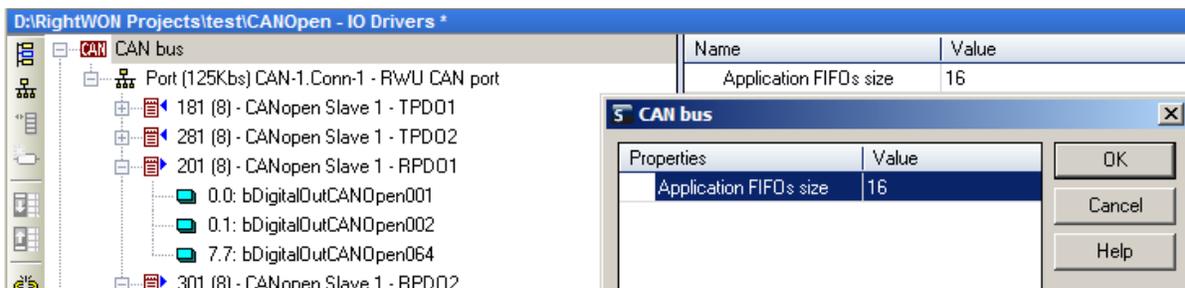


- 5- After you add and configure the CANbus driver, you must **add and configure the physical CANbus port**.

4.3.1. CANbus driver setup

The “**Application FIFOs size**” parameter must be adjusted to a minimum of double the number of simultaneous requests, programmable with the [CanRcvMsg](#) and [CanSndMsg](#) functions, when used. This FIFO (First In First Out) stack manages the storage of requests generated by the application and those sent on the bus. For example, during a single controller cycle, several requests could be generated simultaneously and stored in the FIFO until the bus is available and ready to exchange a new message. The more requests there are and the faster the exchanges are made, the larger the FIFO should be defined.

When the CANbus driver is entirely managed by the fieldbus configurator, the FIFO is not required and can be left at 0.

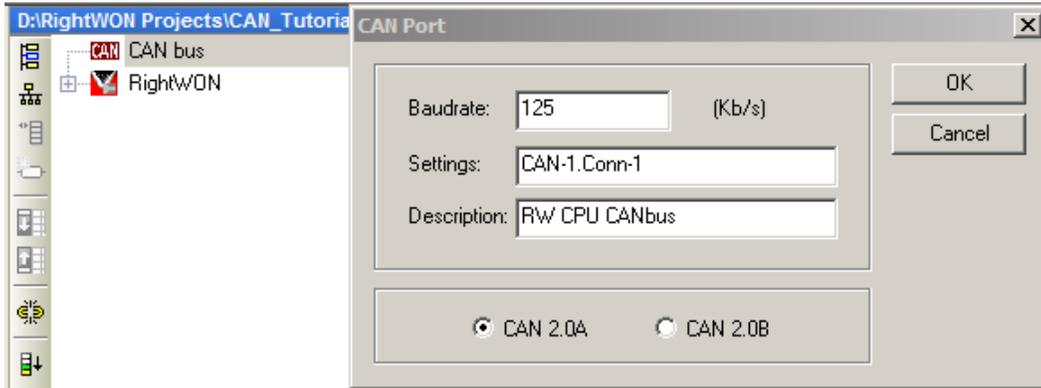




On startup, a common cause of data loss is FIFO overflow or an excessive flow of messages with respect to network capacity. Increasing the FIFO size will solve the problem in many cases.

4.4. Inserting and configuring a port

- 1- In the **IO Drivers** area, select **CAN bus** and execute the command **Insert a master/a port...**



- 2- The CAN Port configuration window appears. Enter the following data:

Baudrate: Communication throughput rate on the CANbus. All nodes in the network must communicate at the same throughput rate. For most CANbus controllers, this rate is set with DIP switches. The rate depends on the length of the network and type of cable used, as indicated in the table below. A shielded twisted pair cable with an impedance between 108 and 132 Ohms is recommended.

Throughput	Length
1000 Kb/s	30m
800 Kb/s	50m
500 Kb/s	100m
250 Kb/s	250m
125 Kb/s	500m

Settings: Enter the address of the CAN communication link defined in the network configurator, usually *CAN-1.Conn-1* which refers to the link for the CANbus controller integrated into the CPU RWU010000 (in case there is no MCU). See ["Adding and configuring a CANbus link"](#) for details.

Description: Enter a name for the connection that is meaningful to your application.

CAN 2.0A/CAN 2.0B: Enter the protocol compatibility level required for IEDs connected to the bus.

- Standard CAN or CAN 2.0 A, with an object identifier encoded on 11 bits, which theoretically permits accepting up to 2048 types of messages (limited to 2031 for historical reasons).
- Extended CAN or CAN 2.0 B, with an object identifier encoded on 29 bits, which theoretically permits accepting up to 536 870 912 types of messages. At the request of the [SAE](#), which is the source of the J1939 standard.

Verify compatibility with the I/O manufacturer. If in doubt, choose the CAN 2.0A protocol.

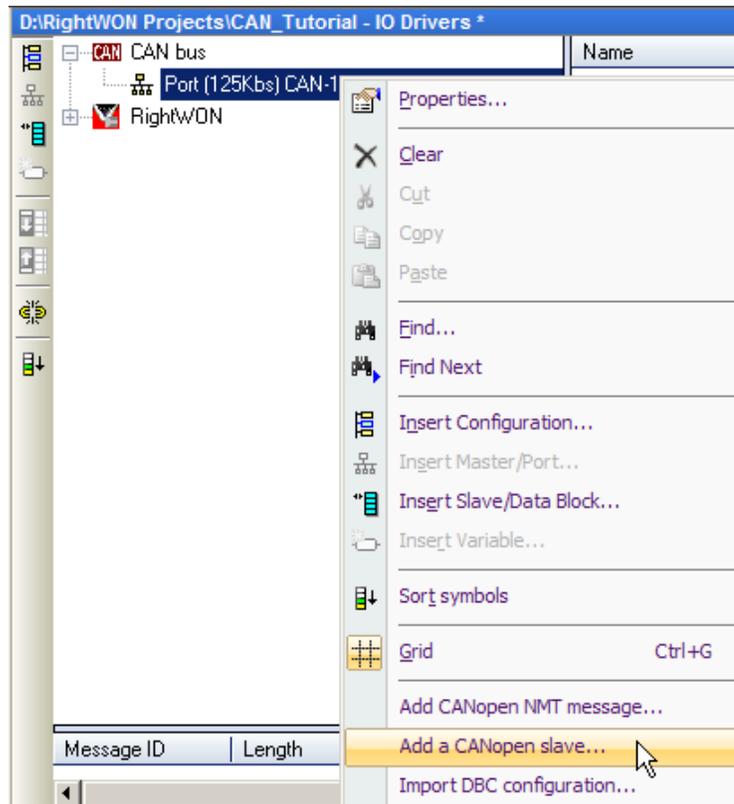
- 3- To create the port, click **OK**. Once the port has been added, you must [create the PDOs required for data exchange with the CANopen coupler](#).

4.5. Creating the TPDOs and RPDOs for data exchange

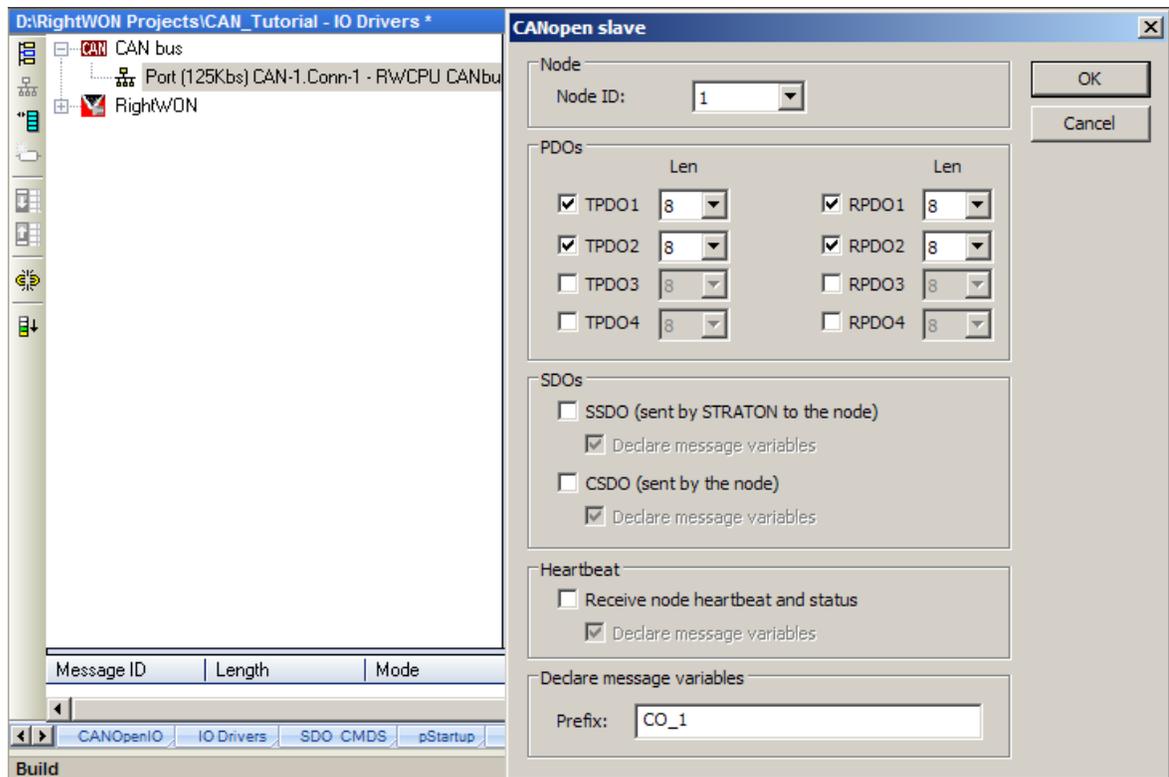
The **Add a CANopen slave...** menu permits defining the PDOs, SDOs and Heartbeat messages that are exchanged between the RightWON and the CANopen coupler.

The following steps create the TPDOs (Coupler → RightWON) and RPDOs (RightWON → Coupler) that are required to exchange the input/output data.

- 1- In the navigation area, click on **Fieldbus Configurations** to access the **IO Drivers** area. Select the **CANbus Port** and right click to select **Add a CANopen slave...**



- 2- The CANopen slave configuration window appears. Configure the parameters according to the following information.



In the **NODE** section:

Node ID: Select the address of the node based on your CANopen coupler (1 to 127). Address 0 designates all nodes; do not use.

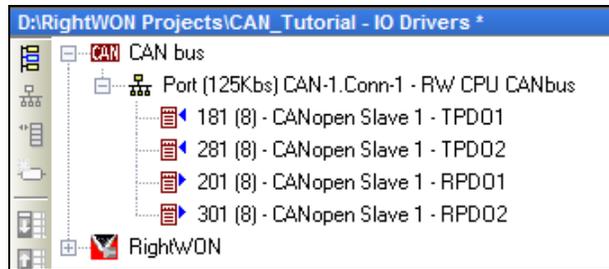
Under **PDOs** ("Process Data Objects"), check the PDOs that are required for data exchange:

- TPDO1:** PDO associated with a maximum of 64 digital inputs on the CANopen coupler.
- TPDO2:** PDO associated with analog inputs 1 to 4 on the CANopen coupler.
- TPDO3:** PDO associated with analog inputs 5 to 8 on the CANopen coupler.
- TPDO4:** PDO associated with analog inputs 9 to 12 on the CANopen coupler.
- RPDO1:** PDO associated with a maximum of 64 digital outputs on the CANopen coupler.
- RPDO2:** PDO associated with analog outputs 1 to 4 on the CANopen coupler.
- RPDO3:** PDO associated with analog outputs 5 to 8 on the CANopen coupler.
- RPDO4:** PDO associated with analog outputs 9 to 12 on the CANopen coupler.

For each PDO:

Len: The message length, from 1 to 8 bytes. A length of 8 bytes can carry 64 digital inputs/outputs or 4 16-bit analog inputs/outputs.

- 3- To create the PDOs, click **OK**. You should create the number of PDOs needed to acquire all analog and digital inputs and update all digital and analog outputs. After these steps, the content of the configurator looks like the following figure.



- 4- After creating the PDOs, you must [configure the data exchange modes between the CANopen coupler and the RightWON](#).

4.6. Configuring the PDO exchange modes

Once the PDOs have been created you must configure how they are to be exchanged with the CANopen coupler:

- [Digital outputs \(RPD01\)](#)
- [Analog outputs \(RPD02 to RPD04\)](#)
- [Digital inputs \(TPD01\)](#)
- [Analog inputs \(TPD02 to TPD04\)](#)

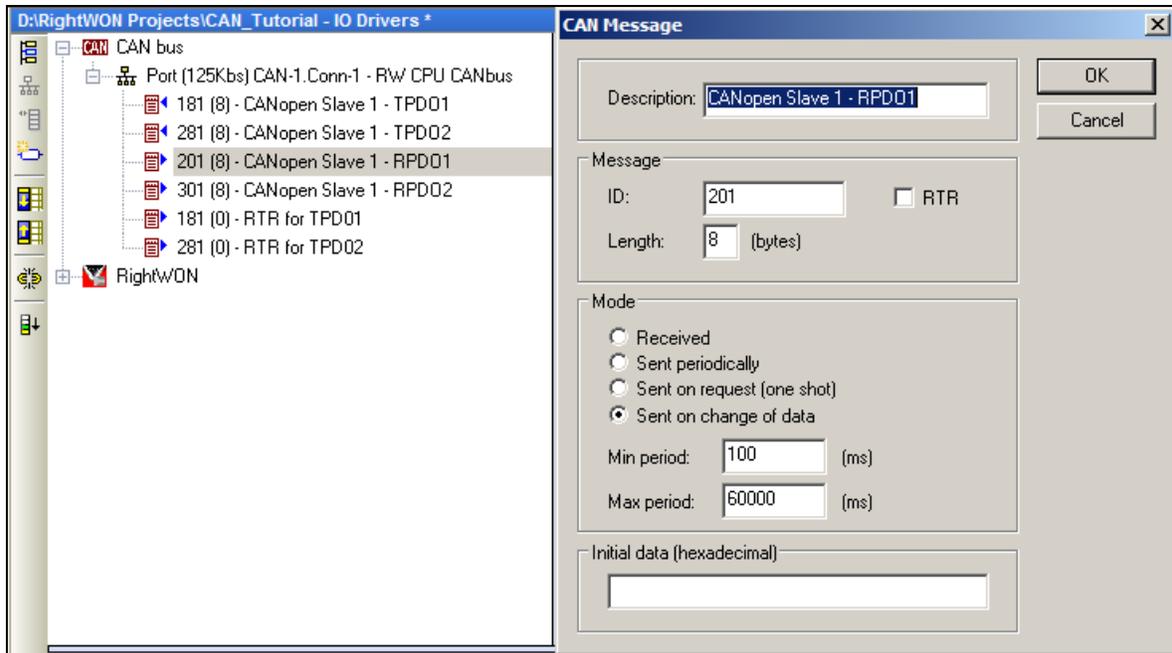
For further details, refer to the sections "[Updating digital and analog outputs](#)" and "[Updating variables from the digital and analog inputs](#)".

To configure the PDOs, double click the desired PDO from the **IO Drivers** window. Configure the window that appears as follows in the next sections.

To acquire the analog and digital inputs you must also [create periodic RTR requests](#) that return information to the RightWON periodically.

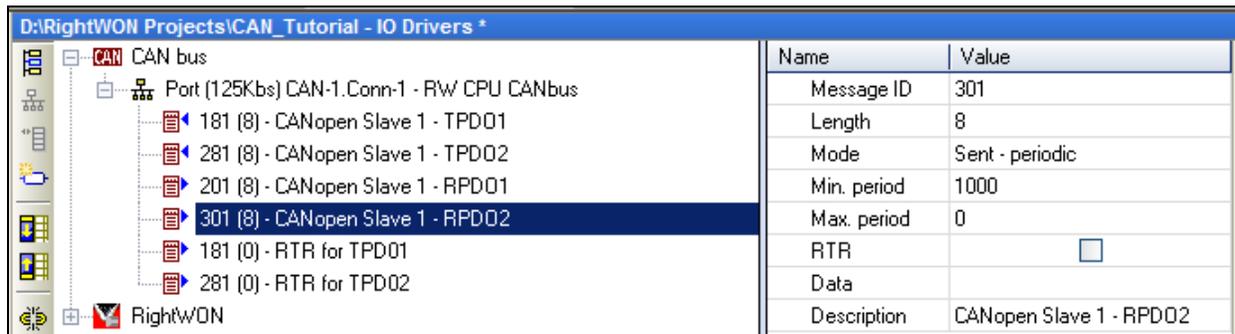
4.6.1. Digital outputs

Digital outputs (RPD01) are usually sent to the coupler on changes of state. Set the minimum period to 100 ms (to avoid saturating the CANbus from successive changes of multiple outputs) and the maximum period to 1 minute (in case problems arise).



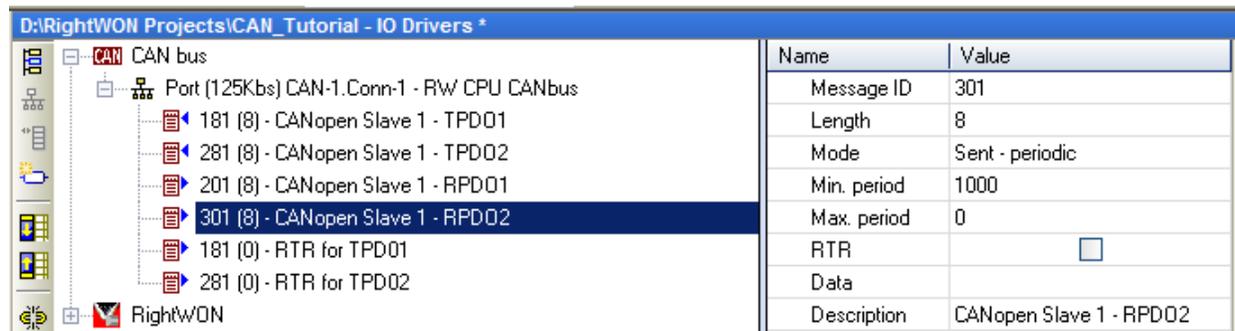
4.6.2. Analog outputs

Analog outputs (RPDO2 to RPDO4) are usually updated periodically (e.g. once per second).



4.6.3. Digital inputs

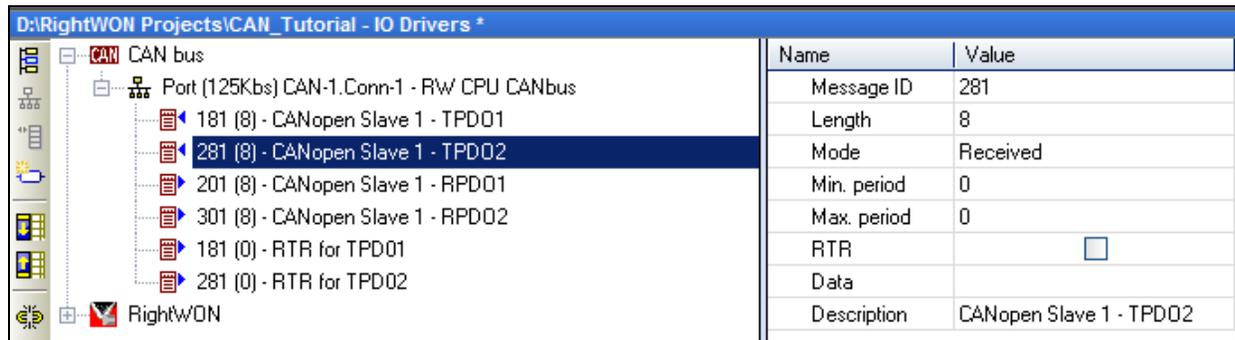
Digital inputs (TPDO1) are updated by event, and are therefore automatically returned to the RightWON by the coupler. The TPDO1 block is used for data reception, and requires no configuration.



It is nevertheless recommended to create a [periodic RTR request](#) that returns the block, for example, once per minute in case any events are lost.

4.6.4. Analog inputs

Analog inputs (TPDO2 to TPDO4) must be returned with an **RTR request**, since by default CANopen controllers do not generate events.

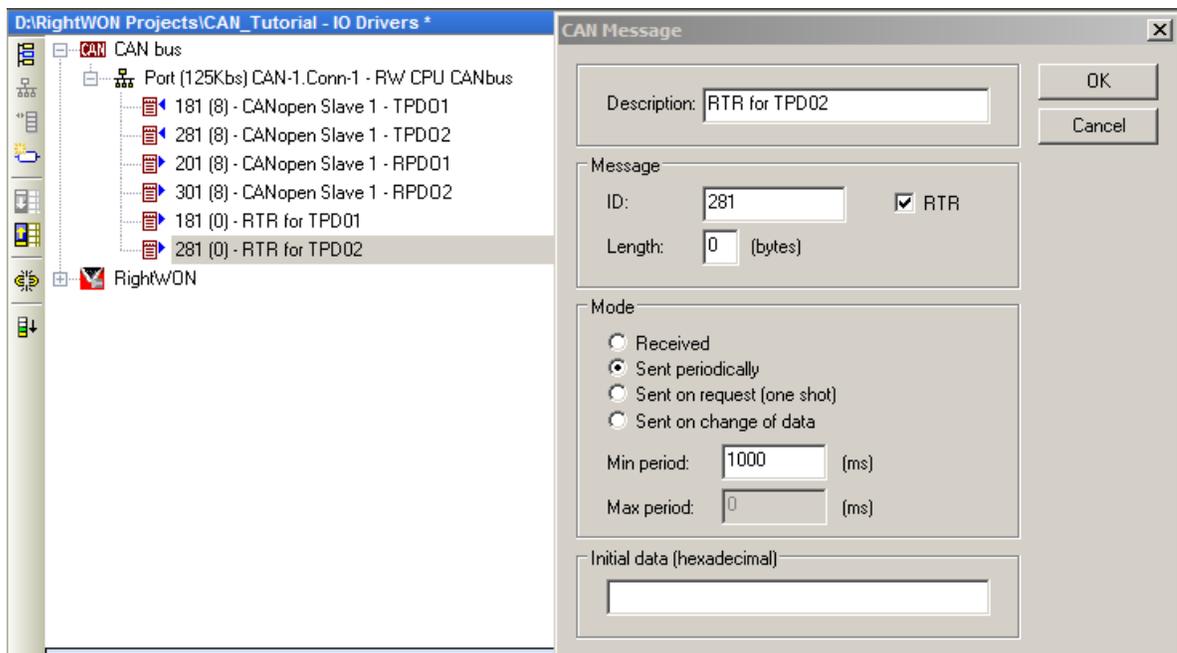


Name	Value
Message ID	281
Length	8
Mode	Received
Min. period	0
Max. period	0
RTR	<input type="checkbox"/>
Data	
Description	CANopen Slave 1 - TPDO2

4.7. Creating periodic RTR requests

RTR requests are used to force the CANopen coupler to send TPDOs to the RightWON. The requests can be periodic or **on demand**. You must create one request for each TPDO that is to be returned to the RightWON. Periodic requests are created as follows:

- 1- Select the CANbus port from the **IO Drivers** window, then click on **Insert Slave/Data Block** .
- 2- Fill out the **Description** field of the request according to your preferences.
- 3- Enter the **ID** of the TPDO (181 for TPDO1 on node 1, 281 for TPDO2 on node 1, 182 for TPDO1 on node 2, etc.) and a **Length** of 0 since there is no data to carry. The request triggers sending the TPDO from the coupler as soon as it is received.
- 4- Check **RTR** to indicate that this is a send request.
- 5- Select **Sent periodically** and set the **Min. period** (60,000 ms recommended for TPDO1 requests, and 1000 ms for TPDO2 to TPDO4 requests).



4.8. Inserting a variable into a PDO

Inputs/outputs from the CANopen coupler are exchanged with the variables via the PDOs. TPDOs are used for the inputs and RPDOs for the outputs.

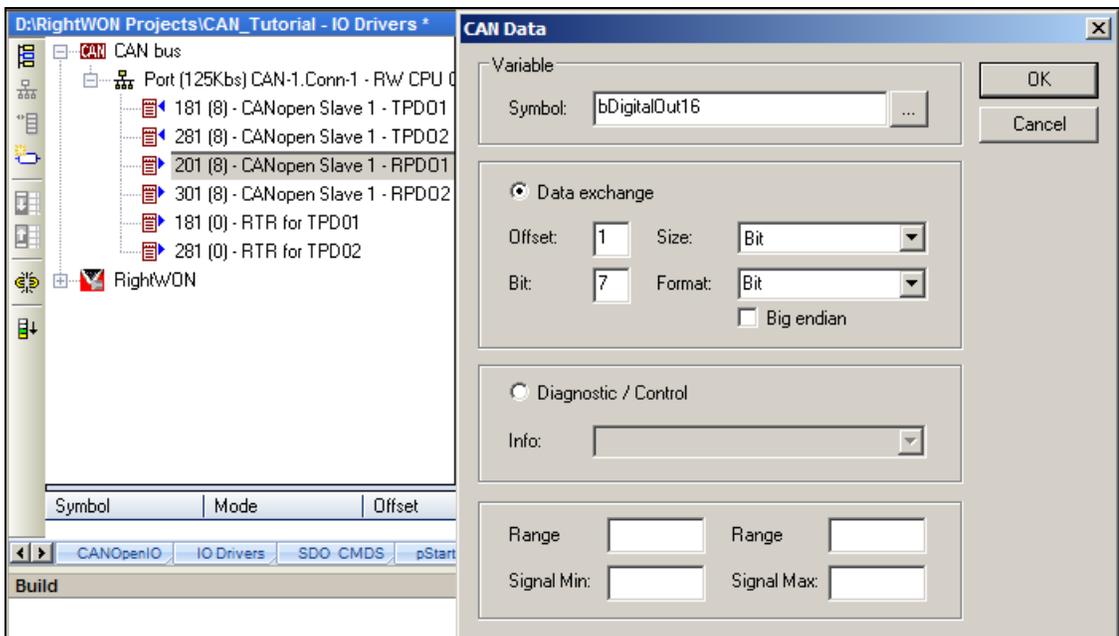
4.8.1. Associating variables with digital inputs/outputs

Digital inputs are associated with TPDO1, while digital outputs are associated with RPDO1. To insert a digital variable in a PDO, carry out the following steps:

- 1- In the **IO Drivers** area, right click on **CANopen Slave** and select **Insert Variable...**



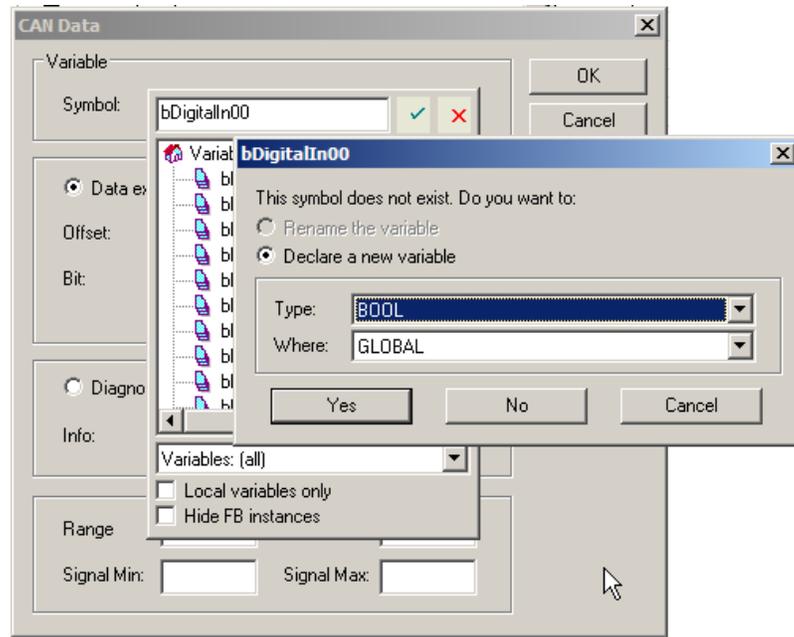
- 2- The CAN Data configuration window appears. This window permits assigning up to 64 variables to the PDO using an *offset.bit* reference system, where the *offset* is byte 0 to 7 of the PDO and the *bit* is the bit in that byte (0 = least significant). For further details, refer to "[Function of PDOs associated with digital inputs/outputs](#)".



- 3- Define the following fields:

Symbol: The name of the variable associated with the input/output. The  button permits inserting a variable:

- Select a variable from the list and click on ✓.
- Or create a new variable by entering its name and clicking on ✓.
Note: It is preferable to have a variable naming convention. For example, prefix DINT variables with *di*, INT variables with *i*, and BOOL variables with *b*.
 A window for configuring the variable appears. Choose **BOOL** as the **Type** and assign it as a **GLOBAL** variable.



Data exchange: Select for data exchange.

Offset: Specify the byte offset in the PDO (0 to 7).

Bit: Specify the bit number in the byte (0 to 7).

Size: Since this is the TPDO1/RPDO1, select Bit.

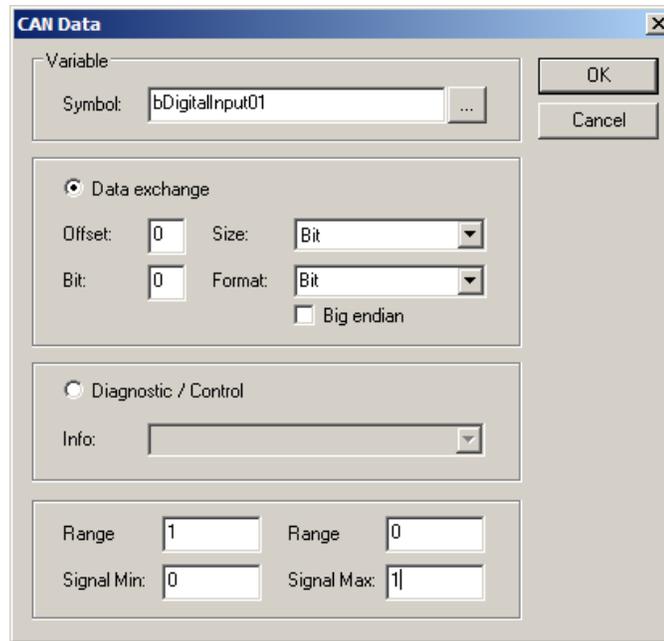
Format: Since this is the TPDO1/RPDO1, select Bit.

Big endian: Inversion of the MSB and LSB bytes in the case of a word. Not required for CANopen systems.

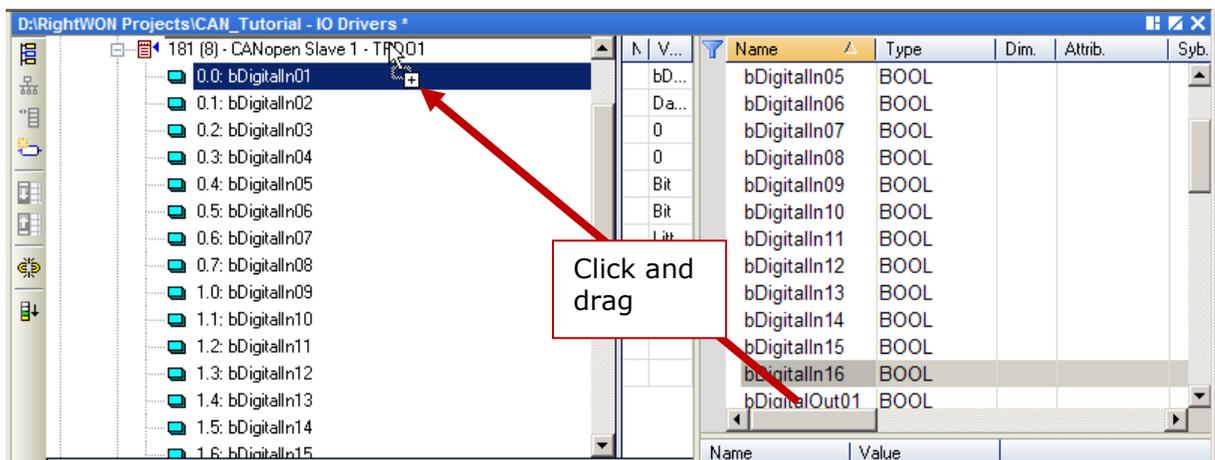
Diagnostic/Control: Check to declare variables relating to exchange control (not required for now).

Range: Defines the operating range of the variable. Not normally used for digital I/Os unless performing data inversion.

Signal Min/Max: Defines the operating range of the input/output. Not normally used for digital I/Os unless performing data inversion. The following example illustrates inversion of an output bit.



- 4- If the variables have already been created, you can make the assignments from the variables area to the PDO.



- 5- For the purposes of this tutorial, configure 16 digital inputs *bDigitalIn01* to *bDigitalIn16*, and assign them to the first 16 entries of TPDO1. Also configure 16 digital outputs *bDigitalOut01* to *bDigitalOut16*, and assign them to the first 16 outputs of RPDO1.

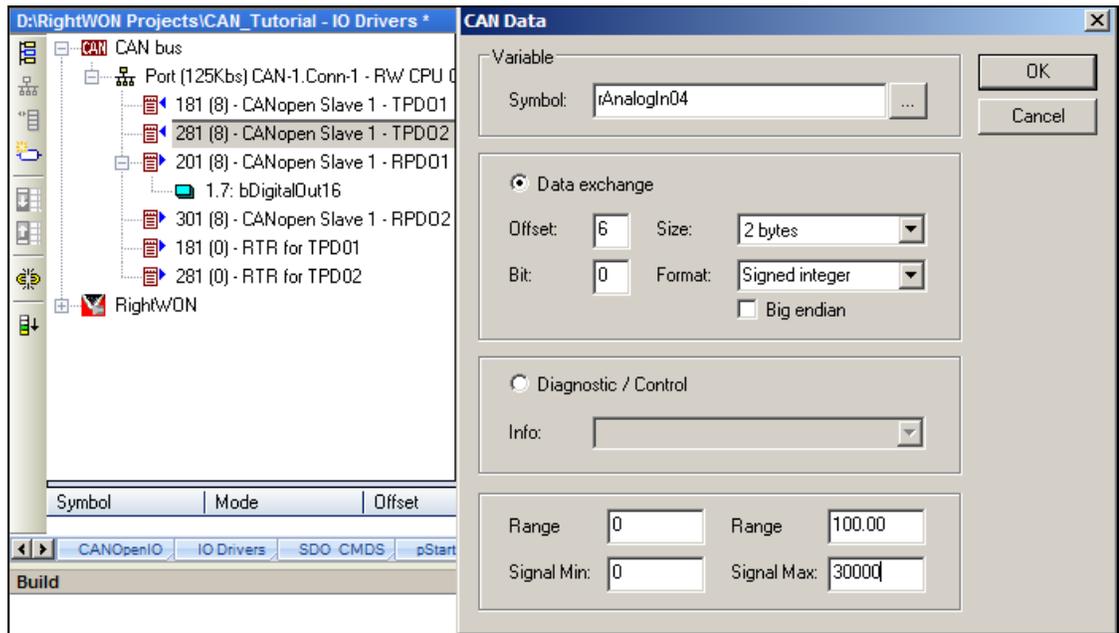
4.8.2. Associating variables with analog inputs/outputs

Analog inputs are associated with the PDOs as follows:

- TPDO2:** PDO associated with analog inputs 1 to 4 on the CANopen coupler.
- TPDO3:** PDO associated with analog inputs 5 to 8 on the CANopen coupler.
- TPDO4:** PDO associated with analog inputs 9 to 12 on the CANopen coupler.
- RPDO2:** PDO associated with analog outputs 1 to 4 on the CANopen coupler.
- RPDO3:** PDO associated with analog outputs 5 to 8 on the CANopen coupler.
- RPDO4:** PDO associated with analog outputs 9 to 12 on the CANopen coupler.

To insert an analog variable in a PDO, carry out the following steps:

- 1- In the **IO Drivers** area, right click on the desired PDO and select **Insert Variable...** 
- 2- The CAN Data configuration window appears. This window permits assigning up to four 16-bit variables to the PDO using an offset system that designates byte 0 to 7 of the PDO. For further details, refer to "[Function of PDOs associated with analog inputs/outputs](#)".



- 3- Define the following fields:

Symbol: The name of the variable associated with the input/output. The  button permits inserting a variable.

Data exchange: Select for data exchange.

Offset: Specify the byte offset in the PDO (0, 2, 4 or 6 for 16-bit data).

Bit: Not used; must be left at 0.

Size: Select 1 byte for 8-bit data or 2 bytes for 16-bit data. In the case of 12-bit converters, choose 2 bytes.

Format: Choose according to the input/output model used. Usually, unipolar 0-10V or 4-20mA inputs/outputs are signed integers.

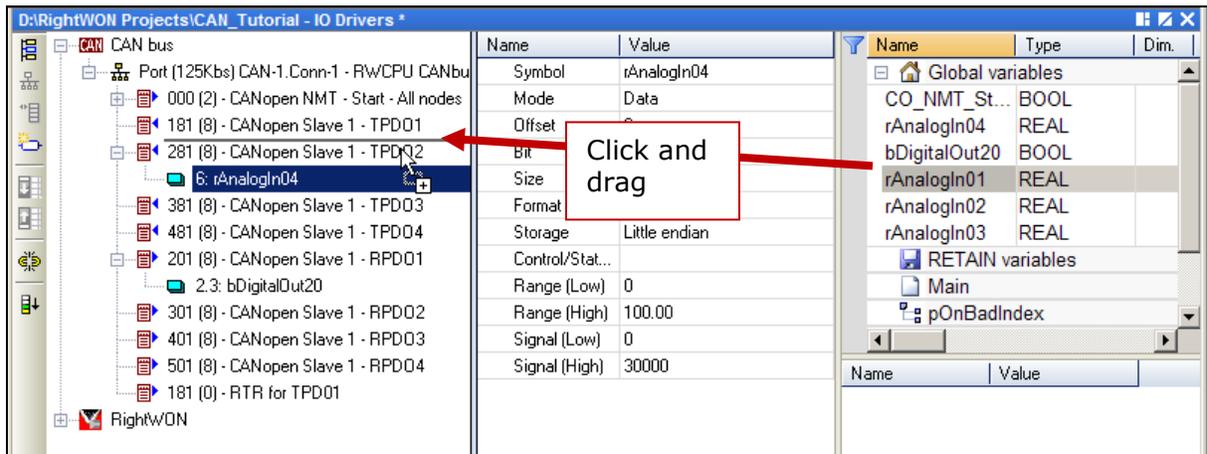
Big endian: Inversion of the MSB and LSB bytes in the data. Not required for CANopen systems.

Diagnostic/Control: Check to declare variables relating to exchange control (not required for now).

Range: Defines the operating range of the variable as an engineering value that relates to the module.

Signal Min/Max: Defines the operating range of the input/output. The example shows an analog input coming from a Phoenix Contact IB IL AI 2/SF module. The module converts 4-20mA to a rough number between 0 and 30000 (Signal Min and Max). The variable associated with the input varies from 0 to 100.00 (Range).

- 4- If the variables have already been created, you can make the assignments from the variables area to the PDO.



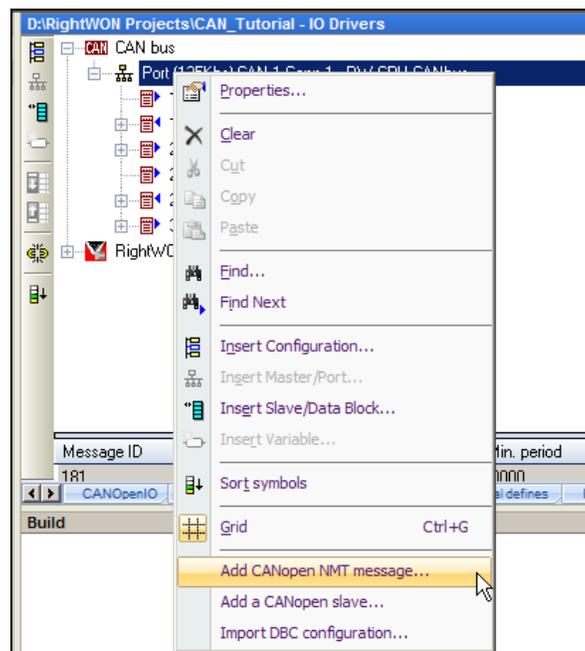
- 5- For the purposes of this tutorial, configure 4 analog inputs *rAnalogIn01* to *rAnalogIn04*, and assign them to the first 4 entries of TPDO2. Also configure 4 analog outputs *rAnalogOut01* to *rAnalogOut04*, and assign them to the first 4 outputs of RPDO2.

4.9. Creating a CANopen NMT message

CANopen NMT messages allow a program to send commands that change the status of CANopen nodes. An NMT START NODE message is required to bring the CANopen coupler online.

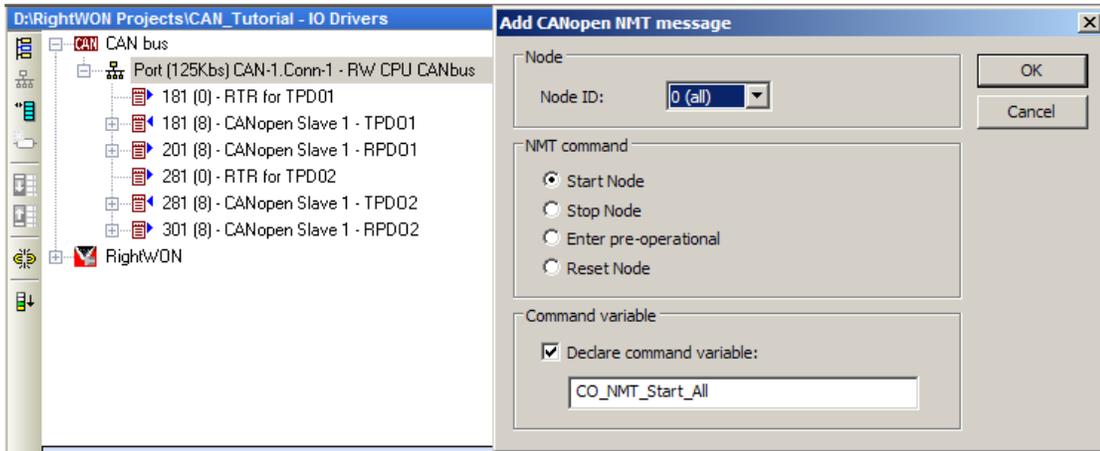
Carry out the following steps to insert a CANopen message.

- 1- In the **IO Drivers** area, right click on **Port CAN-1.Conn-1** and select **Add CANopen NMT message...**



2- The **Add CANopen NMT message** configuration window opens. Here is the list of parameters:

- In the **Node** section, select the identification number for the destination node (COB-ID).
Note: 0 (all) permits sending the message to all nodes.
- In the **NMT command** section, click on the desired action. For the present example, select **Start Node**.
- In the **Command variable** section, enter the name of the variable that will trigger sending the command. This variable will be used in the program to trigger message dispatch.



3- Click **OK** to insert the message.

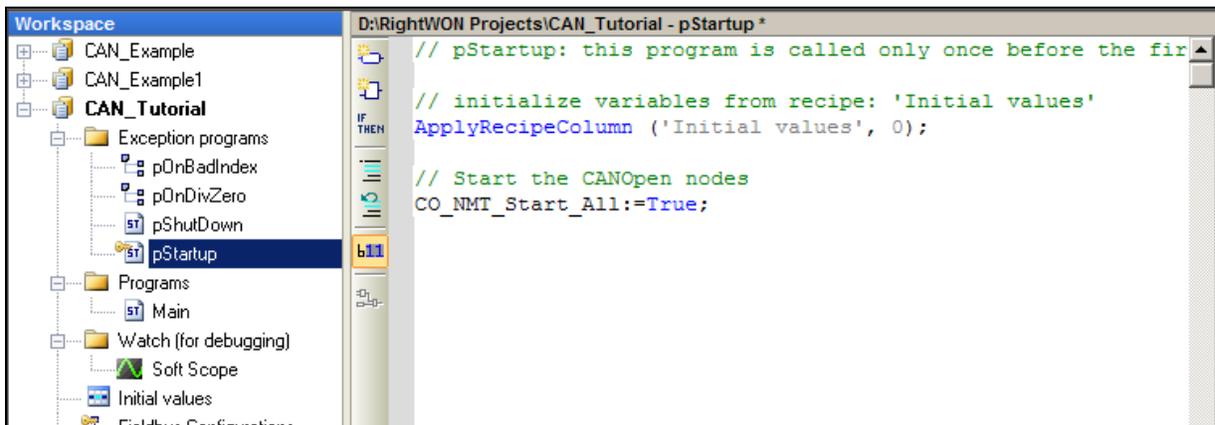


After inserting a CANopen NMT message, you must start command dispatch from a program by manipulating the variable associated with the command.

After CAN coupler power-up you must [start the node](#).

4.10. Starting the nodes with a CANopen NMT message

You must start the nodes to put the CANbus into operational state. To do this, create an NMT message that starts the nodes, and trigger the message dispatch from the startup exception program (pStartup). To do this, simply raise the *CO_NMT_Start_All* request in the pStartup program, which is called only once on RightWON startup. Once this request is sent, the driver automatically resets it to 0.



4.11. Launching the project

Execute steps 1 to 10 of the [CANbus configuration tutorial](#). This tutorial permits the exchange of analog and digital inputs/outputs between the RightWON and the CANopen coupler.

The tutorial is carried out based on a system with 16 digital inputs/outputs and 4 analog inputs/outputs. You can modify it to suit your needs by removing unused PDOs and/or unused variables.

4.11.1. Preparing the CANopen coupler

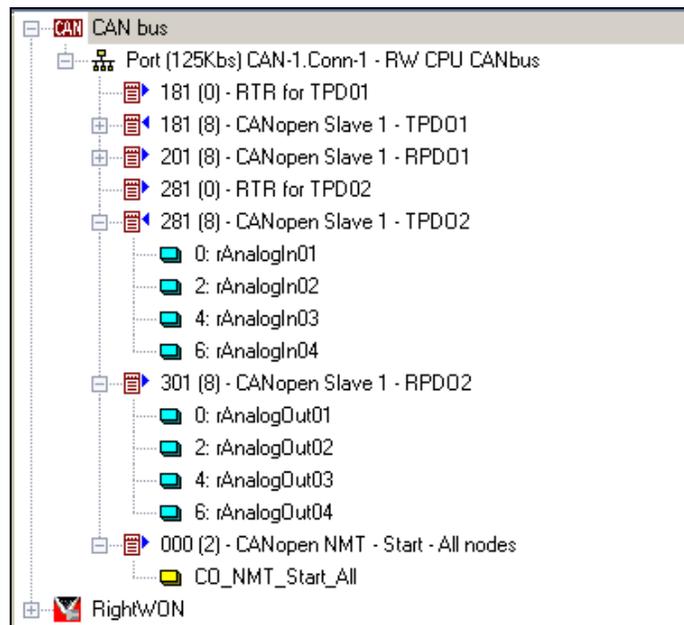
Connect your coupler to the RightWON using a shielded twisted cable. The pin assignment of the CANbus connector on the RightWON, located on the bottom of the unit, is as follows (refer to the specifications in the CPU - RWU010000-SP-en module):

Pin number	Assignment
1 (front)	Protective ground. Connect to cable shield
2	Not connected
3	CAN GND (reference GND)
4	CAN L
5	CAN H
6 (rear)	120-ohm termination. Jump to pin 5 to insert the termination.

For the tutorial example, configure the coupler to a rate of 125 Kbps and an address of 1.

4.11.2. Preparing the RightWON application

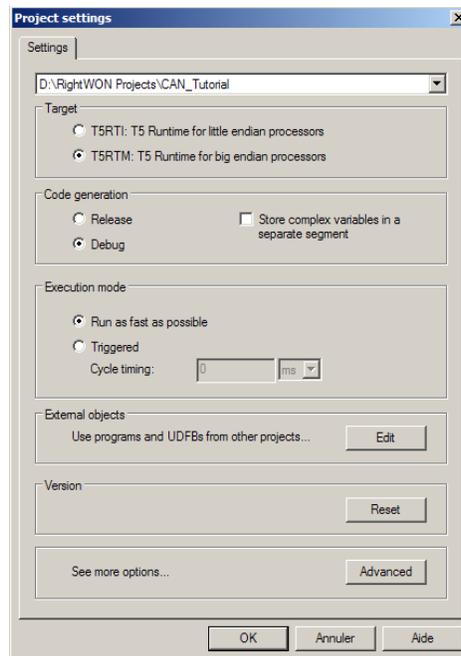
Before compiling the project and bringing it online, ensure that the tree view of the **CANbus** configuration for the tutorial looks like the following figure. Note that TPDO1 and RPDO1 should be assigned with 16 variables each.



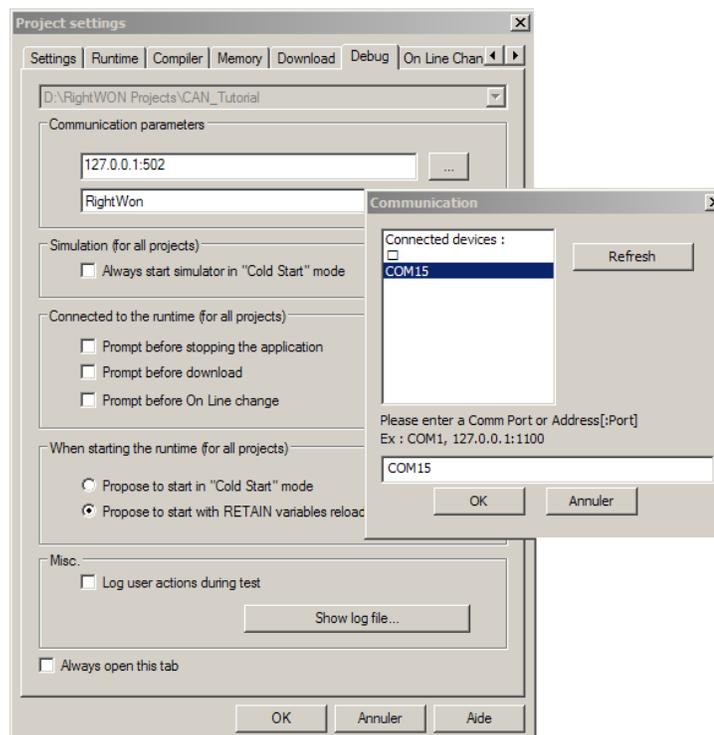
4.11.3. Preparing the project

Before bringing the project online, you must set the project options:

- 1- Execute the **Project/Settings...** command and check the **T5RTM: T5 Runtime for big endian processors** box in order to generate the RightWON code.

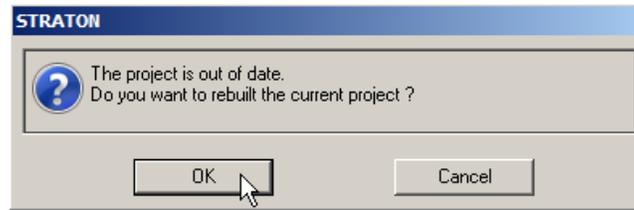


- 2- Execute the **Tools/Communication Parameters...** command, then click the  button to select the communication port that will establish communication between the RightWON and your computer running the RightWON Configuration Suite.

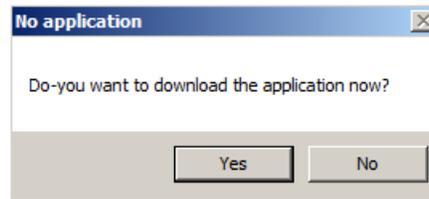


4.11.4. Bringing the project online

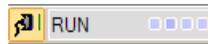
- 1- Click on the **Online**  icon (CTRL+F5) to compile and download the project. Click **OK** to recompile the project.



- 2- Click on **YES** to download the new version of the application.



- 3- The project should now be operational in the RightWON system.

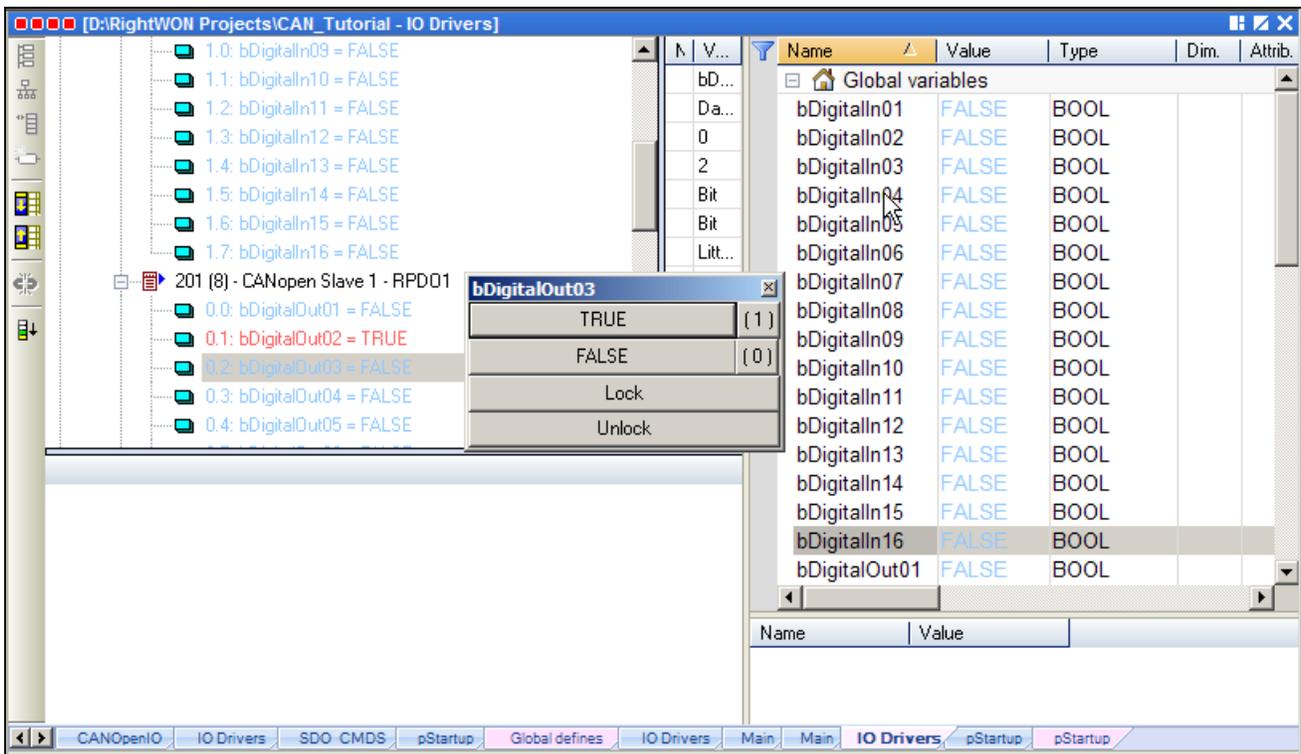


4.11.5. Operation of the CANbus example

The CANbus example permits sending and receiving data between the RightWON and CANopen node 1. The data can be viewed from the variables area or the tree view of the CANbus configuration. In the following example, a voltage of 10V is injected into the first entry (a value of 100.0133 with scaling taken into account). The 3 other analog inputs show a negative number, which indicates that they are out of range.

Name	Value	Type	Dim.	Attrib.
bDigitalOut14	FALSE	BOOL		
bDigitalOut15	FALSE	BOOL		
bDigitalOut16	FALSE	BOOL		
CO_NMT_St...	FALSE	BOOL		
rAnalogIn01	100.0133...	REAL		
rAnalogIn02	-109.220...	REAL		
rAnalogIn03	-109.220...	REAL		
rAnalogIn04	-109.220...	REAL		
rAnalogOut01	0.000000	REAL		
rAnalogOut02	0.000000	REAL		
rAnalogOut03	0.000000	REAL		
rAnalogOut04	0.000000	REAL		

Only the output variables can be forced by the user. To force a digital output, double click on the associated variable in the **I/O Drivers** area or the variables area. Then force the variable to TRUE or FALSE. The same method can be used to update analog outputs.

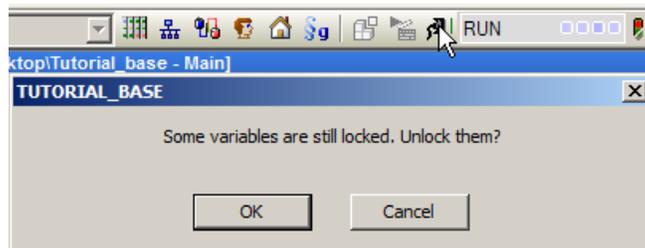


4.11.6. Exiting from online mode

To exit from online mode, click on the **Online** icon. The target program will continue to run, but will not communicate with your computer.



If a window opens warning you that some variables are still locked, click **OK** to unlock them.





Advanced CANopen Protocol Management

This section describes advanced management of the CANopen protocol. It addresses, among other things:

[Dictionary management \(SDO\)](#), describing the use of SDO for of configuring I/O modules in the object dictionary.

[Example of using SDO requests for initialization](#), show an example of using SDO request for configuring I/O modules and initializing the nodes.

[Adding more PDOs to a CANopen configuration](#), describe how to add PDOs to a system to support a larger number of inputs/outputs.

[Description of the CAN_PDO_Example](#), show an example of adding more PDOs to a CANopen configuration (e.g.: add a TRPD05 with COB-ID identifier 182 (TPDO1 of node 2)).

[Using the CANSNDMSG and CANRCVMSG functions](#), these functions permits writing and reading data on the CANbus from an application program rather than the fieldbus configurator.

5.1. Dictionary management (SDO)

SDO (Service Data Object) services are used to write or read objects in the object dictionary (OD) of CANopen modules. With SDOs, the behavior of I/O modules can be changed, for example:

- Changing the operating mode of analog inputs so they will send changes whenever the deadband is exceeded,
- Changing the message identifiers for systems that exceed the basic configuration,
- Changing the configuration of a module, etc.

Like RPDOs and TPDOs, SDOs are identified through the COB message identifier which is composed of the basic identifier plus the node number. Thus a message initiated by the RightWON to the CANopen controller will carry an identifier of 604 for node 4, and the response to this message will carry COB 584.

SDO	Basic identification for node 1 (hexadecimal)	Basic identification for node 4 (hexadecimal)
RightWON → CANopen Controller	601	604
CANopen Controller → RightWON	581	584

Each request sent to the CANopen controller (601 or higher) generates a response from the coupler. This could be data that is read by the RightWON or a confirmation of the write operation.

Each SDO message usually includes 8 bytes:

0	1	2	3	4	5	6	7
Command	Index		Subindex	Data to be exchanged (as required)			

Command: (SDO command identifier): Specifies the type of data exchange. The most commonly used commands are:

Initiate domain download: a write request to the CANbus coupler:

0x23: 4-byte transfer

0x2b: 2-byte transfer

0x2f: 1-byte transfer

Initiate domain upload: a read request from the coupler to the RightWON:

0x40: Data read

Index: Object index. For example, index 0x6006 indicates which 8-bit digital inputs trigger the dispatch of an RPDO1 message to the RightWON (Interrupt mask any change 8-bit)

Subindex: Object sub-index. Usually designates an I/O point of the module. For example, sub-index 2 of index 0x6006 refers to the second group of 8 bits.

Data: Designates the data to be exchanged. This field is ignored during a read request from the coupler.

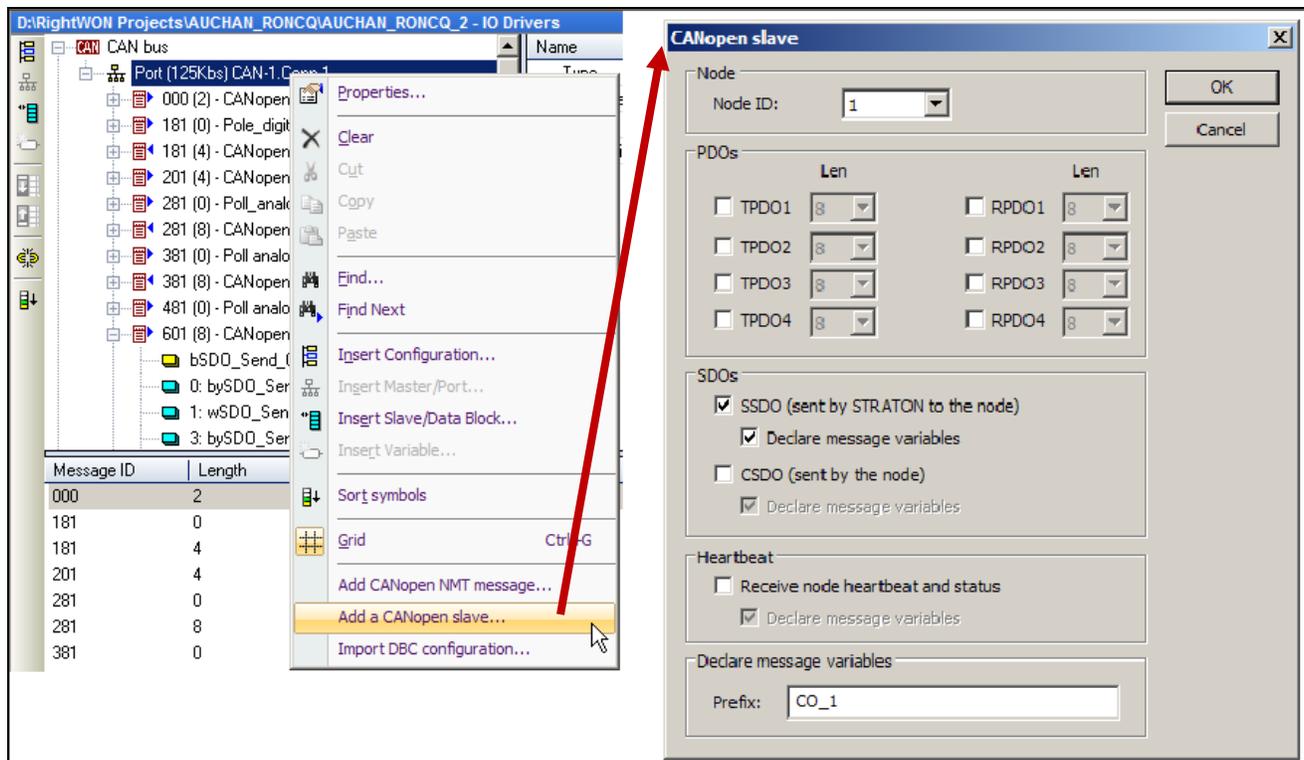
5.1.1. Defining SDO messages in the CANbus driver

Two methods can be used to define the SDO messages in the CANbus driver:

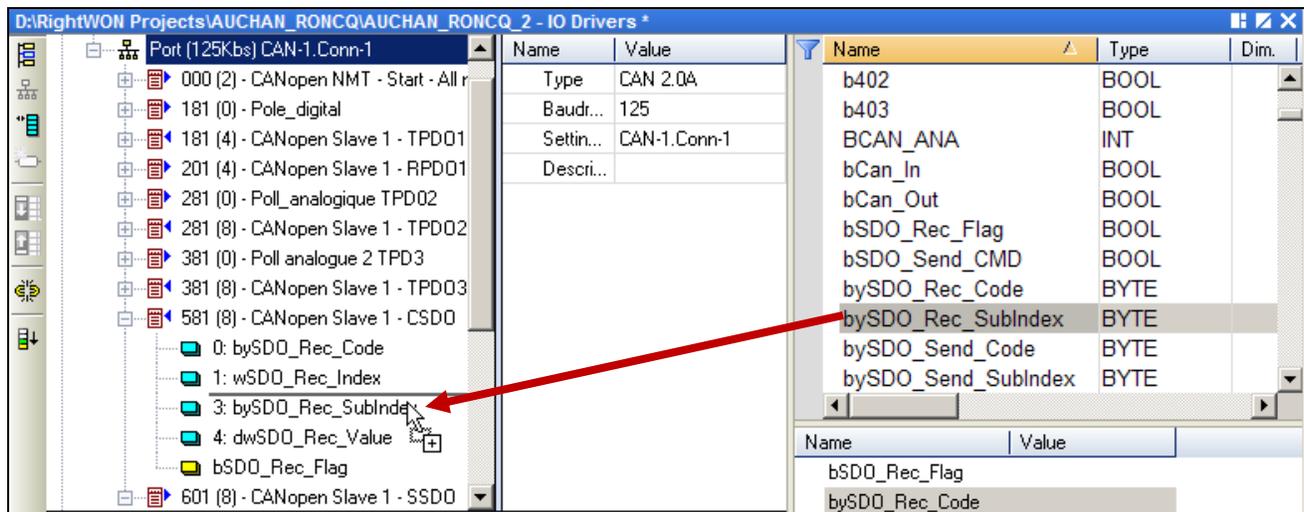
1. [Adding a CANopen slave](#)
2. [Adding a message](#) ("Insert Slave/Data block")

5.1.2. Adding a CANopen slave

- 1- Access the "Add a CANopen slave" command by right clicking on the CANbus port.
- 2- Uncheck all the PDOs, and check SSDO for requests sent by the RightWON to the SDO (0x601 and higher) or check CSDO for responses from the coupler to the RightWON (0x581 and higher).
- 3- Check "Declare message variables" to automatically declare 5 variables associated with the request.
- 4- If you have chosen to declare variables automatically, identify the prefix for the variables in the "Prefix" field.



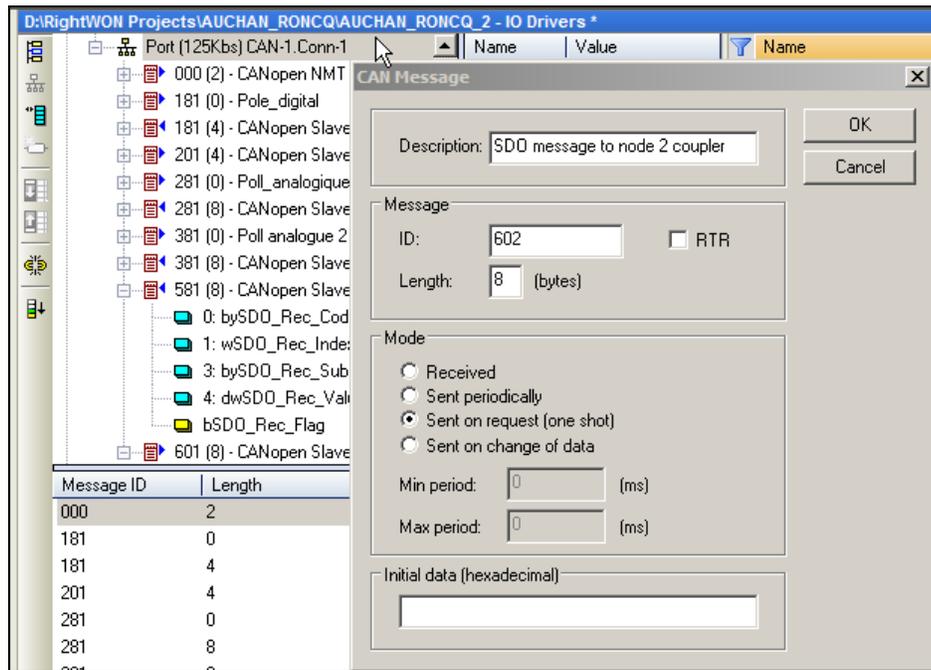
- 5- Rename the variables, if necessary. According to programming standards used by Vizimax, the prefixes bVariable, byVariable, wVariable and dwVariable denote a bit, byte, word and double word, respectively.
- 6- You can reassign the variables to the requests by dragging and dropping selected variables from the variables area into the requests.
- 7- Repeat steps 1-6 to generate the CSDO response request.



5.1.3. Adding a message (“Insert Slave/Data block”)

- 1- Select the CANbus port and click “Insert Slave/Data block” ( tool).

- 2- Enter the message description, COB message identifier (600 + node address), message length (8 bytes) and select *Sent on request*.

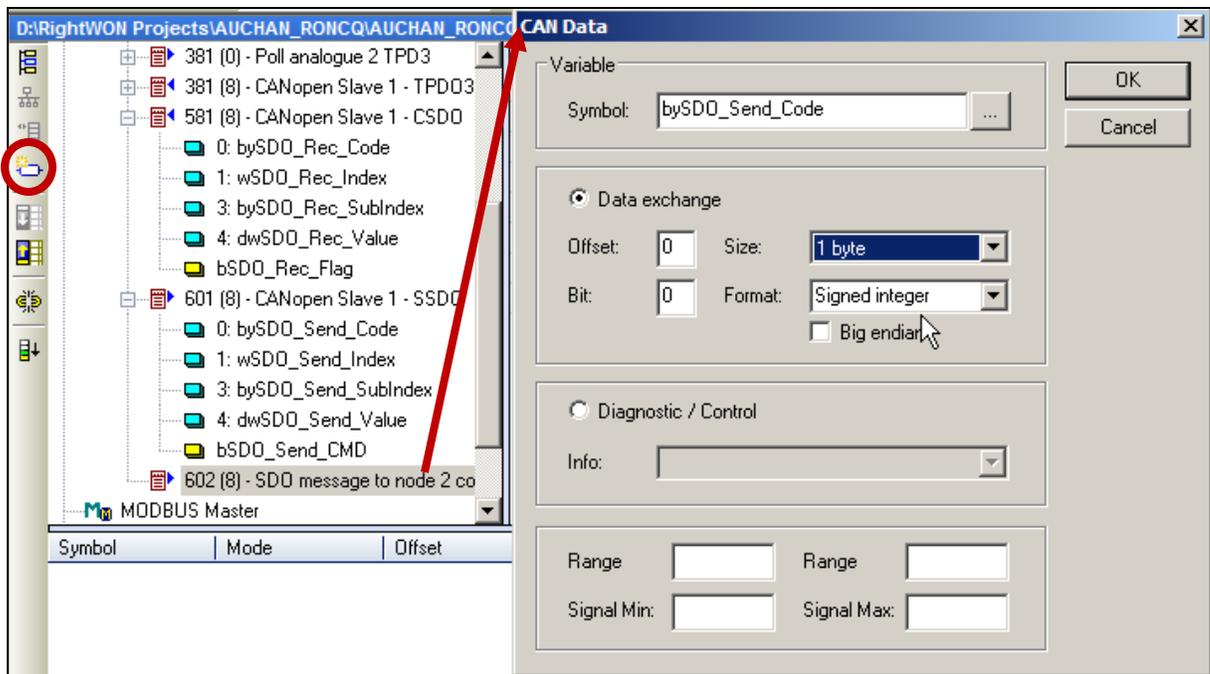


- 3- In the variables area, define the 5 variables used in the exchange:

- Command
- Index
- Sub-index
- Data to be exchanged
- Send bit of the SDO request



- 4- Associate the variables one by one with the requests using the "Add variable" command. Alternatively, you can assign the variables to the requests by dragging and dropping the selected variables from the variables area into the requests.



- 5- Repeat steps 1-4 to define the response (with COB message identifier 580 + node address).

5.2. Example of using SDO requests for initialization

The following example, "Init_Example", illustrates the configuration of 4 modules, each with 2 Phoenix Contact IB IL AI 2/SF analog inputs. These modules are configured as 0-10V by default, and are reassigned as 4-20mA inputs.

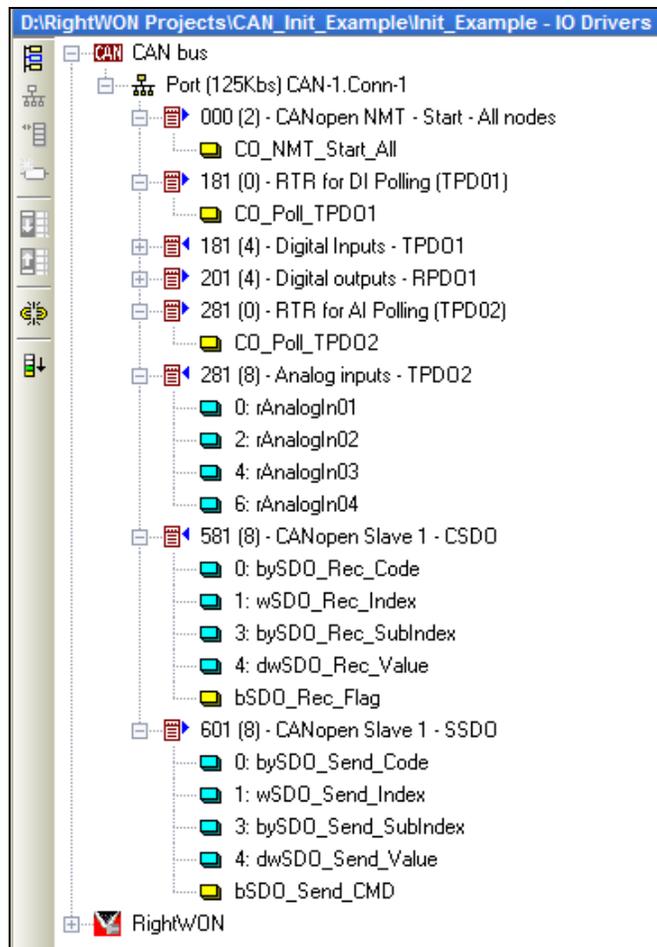
According to the module and CANopen controller documentation, the following actions must be performed:

Write the 0x800A data (one-word command = 0x2B) to index 0x2400 (AIP range) for the 8 sub-indexes from 1 to 8 (first 8 analog inputs).

This command is carried out before node startup.

5.2.1. CANopen configuration

In this example, the analog and digital inputs are returned through an on-demand RTR request (programmable). The SDOs are managed by blocks 581 and 601.



5.2.2. Startup program (pStartup)

The startup program initializes some of the variables required for management of the CANopen coupler.

```

D:\RightWON Projects\CAN_Init_Example\Init_Example - pStartup *
// CANbus init : variables are GLOBAL
iCAN_Step:=0;           // Step number
CO_NMT_Start_All:= False; // Start node command
IF THEN
CO_Poll_TPD01:= False; // Digital input polling
CO_Poll_TPD02:= False; // Polling of the first 4 analog inputs

```

5.2.3. CANopen management program

In steps 0 to 15 (iCAN_Step), the CANopen_Init program does the following:

- 1- Sends successive SDO write requests to the controller, specifying each time a sub-index from 1 to 8,
- 2- Waits for the response from the controller for each request. Note that the response is not validated.

Subsequently, the nodes are started through an NMT request (step 16). The analog and digital inputs are returned on the rising edge (r_trig function) of Blink timers, which sends an RTR request on demand.

```

// Initialize the data to be sent on the CANOpen controller
bSDO_Send_CMD:=0;
bySDO_Send_Code:=16#2b; // Command : Write 2 bytes
wSDO_Send_Index:=16#2400; // Data : Phoenix modules AIP Range
dwSDO_Send_Value:=16#800A; // Configuration data, 16-bit mean, IB IL, 4-20mA

CASE iCAN_Step OF
  //Initialize the first 8 analog inputs
  0,2,4,6,8,10,12,14:
    // The sub-index designate the channel number.
    // Index 0 is the number of analog inputs
    bySDO_Send_SubIndex:= ANY_TO_USINT(1 + (iCAN_Step/2));
    // Go to next step
    iCAN_Step := iCAN_Step + 1;
    // Send the SDO command
    bSDO_Send_CMD:=True;
  // Wait for the SDO response
  1,3,5,7,9,11,13,15:
    // The send command is valid for one cycle
    bSDO_Send_CMD:=False;
    // Flag should be on when receiving the answer
    if bSDO_Rec_Flag = True then
      iCAN_Step := iCAN_Step + 1; // Go to next step
    end_if;
  16:
    // Start the nodes
    CO_NMT_Start_All := True; // Noeuds CANOpen initialisés
    iCAN_Step := iCAN_Step + 1; // Go to next step
  17:
    // Periodic scanning of the inputs
    // Timer for DIs
    T_Blink_DI(True, T#60000ms); // See Blink timer in Help
    T_Blink_AI(True, T#1000ms);
    // Detect rising edge of timers
    r_trig_DI(T_Blink_DI.Q); // See r_trig function in Help
    r_trig_AI(T_Blink_AI.Q);
    // Each Poll request must have an independant variable
    CO_Poll_IPDO1 := r_trig_DI.Q;
    CO_Poll_IPDO2 := r_trig_AI.Q;
  ELSE
    iCAN_Step := 17;
END_CASE;

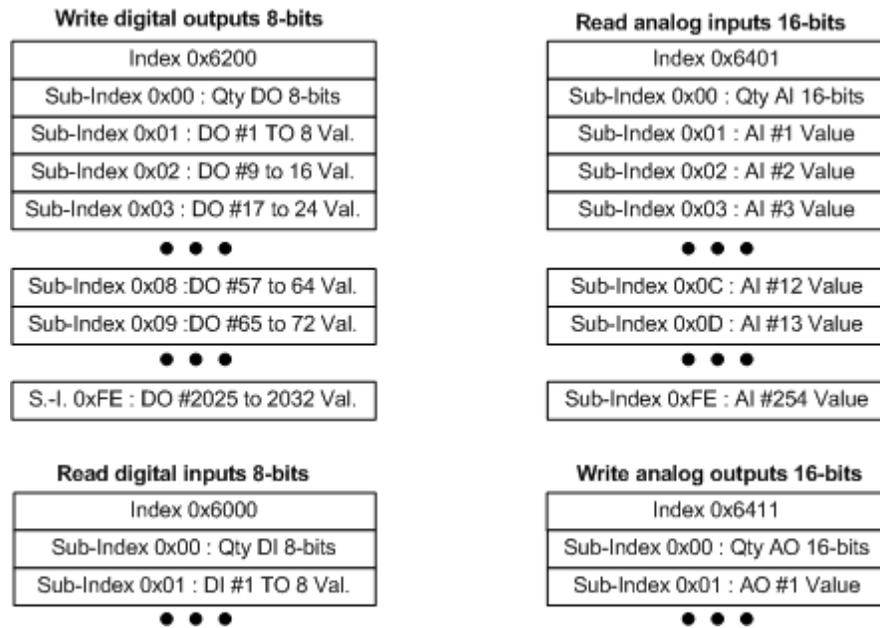
```

5.3. Adding more PDOs to a CANopen configuration

As previously mentioned, CANopen couplers that comply with the DS401 specification can support up to 12 analog inputs, 12 analog outputs, 64 digital inputs and 64 digital outputs without any configuration. The "CAN_PDO_Example" example illustrates how to add PDOs to a system to support a larger number of inputs/outputs. Before [describing the CAN_PDO_Example project](#), the [basic principles for using registers associated with the PDOs](#) are described.

5.3.1. SDO read/write commands on inputs/outputs

The RightWON is able to read inputs and update outputs using command/response SDOs. In the following figure, a read request (0x40) at index 0x6401, sub-index 00 returns a byte indicating the number of 16-bit analog inputs connected to the CANopen coupler. A read of index 0x6401, sub-index 12 (0x0C) returns the value of analog input #12 as a 16-bit signed integer.



In the same way, it is possible to update 8 bits on the third output module using a single-byte write command byte (0x2F) at index 0x6200, sub-index 3. This read/write method for additional inputs/outputs exceeding the basic PDOs is supported by the RightWON, but it generates a considerable amount of communication traffic on the CANbus. It is better to create additional PDOs using the PDO mapping registers.

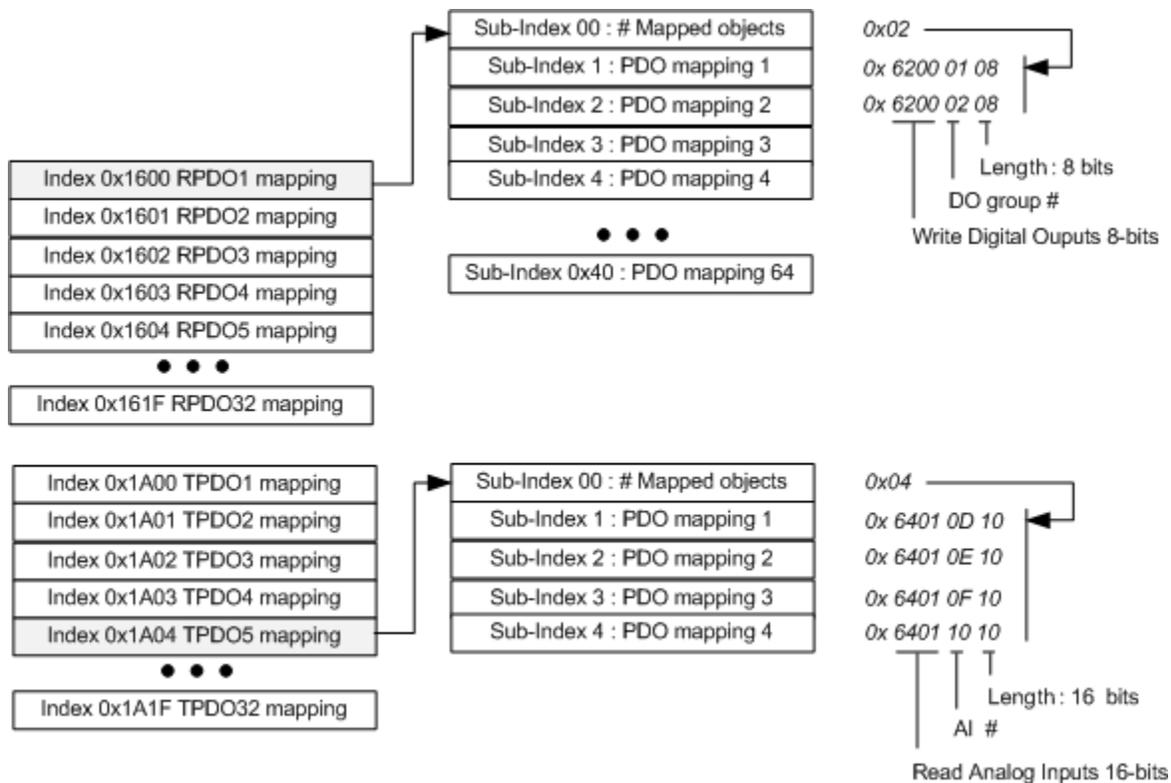
5.3.2. PDO mapping registers

Mapping registers permit assigning the contents to the PDOs. The number of mapping registers depends on the coupler manufacturer. For example, the Phoenix Contact IL CAN BK-TC CANopen coupler supports the definition of a maximum of 32 RPDOs and 32 TPDOs, including the basic PDOs (4 of each type). As shown in the following figure, there is registration (indexing) by PDO. Sub-index 0 contains the number of objects to be mapped, and the following sub-indexes define the object mapping, that is, the list of [SDO read/write commands](#) to be performed on the modules.

The example in the following figure illustrates the mapping of RPDO1 (index 0x1600) on a coupler equipped with two 8-bit digital output modules (sub-index 00 = 2). Sub-indexes 1 and 2 contain the 8-bit SDO write command for the first and second modules, respectively. If 72 digital outputs had to be supported, the first 64 outputs would be automatically assigned to RPDO1. It would then be necessary to create, for example, RPDO5 (index 0x1605) and assign one object (sub-index 00) with a mapping of 0x6200 09 08 (9th group of 8-bit outputs).

The example also illustrates the mapping of TPDO5 (index 1A04), which has 4 analog inputs (inputs #13 to #16). The mapping contains the number (sub-index 0 = 4) and the list of TPDO inputs that are put into read commands.

In addition to defining the PDO mapping, you must also [assign the COB-ID identifier](#).



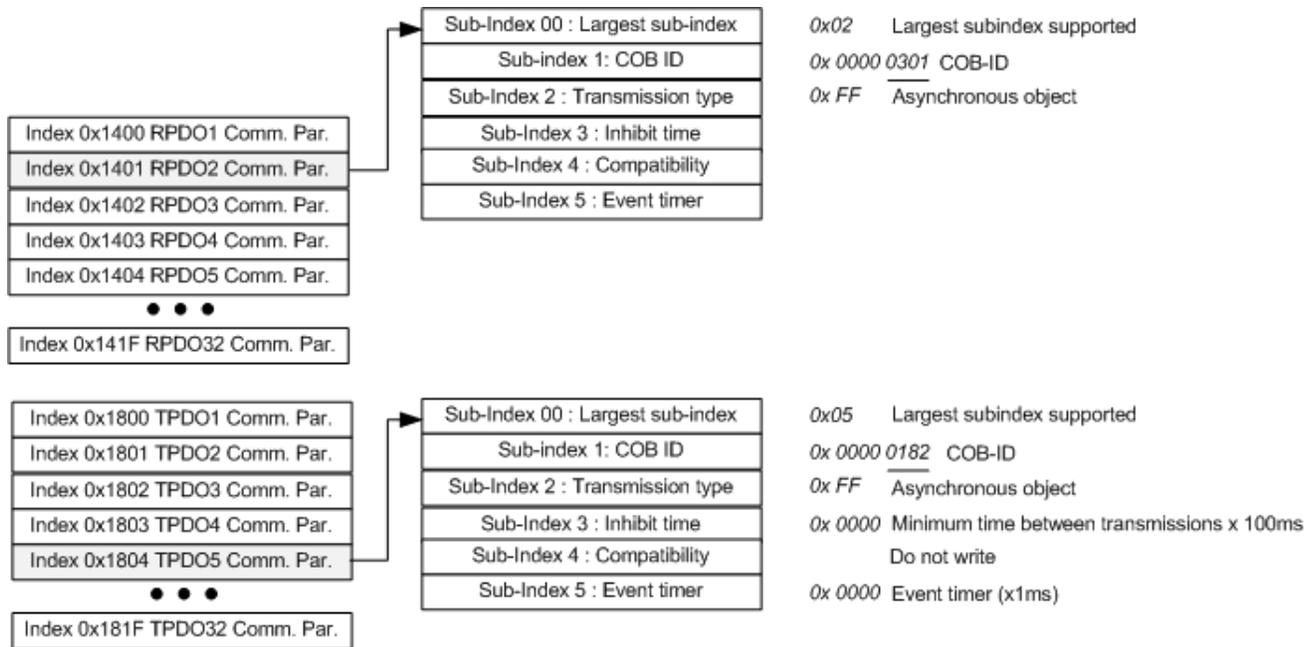
5.3.3. Assigning an identifier to a PDO

The identifier and behavior of RPDOs and TPDOs are managed by communication parameter registers, shown in the following figure. One register is associated with each PDO. The programming and number of registers vary, depending on the CANopen coupler manufacturer. For example, the Phoenix Contact IL CAN BK-TC CANopen coupler supports the definition of a maximum of 32 RPDOs and 32 TPDOs, including the basic PDOs (4 of each type).

The example in the following figure shows the default configuration of a system with at least 4 analog outputs. Generally, RPDOs support only 2 sub-index inputs (sub-index 0 = 2). Sub-index 1 is equal to the COB-ID of the RPDO (301 in the case of RPD02, or 0x80000000 if the RPDO of the corresponding index does not exist). Index 02 is equal to 0xFF for asynchronous RPDOs (not connected to a SYNC command; updated directly or by RTR request).

The example also illustrates the configuration of TPDO5. Identifier 182 (TPDO1 on node 2) was chosen for TPDO5, since the basic CANopen protocol supports only 8 PDO identifiers per node. An RTR request to TPDO1 on node 2 forces the coupler at node 1 to send its TPDO5.

Note that strict sequencing when [defining and assigning PDOs](#) must be respected.



5.3.4. PDO definition and assignment sequence

Creation or modification of a PDO mapping and its assignment to an identifier should be carried out according to the following procedure:

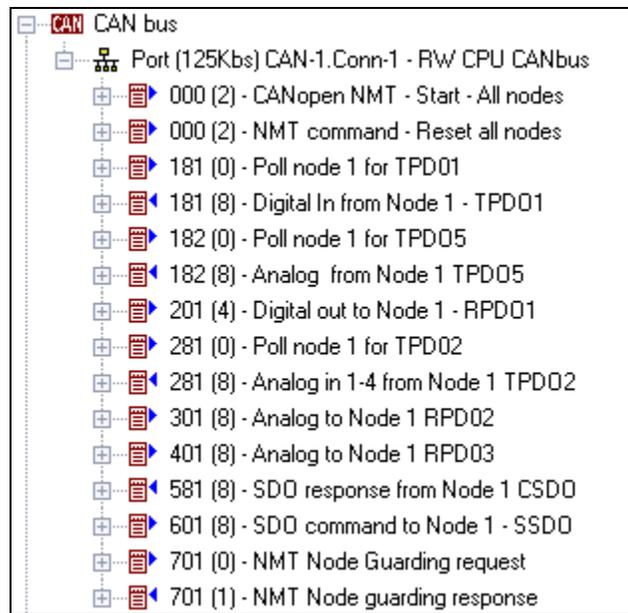
- 1- Choose the index of the communication parameter register based on the selected PDO
- 2- Disable the identifier assignment by writing 0x8000 0000 to sub-index 1 (COB-ID)
- 3- Choose the index of the mapping register based on the PDO to be created (see "PDO mapping registers")
- 4- Write to byte 0 at sub-index 0 to destroy the current mapping
- 5- For each mapping to be carried out (sub-index 1 to *n* mappings), write a double word that contains the command to be performed on the module
- 6- At sub-index 0, write the *n* number of mappings to be carried out
- 7- Assign a COB-ID identifier to the PDO that has been created

5.4. Description of the CAN_PDO_Example project

The CAN_PDO_Example program provides an example where TPDO5 is defined with a COB-ID of 182 (TPDO1 on node 2). The first 2 analog inputs from the coupler are assigned to TPDO5 in addition to TPDO2.

5.4.1. CANopen configuration

In this example, the analog and digital inputs are returned through on-demand RTR request (programmable). The SDOs are managed by blocks 581 and 601. In addition, the program monitors the coupler with NMT Node Guarding RTR requests (701).



5.4.2. Definitions

The following symbols are used in the program. These definitions are global.

```
// CANbus defines

#define WriteByte BYTE16#2F // SDO command identifier for initiate domain download e,s=1, n=3
#define WriteWord BYTE16#2B // SDO command identifier for initiate domain download e,s=1, n=2
#define WriteDWord BYTE16#23 // SDO command identifier for initiate domain download e,s=1, n=0
#define ReadSDC BYTE16#40 // SDO command identifier for initiate domain upload
#define Preoperational 16#7F // CAN coupler is preoperational
#define Operational 16#05 // CAN coupler is operational
#define NodeGuardAnswerMask 16#7F // Node Guard Answer mask
#define SDOAnswerMask 16#E0 // SDO command answer mask
#define SDODownloadAnswer 16#60 // SDO Initiate Domain Download valid answer
#define CANInit 0 // CANbus initialization in progress
#define CANInitError -1 // CANbus initialization error
#define CANRunning 1 // CANbus is now running
#define CAN_NoHBAnswer -2 // CANbus no answer to heart beat request
#define CAN_BadNode -3 // CANbus bad answer to heart beat
```

5.4.3. Startup program (pStartup)

The startup program initializes the iCAN_Step variable, required for managing the CANopen coupler.

```
D:\RightWON Projects\CAN_PDO_Example - pStartup
// pStartup: this program is executed once
// and continues with the other programs
// in the same cycle

// CANbus variables initialization
iCAN_Step:=0; // Step number of the CANbus initialization
```

5.4.4. Command recipe

SDO commands are programmed in the SDO_CMDS recipe such that they can be handled in sequence. The recipe contains all the fields of the SDO command:

bySDO_Send_Code: SDO command identifier (see "[Dictionary management \(SDO\)](#)").

wSDO_Send_Index: Object dictionary index

bySDO_Send_SubIndex: Object dictionary sub-index

dwSDO_Send_Value: Data to be written to the dictionary

The *wSDO_Send_Index* field associated with the first column (0) contains the number of SDO commands contained in the recipe.

D:\RightWON Projects\CAN_PDO_Example - SDO_CMDS.rcp						
Name	Value	0	1	2	3	4
bySDO_Send_Code	BYTE#0	BYTE#16#23	BYTE#16#2F	BYTE#16#23	BYTE#16#23	BYTE#16#23
wSDO_Send_Index	WORD#8	Word#16#1804	WORD#16#1A04	WORD#16#1A04	WORD#16#1A04	WORD#16#1A04
bySDO_Send_SubIndex	BYTE#0	BYTE#16#01	BYTE#16#0	BYTE#16#01	BYTE#16#01	BYTE#16#02
dwSDO_Send_Value	DWORD#0	DWORD#16#80000000	DWORD#16#0	DWORD#16#64010110	DWORD#16#64010110	DWORD#16#64010210
Name	Value	5	6	7	8	
bySDO_Send_Code	BYTE#16#2F	BYTE#16#23	BYTE#16#2B	BYTE#16#2B	BYTE#16#2B	
wSDO_Send_Index	WORD#16#1A04	Word#16#1804	Word#16#2400	Word#16#2400	Word#16#2400	
bySDO_Send_SubIndex	BYTE#16#0	BYTE#16#01	BYTE#16#01	BYTE#16#01	BYTE#16#02	
dwSDO_Send_Value	DWORD#16#02	DWORD#16#00000182	DWORD#16#800A	DWORD#16#800A	DWORD#16#800A	

Columns 1 to 8 perform the following functions:

- 1- Invalidation of TPDO5 (index 0x1804) by writing 0x80000000 in the COB-ID register (sub-index 1)
- 2- Invalidation of the TPDO5 mapping by writing 0 in the number of objects mapped
- 3- First mapping, corresponding to analog input #1 (also mapped in TPDO2)
- 4- Second mapping, corresponding to analog input #2 (also mapped in TPDO2)
- 5- Validation of the TPDO5 mapping by setting the number of analog inputs to 2
- 6- Validation of TPDO5 by setting its COB-ID to 182 (TPDO1 on node 2)
- 7- Setting analog input #1 to 4-20mA current input
- 8- Setting analog input #2 to 4-20mA current input

Note that the recipe can be exported/imported in CSV format for editing purposes. Steps 1-6 can be repeated to program other PDOs or entries in a PDO.

5.4.5. CANopen management program

The CANopen_Init program performs the following steps (iCAN_Step):

- 0- Resets all nodes to return to the default configuration.
- 1- Queries the status of the node with an NMT Node Guard RTR request.
- 2- Verifies whether the node is operational, and skips to step 4 if it is.
- 3- If there is no response, manages the timeout based on the T_blink_AI timer. Returns to step 1 if the node is not operational.

```

// Time base for time-out management
T_blink_AI(True, T#1000ms); // Period between each Analog input scan
r_trig_AI(T_blink_AI.Q); // Edge detect
T_blink_DI(True, T#60000ms); // Period between DI integrity Scans
r_trig_DI(T_blink_DI.Q); // Edge detect
T_HB(True, T#10000ms); // Period between heart beat
r_trig_HB(T_HB.Q); // Edge detect

// Init sequence
CASE iCAN_Step OF
  0:
    // Reset all CAN Open nodes
    bRESET_CANOpen_Nodes:=True;
    // Go to next step
    iCAN_Step:= iCAN_Step + 1;

  1:
    // Request the Node 1 status
    bNode_Guarding_Node1:=True;
    byScratchPad:=0;
    iCAN_Step:= iCAN_Step + 1;

  2:
    // Verify if Node 1 is pre-operational
    byScratchPad:= AND_MASK(byNodeGuardAnswer,NodeGuardAnswerMask);
    if (byScratchPad = Preoperational) then
      iCAN_Step:= iCAN_Step + 2; // Step 4
    else
      iCAN_Step:= iCAN_Step + 1; // Step 3
    end_if;

  3:
    // Wait until Node 1 is pre-operational (loop to 1)
    if r_trig_AI.Q then iCAN_Step:=iCAN_Step-2;
    end_if;

```

In step 4, SDO commands are prepared from the recipe.

```

4:
  // Prepare the SDO commands initialization.
  // SDO commands are stored in SDO_CMDS recipe
  // Send_index in first column (0) contains
  // the quantity of SDO commands to send
  ApplyRecipeColumn ('SDO_CMDS.rcp', 0);
  diSDO_CMDS_QTY:=ANY_TO_DINT(wSDO_Send_Index); // Number of SDO commands
  diSDO_CMDS_PTR:=1;
  iCAN_Step:= iCAN_Step + 1; // SDO commands pointer

```

The following steps are for sending and receiving SDO commands:

- 5- Sends the SDO command. Note that the command value (double word) is assigned to 4 bytes. This conversion is not required in RightWON Configuration Suite versions 1.7 and

higher. To this end, the *dwSDO_Send_Value* variable can be directly assigned to request 601 (offset 4).

- 6- Verifies the response to the SDO request. Step 5 is repeated until there are no more commands to send. If there is an error in the response, skips to step 100 and indicates the error to other applications (iCANStatus).

```

5:
  // Load the SDO command and send it
  ApplyRecipeColumn ('SDO_CMDS.rcp', diSDO_CMDS_PTR); // Load the command
  // Conversion from DWORD to Byte (overcome driver problem in version 1.6)
  bySDO_Send_Value00:=lobyte(loword(dwSDO_Send_Value));
  bySDO_Send_Value01:=hibyte(loword(dwSDO_Send_Value));
  bySDO_Send_Value02:=lobyte(hiword(dwSDO_Send_Value));
  bySDO_Send_Value03:=hibyte(hiword(dwSDO_Send_Value));
  bSDO_Send_CMD:=True; // Send the command
  iCAN_Step := iCAN_Step + 1; // Next step

6:
  // Wait for the SDO response receive flag
  if bSDO_Rec_Flag = True then
    // Validate the answer
    bSDO_Rec_Flag:=False;
    byScratchPad:= AND_MASK(bySDO_Rec_Code,SDOAnswerMask);
    if byScratchPad = SDODownloadAnswer then
      // Valid answer : Execute next command if any
      diSDO_CMDS_PTR:= diSDO_CMDS_PTR + 1; // Next command
      if diSDO_CMDS_PTR > diSDO_CMDS_QTY then
        iCAN_Step := iCAN_Step + 1; // Valid response
      else
        iCAN_Step := iCAN_Step - 1; // Loop for next
      end_if;
    else
      // Invalid answer. Indicate the error
      iCANStatus:= CANInitError; // Initialization error
      iCAN_Step:=100;
    end_if;
  end_if;

```

Steps 7 to 11 perform the following actions:

- 7- Node start-up.
- 8- RTR request to read the status of node 1.
- 9- Verifies whether the node is operational, and skips to step 11 if it is.
- 10- Manages the timeout for a response from the node. Returns to step 8 if the node is not operational.
- 11- Read request for digital inputs and preparation for periodic dispatch of TPDOs.

```
7:
// Start CANbus nodes
bSTART_CANOpen_Nodes := True; // Go in operational mode
iCAN_Step := iCAN_Step + 1; // Next step

8:
// Request the Node 1 status
bNode_Guarding_Node1:=True;
iCAN_Step:= iCAN_Step + 1;

9:
// Verify if Node 1 is Operational
byScratchPad:= AND_MASK(byNodeGuardAnswer,NodeGuardAnswerMask);
if (byScratchPad = Operational) then
    iCAN_Step:= iCAN_Step + 2; // Step 11
else
    iCAN_Step:= iCAN_Step + 1; // Step 10
end_if;

10:
// Wait until Node 1 is operational (loop to 8)
if r_trig_AI.Q then iCAN_Step:=iCAN_Step-2;
end_if;

11:
// Force a digital input scanning
bPoll_TPDO1 := True;
byNodeGuardAnswer:=0;
bHeartBeatRcv:=False;
iCANStatus := CANRunning; // Initialization done!
iCAN_Step := iCAN_Step + 1; // Next step
```

In step 12, the on-demand read requests (RTRs) digital inputs (TPDO1) and analog inputs (TPDO2 and TPDO5) are carried out periodically. In addition, there is a periodic node status read command for detecting communication errors between the RightWON and the CANopen coupler.

```

12:
// Periodic scanning of inputs
bPoll_TPDO1 := r_trig_DI.Q;
bPoll_TPDO2:= r_trig_AI.Q;
bPoll_TPDO5:= r_trig_AI.Q;

// Send the heart beat
bNode_Guarding_Nod1:=r_trig_HB.Q;

// Node Heartbeat
// Send the heartbeat at rising edge of blink timer
if bNode_Guarding_Nod1 then
    byNodeGuardAnswer:=1; // Indicate that the HB is sent
end_if;
// Verify the validity of node status if a message is received
if bHeartBeatRcv then
    bHeartBeatRcv:=False;
    byScratchPad:= AND_MASK (byNodeGuardAnswer,NodeGuardAnswerMask);
    if (byScratchPad <> Operational) then
        // Invalid answer. Indicate the error
        iCANStatus:=CAN_BadNode; // Initialization error
        iCAN_Step:=CAN_BadNode;
    end_if;
    // Otherwise check the time-out
    else
        // No answer time-out management
        if ((not (T_HB.Q)) and byNodeGuardAnswer = 1) then
            iCANStatus:= CAN_NoHBAnswer;
            iCAN_Step:=CAN_NoHBAnswer;
        end_if;
    end_if;

ELSE
    iCAN_Step := iCAN_Step;
END_CASE;

```

5.5. Using the CANSNDMSG and CANRCVMSG functions

The **CANSNDMSG** and **CANRCVMSG** functions can be used for writing and reading data on the CANbus from an application program rather than the fieldbus configurator. The **CAN_PDO_PGM** project contains the same example as the **CAN_PDO_Example** project. However, steps 5 and 6 send the SDO commands through the **CANSNDMSG** function and receive responses through the **CANRCVMSG** function, rather than carrying out the exchanges via the fieldbus configurator. The commands are sent in a table (*usCAN_TSDO_DATA[]*) that is assigned from the modified **SDO_CMDS** recipe and includes the number of bytes to send (*diTSDO_Data_Len*).

Note that responses configured through fieldbus settings are intercepted and are not passed on to the **CANRCVMSG** function. In addition, you must configure the FIFO associated with the driver as soon as the **CANSNDMSG** and **CANRCVMSG** functions are used.

The screenshot displays a software development environment with a code editor on the left and a project tree on the right. The code editor shows two sections of code, labeled 5: and 6:, which handle SDO (Service Data Object) commands and responses. Section 5: involves loading a recipe column, stuffing data into an array, and sending the command via CANopen. Section 6: involves waiting for a response, validating it, and updating the command pointer and step counter. The project tree on the right shows a hierarchy of folders including CANbus, CANopen, and various protocol-related folders.

```

5:
// Load the SDO command and send it
ApplyRecipeColumn ('SDO_CMDS.rec', diSDO_CMDS_PTR); // Load the command
// Data stuffing of recipe (for clarity purpose)
usCAN_TSDO_DATA[0]:=bySDO_Send_Code;
usCAN_TSDO_DATA[1]:=lobyte(wSDO_Send_Index);
usCAN_TSDO_DATA[2]:=hibyte(wSDO_Send_Index);
usCAN_TSDO_DATA[3]:=bySDO_Send_SubIndex;
usCAN_TSDO_DATA[4]:=lobyte(loword(dwSDO_Send_Value));
usCAN_TSDO_DATA[5]:=hibyte(loword(dwSDO_Send_Value));
usCAN_TSDO_DATA[6]:=lobyte(hiword(dwSDO_Send_Value));
usCAN_TSDO_DATA[7]:=hibyte(hiword(dwSDO_Send_Value));
bSDO_Send_CMD:=CANSNDMSG( strPORT, ucCANID, diTSDO_Data_Len, usCAN_TSDO_DATA, bRTR );
iCAN_Step := iCAN_Step + 1; // Next step

6:
// Wait for the SDO response receive flag
CAN_SDO_RCV(strPORT,usCAN_RSDO_DATA);
if ((CAN_SDO_RCV.OK = True) and (CAN_SDO_RCV.ID = SDORcvID)) then
  dw_RSDO_CANID:=CAN_SDO_RCV.ID;
  diRSDO_Data_Len:=CAN_SDO_RCV.LEN;
  // Validate the answer
  byScratchPad:= AND_MASK(usCAN_RSDO_DATA[0],SDOAnswerMask);
  if ((byScratchPad = SDODownloadAnswer)) then
    // Valid answer : Execute next command if any
    diSDO_CMDS_PTR:= diSDO_CMDS_PTR + 1; // Next command
    if diSDO_CMDS_PTR > diSDO_CMDS_QTY then
      iCAN_Step := iCAN_Step + 1; // Valid response
    else
      iCAN_Step := iCAN_Step - 1 ; // Loop for next
    end_if;
  else
    // Invalid answer. Indicate the error
    iCANStatus:= CANInitError; // Initialization error
    iCAN_Step:=100;
  end_if;
end_if;

```

The project tree on the right shows the following structure:

- CANOpenIO
 - bHeartBeat_HB_Received
 - bHeartBeat_Status
 - byNodeGuardAnswerMask
 - byScratchPad
 - diSDO_CMDS_PTR
 - diSDO_CMDS_QTY
 - iCAN_Init
 - iSCAN_SEQ
 - iSSDO_SSDO_Send
 - r_trig_AI
 - r_trig_DI
 - r_trig_HB
 - T_blink_AI
 - T_blink_DI
- BACnet
- Booleans
- CANbus
 - CANRCVMSG (*Receive...
 - CANSNDMSG (*Send C...
- CANopen
- Clock
- Comparisons
- Conversions
- Counters
- DNP3
- Embedded HMI
- Files
- IEC61850
- Maths
- Miscellaneous



Vizimax Inc.
2284 de la Province
Longueuil, Québec
Canada J4G 1G1

Tel. (450) 679-0003
Fax: (450) 679-9051
Sales@vizimax.com
www.vizimax.com