PCANopen Inspector
User Manual

Manual Revision 1.00

PEAK System Technik

Information in this document is subject to change without notice and does not represent a commitment on the part of the manufacturer. The software described in this document is furnished under license agreement or nondisclosure agreement and may be used or copied in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without prior written permission.

Every effort was made to ensure the accuracy in this manual and to give appropriate credit to persons, companies and trademarks referenced herein.

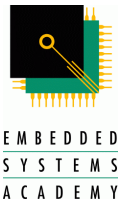
© PEAK-System Technik GmbH and Embedded Systems Academy, Inc. 2006-2009
All Rights Reserved

Microsoft® and Windows™ are trademarks or registered trademarks of Microsoft Corporation.
PC® is a registered trademark of International Business Machines Corporation.

For support contact support@esacademy.com

For the latest news on PCANopen Inspector visit PEAK-System Technik at

www.peak-system.com



PCANopen Inspector was developed for PEAK-System Technik by Embedded Systems Academy. Embedded Systems Academy provides training and consulting services, specializing in CAN, CANopen and Embedded Internetworking. For more information visit

www.esacademy.com

Contents

About This Manual

This manual follows some set conventions with the aim of making it easier to read. The following conventions are used:

0x	Hexadecimal (base 16) values are prefixed with "0x".
italictext	Replace the text with the item it represents
[ ]	Items inside [and] are optional
a | b	a OR b may be used
…	One or more items may go here.

This manual frequently uses CANopen terminology as defined by the CANopen standard DS301 (see www.can-cia.org for more info). Readers that are not yet familiar with all the CANopen terms may want to consider reading a book like www.canopenbook.com or the official standard to update their knowledge on CANopen technology and terminology.

Chapter 1 – Introduction

1.1 About CANopen

CANopen is a higher layer protocol that runs on a CAN network. The CAN specification defines only the physical and data link layers in the ISO/OSI 7-layer Reference Model. This means that only the physical bus and the CAN message format is defined, but not how the CAN messages should be used. CANopen provides an open and standarized but customizable description of how to transfer data of different types between different CAN nodes. This allows off the shelf CANopen compliant nodes to be purchased and plugged into a network with the minimum of effort. It also can be used in place of an in-house proprietory higher layer protocol development.

The development of CANopen is supervised by the CAN in Automation User's Group and is being turned into an international standard. Use of CANopen does not require the payment of any royalties and the specification may be expanded or altered to suit if closed networks are being developed.

Typical applications for CANopen include:

- Commercial Vehicles
- Medical Equipment
- Maritime Electronics
- Building Automation
- Light Rail Systems

1.2 About PCANopen Inspector

PCANopen Inspector is a professional grade CANopen node testing tool. By running a suite of configurable tests, the performance and reliability of a CANopen node may be determined. Multiple aspects of the performance of the node may be measured under different situations with and without simulated background traffic and warnings or errors generated when specific user configurable conditions occur. In addition, automated test modes allow PCANopen Inspector to find the limits of a node's performance, allowing precise characterization of the node.

The CAN in Automation Performance Test (CiA TR-308) is implemented in PCANopen Inspector, providing a standardized means of testing nodes.

In comparison to the official CiA CANopen conformance test, the PCANopen Inspector is a compatibility and performance test. Where the official conformance test checks if all expected responses are given, the Inspector adds detailed timing information. CANopen devices that pass the conformance test may still be inoperable if for example timeouts are used differently. With PCANopen Inspector one can measure the maximum timeouts and delays that occur within a CANopen device.

PCANopen Inspector is a complex application, and it is strongly recommended that this manual be read in it's entirety before using it. Without understanding how the tests operate, the results obtains may be inaccurate or meaningless.

Note that the timestamps used for calculations are accurate to 50 microseconds.

1.3 PCANopen Inspector Features

The following is a list of features in PCANopen Inspector. The list is not exhaustive by any means, but does give a good overview of the abilities of PCANopen Inspector.

- Measures various parameters of a node
  - SDO minimum response time
  - SDO maximum response time
  - Average SDO response time
  - Difference in SDO response times
  - Minimum TPDO response/event time (SYNC, event and turnaround)
  - Maximum TPDO response/event time (SYNC, event and turnaround)
  - Difference in TPDO response/event times (SYNC, event and turnaround)
  - Deviation in TPDO event time
  - Heartbeat minimum event time
  - Heartbeat maximum event time
  - Heartbeat event time deviation
  - Node Guarding minimum response time
  - Node Guarding maximum response time
  - Difference in Node Guarding response time
  - Minimum bootup time
  - Maximum bootup time
  - Average bootup time
- Determines various features of the node
  - Device Type
  - Vendor information
  - Layer Setting Services support
  - PDO linking and mapping functionality
  - Number of supported RPDOs and TPDOs
  - Number of SDO servers and clients
  - NMT Master implemented
  - SDO Manager implemented
  - Configuration Manager implemented
  - Autostart supported
  - SYNC consumer or producer
  - Heartbeat consumer or produder
  - Node Guarding supported
  - Ability to store parameters
  - Emergency consumer or producer
- Ability to configure the node before each test from a Device Configuration File
- Option to describe the node on the network, along with Electronic Datasheets

- Generates customizable compatability, performance and reliability ratings
  - Uses a scripting language for complete customization
- Network Description File, compatible with PCANopen Magic Pro
  - Written in XML based CANopen eXchange Language and describes the network
  - Symbolic information from the files are used where applicable
  - Allows description of customized node Object Dictionaries
- Generate detailed test reports with enough information to reproduce the tests
- Validates previously generated test response as genuine
- Save and load test configurations
- Implements the CAN in Automation Performance Test (CiA TR-308)
- OD Scan Test
  - Scan the Object Dictionary Communication Profile area for supported entries
  - Shows current entry values
  - Shows SDO access time for each entry
- PDO Analysis Test
  - Shows currently implemented PDOs
  - Shows Read/write access information for PDO communication and mapping parameters
  - Shows supported PDO transmission types
  - Shows current communication and mapping values for each PDO
- TPDO Sync Test
  - Measures response time of synchronous TPDOs
  - Configurable sync periods
  - Works with the alarm system and automatic limits testing
  - Optionally generate background traffic according to CiA TR-308
- TPDO Sync with RPDOs Test
  - Measures response time of synchronous TPDOs while receiving RPDOs
  - RPDOs may be configured
  - Configurable sync periods
  - Works with the alarm system and automatic limits testing
  - Optionally generate background traffic according to CiA TR-308
- TPDO Event test
  - Measures the event time of event driven TPDOs
  - Configurable event periods
  - Works with the alarm system and automatic limits testing
  - Optionally generate background traffic according to CiA TR-308
- TPDO Event with RPDOs Test
  - Measures the event time of event driven TPDOs while receiving RPDOs
  - RPDOs may be configured
  - Configurable event periods
  - Works with the alarm system and automatic limits testing
  - Optionally generate background traffic according to CiA TR-308
- PDO Turnaround Test
  - Measures the PDO turnaround time
  - Two phases of RPDOs configurable to ensure stimulation of the node
  - Configurable RPDO transmission periods
  - Works with the alarm system and automatic limits testing
  - Optionally generate background traffic according to CiA TR-308
- Configurable alarm system

    - Generate a warning or error for tests when specific situations occur, such as timings outside of allowed limits
    - Expressions used to describe complex rules for when a alarm occurs
- Powerful automatic limits testing
    - Automatically find the limits of a node for any timing aspect
    - Provides detailed information on the limitations of a node
    - Expressions used to describe complex limits
- Output log showing execution times and progress of tests
- Automatically remembers window positions and settings, hardware and network configurations
- Works with PEAK's PCAN-USB CAN interface
- Can run at the same time, on the same CAN network as other compatible tools, including PCANExplorer, PCANView and PCANopen Magic Pro

1.4 Obtaining Compatible CAN Interfaces

The PEAK PCAN-USB CAN interface is supported. Visit www.peak-system.com to locate the nearest distributor.

Use of other PEAK CAN interfaces is not recommended.

Chapter 2 – Installation and Setup

2.1 Installation

Minimum Requirements

The following is a list of the recommended minimum requirements for running PCANopen Inspector.

- Pentium III 866MHz
- Windows 95
- Pointing device (mouse, trackball, etc.)
- 5Mb of disk space
- 64Mb of RAM

Installation Overview

Installation is a four part process. The following steps must be completed to install PCANopen Inspector:

- Install CAN Interface Driver
- Install PCANopen Inspector
- Install CAN Professional Driver
- Activate PCANopen Inspector

If you have already installed PCANopen Inspector and you are upgrading to a new version, then you normally only need to install PCANopen Inspector itself. I.e. the drivers only need to be uninstalled and reinstalled if they have been updated.

The PCANopen Inspector installation wizard can combine steps two to four into one easy step.

Install CAN Interface Driver

The CAN driver for the interface is supplied with the interface itself on disk or CD. Alternatively, drivers may be downloaded from the PEAK System Technik web site at www.peak-system.com.

Follow the instructions supplied with the interface to install.

Install PCANopen Inspector

To install PCANopen Inspector, simply run the installation executable. An installation wizard will guide you through the steps necessary to install the software. When the activation page is shown, enter the activation code you were supplied with from the place you purchased your copy of PCANopen Inspector. Enter your name and company name. If you did not receive an activation code, contact the place where you purchased the software.

At one point in the installation wizard you will be given the option of also installing the CAN Professional Driver. If you choose to do this (the CAN Professional driver must be installed to use PCANopen Inspector) then a separate installation wizard for the driver will automatically run when PCANopen Inspector has finished installing.

Install CAN Professional Driver

The best way to install the CAN Professional driver is to select the option to install it during installation of PCANopen Inspector.

Follow the prompts in the installation wizard to install the driver.

Note that the CAN Professional driver installation is separate from the PCANopen Inspector installation. Uninstalling PCANopen Inspector will not uninstall the CAN Professional driver. Instead you must seperately uninstall the CAN Professional driver if you wish to remove it.

Activate PCANopen Inspector

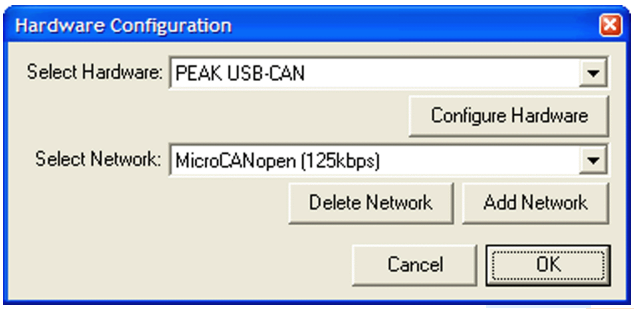
Each copy of PCANopen Inspector needs a valid activation code in order to run. An activation code usually consists of around 14 to 16 letter and numbers. If you did not receive an activation code with your copy of PCANopen Inspector, then contact the place where you purchased your copy and ask for one.

To activate your copy, enter the activation code during the PCANopen Inspector installation when prompted. If you are installing over a previous version then you may find that the code has already been filled in for you.

2.2 Setup

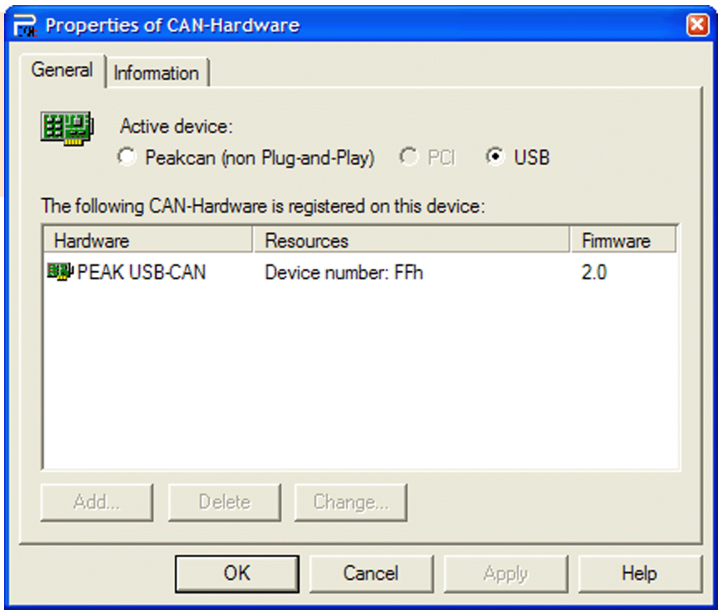
Configuring PCANopen Inspector

Each time you run PCANopen Inspector, the hardware configuration window will open.



1. Hardware Configuration Window

Clicking on "Configure Hardware" will display a window that allows configuration of the CAN interface to use.

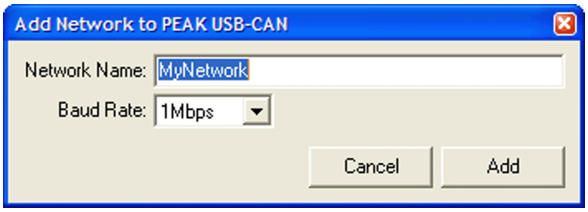


2. CAN Interface Properties Window

Select the category of your interface, for example, PCI for PCI cards, USB for USB interfaces and Peakcan for parallel port (printer port) dongles. Some hardware, such as parallel port dongles must be added by clicking on the "Add" button and entering the settings that apply to your PC.

Make sure your CAN interface is highlighted and click the "OK" button. The Hardware Configuration window should now show your selected interface.

Once a CAN interface has been selected, the bottom part of the Hardware Configuration window will contain a drop-down list of currently available networks. For each network the CAN baud rate is shown. If you do not see a network for the baud rate you wish to use, then click on the "Add Network" button. The Add Network window will be displayed.



3. Add Network Window

Enter a name for the network and select a baud rate. Click on the "Add" button. Your new network should now be displayed in the Network Configuration window.

PCANopen Inspector automatically remembers networks you add. If you wish to delete a network, then simply select it and click on the "Delete Network" button.

Once you have completed the Hardware Configuration, click on the "OK" button to start using PCANopen Inspector.

If you wish to use other PEAK Professional development tools at the same time as PCANopen Inspector, then you will need to select the same network in all the applications.

Configuring the Inspector Support Module

When PCANopen Inspector is running, the Inspector Support Module (ISM) must be connected to the CAN bus. However, the Inspector Support Module may not be configured for the correct baud rate of the CAN bus in use. To configure the ISM follow the steps in the configuration wizard. The wizard may be accessed several ways:

- Click on the Configure Inspector Support Module button on the toolbar
- Choose "Configure Inspector Support Module…" from the Options menu

Ensure that the node ID of the ISM does not conflict with the device under test (DUT).

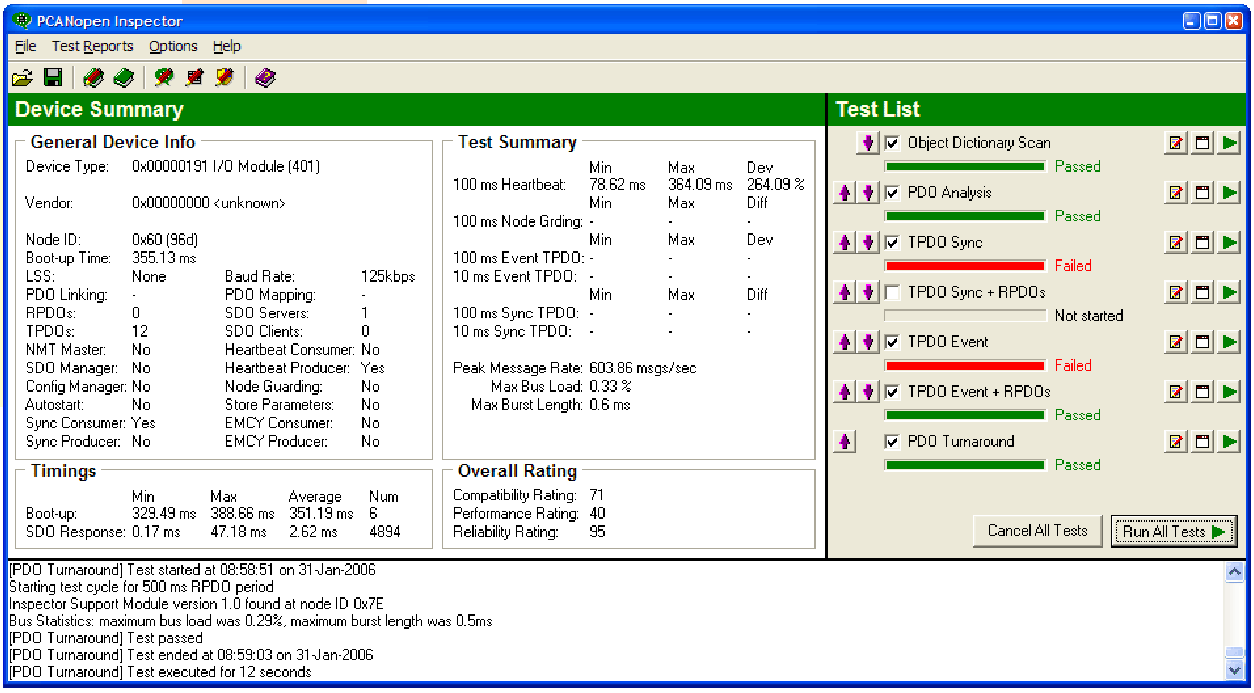
Network Setup

When PCANopen Inspector is running, there must be only two nodes connected to the CAN bus.

- The device under test (DUT)
- The Inspector Support Module (ISM)

Connecting additional nodes may interfere with the operations of the tests and produce incorrect results.

Chapter 3 – User Interface



4. The Main Window

3.1 Quick Tour

The main window is divided into three sections. The top left section is titled "Device Summary" and shows various information obtained during the course of the testing. As the information is discovered it is filled in. The top right section is titled "Test List" and provides a configurable list of the tests, along with various buttons to control the tests. The bottom section is the message log. As the tests run messages appear in the log. The log is included in generated test reports.

When individual tests are run, the device summary is not reset. Instead the information is added to by results from the individual tests. When a group of tests is run, the device summary is reset.

Across the top are the usual menus and a toolbar for quick access to some of the features. Hovering the pointer over a toolbar button will display a hint describing the functionality of the button.

The window may be resized and the dividers dividing the three sections may be dragged around to resize the individual sections.

3.2 Test List

The test list shows all the available tests that can be run along with various buttons that can control the test and some information on the test status.

Run All Tests and Cancel All Tests

Next to each test name is a checkbox. By checking the check box the test will be run when the "Run All Tests" button is clicked. The tests will be run in the same order that they are listed in the window, starting with the test at the top.

To stop the current test or to stop the current group of tests, click on the "Cancel All Tests" button. The current test status will indicate "Cancelled".

Test Order

The purple arrows change the order in which the tests will be run. Tests can be moved up and down the list by clicking on the up and down arrows next to the test that is to be moved.

Test Configuration, Window and Run

To the right of each test name is a set of three buttons. The first button opens the configuration window for that test. The second button opens the results window for the test, showing detailed information. The third button runs that test individually.

In the test windows the progress bar, status, configuration, cancel and run buttons are reproduced, allowing the test to be completely controlled from the test window.

Test Status

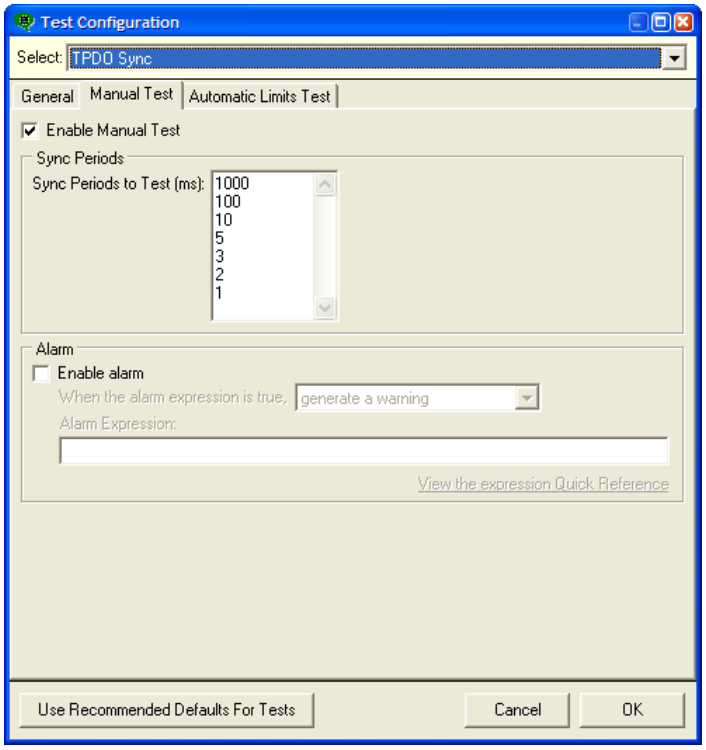
Below each test name is a progress bar and test status. While a test is running the progress bar displays how much of the test has completed. The test status shows such information as "Running", "Warning" and "Failed".

Chapter 4 - Configuration

4.1 Test Configuration

The test configuration window may be accessed by several different methods:

- Clicking on the configuration button of an individual test in the Test List
- Clicking on the "Configure the tests" button on the toolbar
- Choosing "Configure Tests…" from the Options menu
- Clicking on the configuration button in a test window



5. Test Configuration Window

In this window the configuration of any test along with the global test configuration may be accessed. Simply choose the desired test or the global configuration from the drop down list at the top of the window. If the window was opened by clicking on the configuration button of an individual test in the test list, then the window will automatically select the configuration of that test to display.

To use the Embedded Systems Academy, Inc. recommended defaults, simply click on the "Use Recommended Defaults" button at the bottom.

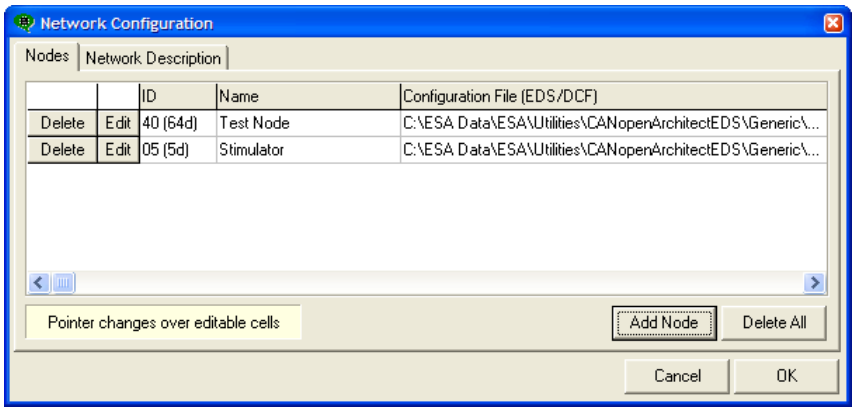
Clicking on "Cancel" will ignore all changes that have been made since opening the window.

Clicking on "OK" will accept the changes made. If there are any errors or problems in the selected configurations then messages will appear.

4.2 Network Configuration

The network configuration window may be accessed by several methods:

- Clicking on the "Configure the network" button on the toolbar
- Choosing "Configure Network…" from the Options menu



6. Network Configuration Window

The network configuration window allows the entry of a description for the whole network. Even though only one node may be connected to the CAN bus at any one time (along with the ISM), it is possible to describe all nodes in one go.

Nodes

To edit a node description, click on the "Nodes" tab and click on the "Edit" button for the desired node. In the node configuration window that appears, enter the name of the node and select the Electronic Datasheet file (EDS) or Device Configuration File (DCF) for the node to specify the Object Dictionary. The Object Dictionary is used in the OD Scan test.

To remove a node description, click on the "Edit" button and then erase the node name and path to the EDS or DCF file in the node configuration window.

Network Description

A network may be described using a CANopen eXchange Language (CXL) file, which is an XML based file format. Currently CXL files are used by PCANopen Magic Pro, PCANopen Magic ProDS and other applications. The file may describe the nodes on the network and their Object Dictionaries, along with custom messages, device types, etc.

By clicking on the "Network Description" tab a CXL file may be specified. Once selected, PCANopen Inspector will use information from the file when needed. Examples of when the information is used are:

- Displaying details of the device type read from the node
- Displaying the vendor information for the node
- Displaying the node name
- Displaying the names of Object Dictionary entries discovered in the node

Information entered under the "Nodes" tab overrides any information in the CXL file.

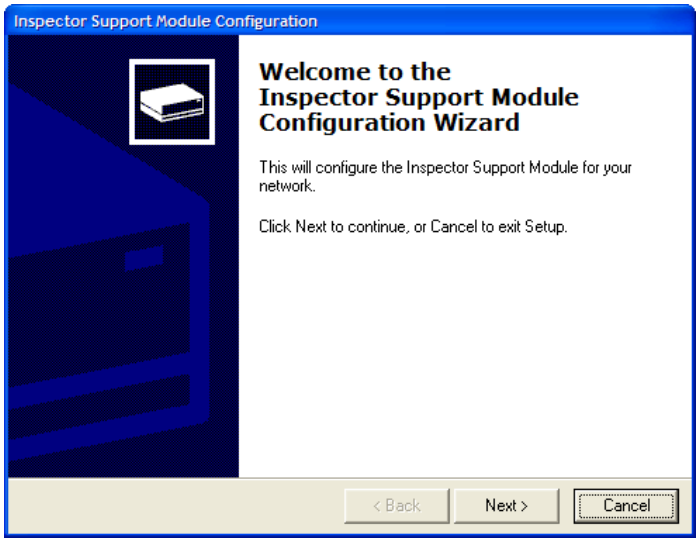
Note that not all information in the CXL file is used by PCANopen Inspector. For a complete description of the file format and the information it contains, refer to appendix A.

4.3 Inspector Support Module Configuration

The Inspector Support Module (ISM) is a support node used by PCANopen Inspector to implement the tests. In order for PCANopen Inspector to operate, the Inspector Support Module must be connected to the CAN bus along with the device under test (DUT).

The ISM remembers the last baud rate and node ID it was configured to, therefore it will be necessary from time to time to change these parameters to suit the network currently being used. In order to change the parameters, the ISM configuration wizard is used. The wizard may be accessed by several different methods:

- Clicking on the "Configure Inspector Support Module" button on the toolbar
- Choosing "Configure Inspector Support Module…" from the Options menu



7. ISM Configuration Window

To configure the ISM simply follow the steps in the wizard.

4.4 Node Configuration

A Device Configuration File (DCF) may be written to the node before the start of every test cycle. This is useful if the node resets into a default state with some functionality such as PDOs disabled.

To specify the DCF to write to the node:

- Click on the Configure the Tests button on the toolbar or choose "Configure Tests…" from the Options menu
- Select "Global Configuration" from the drop-down list at the top of the dialog window
- Check the option "Configure node after each reset"
- Click on "Browse…" and select the DCF file to use
- Click on "OK"

Chapter 5 – Tests

5.1 General Information

Tests will attempt to configure enabled RPDOs and TPDOs to match the test being performed. For example, in the TPDO Sync test, the test will attempt to configure all enabled TPDOs to have a transmission type of 0x01 (transmit on reception of every Sync message).

Tests will not enable any TPDOs or RPDOs that are disabled. If the node has all TPDOs disabled after reset then most of the tests will not work (note that the node will be reset at the start of every test cycle). Instead the node must be configured with a Device Configuration File (DCF) after every reset. PCANopen Inspector can do this automatically and the procedure to enable this feature is described in Chapter 4.

In the following sections each test is described in turn.

5.2 Resetting

To reset all test data, test states, and the log, choose Reset from the Data menu or click on the Reset toolbar button.

5.3 Test Cycles

Many of the tests implement test cycles. A cycle is an individual run of the test with specific settings, and a test may comprise multiple test cycles. The length of time a test cycle runs for is determined by the settings. The relationship is shown in the following table.

Event Time or Sync Period	Test Cycle Run Time
Greater than 5 seconds	Event time or sync period times 6
1 second to 5 seconds	20 seconds
100ms to 1 second	10 seconds
less than 100ms	4 seconds

5.4 TPDO Sync Test

Description

One Sync period is used per test cycle. The Sync message ID is configurable.

At the start of each test cycle the error control of the node is configured. If heartbeats are supported then the node is configured to generate a heartbeat every 100ms. If heartbeats are not supported and node guarding is supported, then the node is configured to use node guarding, with a guard time of 100ms and a life time factor of 5.

The columns in the results table will show either the heartbeat response times and deviation from 100ms, or the node guarding response times and difference.

The TPDO Sync test configures all enabled TPDOs in the node for transmission type 0x01 (transmit on reception of every SYNC) if needed. If PCANopen Inspector fails to configure a TPDO then the test will fail.

The node is placed in Operational mode.

If node guarding is being used then the ISM will start transmitting node guarding requests to the node.

Sync messages will be transmitted by the ISM periodically according to the Sync period for the test cycle. During the test cycle, three SDO read accesses are made from the ISM to index 0x1800, subindex 0x01 every 400ms.

At the end of the test cycle the results are analysed and displayed. The times between the SYNC and the transmission of the TPDOs are measured and displayed.

If node guarding was being used and any responses were not received or the toggle bit was not toggled, then a warning will be generated. If heartbeats were being used and no heartbeats were received then a warning will be generated.

If not all the SDO request and/or responses for access to index 0x1800, subindex 0x01 were seen during the test cycle, then a warning will be generated.

If not all the enabled TPDOs were transmitted after each Sync message or too many were transmitted, then a warning will be generated.

Manual Test

The manual version of the TPDO Sync test may be enabled or disabled as desired. When enabled, it is possible to enter exactly which Sync periods should be tested. They are tested in order, allowing the node to be pushed harder and harder up or even beyond the necessary performance level. Simply enter the desired Sync periods into the box, one per line.

At the end of each test cycle, if the alarm option is enabled then the test results for the test cycle are placed in the variables, and the alarm expression is evaluated. It is possible to generate a warning, generate an error or cancel the test if the expression evaluates to true. For more information on expressions, refer to the expressions chapter.

Automatic Limits Test

The automatic limits version of the TPDO Sync test may be enabled or disabled as desired. When enabled the test automatically generates test cycles to find a limit in the performance of a node. A modified binary search algorithm is used.

The start and end ranges for the Sync period are entered along with an expression defining the limit to find. The test then searches for the shortest Sync period where the expression becomes true inside the range. For more on expressions, refer to the expressions chapter.

The automatic limits test may be enabled at the same time as the manual test. The automatic limits test will start when the manual test has finished.

CiA TR-308 Performance Test

It is possible to run the TPDO Sync test in accordance with the CAN in Automation (CiA) Technical Report 308 (TR-308) Performance Test. The report specifies that timing measurements are taken while specific simulated background traffic appears on the bus.

The background traffic is divided into four types:

- Error Control traffic
- SDO traffic
- PDO traffic
- Sync traffic

Each group may be individually turned on or off by checking the relevant box. It is recommended that TR-308 is read before using this mode and the limitations of the test are fully understood. The limitations are:

- The device under test (DUT) and the ISM must not have one of the following as node IDs: 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- The device under test must not use any IDs that are in the default connection set of nodes 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- **The device under test must not respond to Sync ID 0x080.**

5.5 TPDO Sync + RPDOs Test

Description

One Sync period is used per test cycle. The Sync message ID is configurable.

At the start of each test cycle the error control of the node is configured. If heartbeats are supported then the node is configured to generate a heartbeat every 100ms. If heartbeats are not supported and node guarding is supported, then the node is configured to use node guarding, with a guard time of 100ms and a life time factor of 5.

The columns in the results table will show either the heartbeat response times and deviation from 100ms, or the node guarding response times and difference.

The TPDO Sync test configures all enabled TPDOs and RPDOs in the node for transmission type 0x01 (transmit on reception of every SYNC/actuate on reception of every SYNC) if needed. If PCANopen Inspector fails to configure a TPDO or RPDO then the test will fail.

The node is placed in Operational mode.

If node guarding is being used then the ISM will start transmitting node guarding requests to the node.

Sync messages will be transmitted by the ISM periodically according to the Sync period for the test cycle. After each Sync message RPDOs will be transmitted to the node with configurable data. During the test cycle, three SDO read accesses are made from the ISM to index 0x1800, subindex 0x01 every 400ms.

At the end of the test cycle the results are analysed and displayed. The times between the SYNC and the transmission of the TPDOs are measured and displayed.

If node guarding was being used and any responses were not received or the toggle bit was not toggled, then a warning will be generated. If heartbeats were being used and no heartbeats were received then a warning will be generated.

If not all the SDO request and/or responses for access to index 0x1800, subindex 0x01 were seen during the test cycle, then a warning will be generated.

If not all the enabled TPDOs were transmitted after each Sync message or too many were transmitted, then a warning will be generated.

Note that the first set of RPDO data will be transmitted with an ID of the first enabled RPDO in the node, the second set transmitted with an ID of the second enabled RPDO, etc.

Manual Test

The manual version of the TPDO Sync test may be enabled or disabled as desired. When enabled, it is possible to enter exactly which Sync periods should be tested. They are tested in order, allowing the node to be pushed harder and harder up or even beyond the necessary performance level. Simply enter the desired Sync periods into the box, one per line.

At the end of each test cycle, if the alarm option is enabled then the test results for the test cycle are placed in the variables, and the alarm expression is evaluated. It is possible to generate a warning, generate an error or cancel the test if the expression evaluates to true. For more information on expressions, refer to the expressions chapter.

Automatic Limits Test

The automatic limits version of the TPDO Sync test may be enabled or disabled as desired. When enabled the test automatically generates test cycles to find a limit in the performance of a node. A modified binary search algorithm is used.

The start and end ranges for the Sync period are entered along with an expression defining the limit to find. The test then searches for the shortest Sync period where the expression becomes true inside the range. For more on expressions, refer to the expressions chapter.

The automatic limits test may be enabled at the same time as the manual test. The automatic limits test we start when the manual test has finished.

CiA TR-308 Performance Test

It is possible to run the TPDO Sync test in accordance with the CAN in Automation (CiA) Technical Report 308 (TR-308) Performance Test. The report specifies that timing measurements are taken while specific simulated background traffic appears on the bus.

The background traffic is divided into four types:

- Error Control traffic
- SDO traffic
- PDO traffic
- Sync traffic

Each group may be individually turned on or off by checking the relevant box. It is recommended that TR-308 is read before using this mode and the limitations of the test are fully understood. The limitations are:

- The device under test (NUT) and ISM must not have one of the following as node IDs: 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- The device under test must not use any IDs that are in the default connection set of nodes 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- **The device under test must not respond to Sync ID 0x080.**

5.6 TPDO Event Test

Description

One event time is used per test cycle.

At the start of each test cycle the error control of the node is configured. If heartbeats are supported then the node is configured to generate a heartbeat every 100ms. If heartbeats are not supported and node guarding is supported, then the node is configured to use node guarding, with a guard time of 100ms and a life time factor of 5.

The columns in the results table will show either the heartbeat response times and deviation from 100ms, or the node guarding response times and difference.

The TPDO Event test optionally configures all enabled TPDOs in the node for transmission type 0xFE (manufacturer defined) or 0xFF (device profile defined, asynchronous change of state transmission) and the event time for the test cycle if needed. If PCANopen Inspector fails to configure a TPDO then the test will fail.

The node is placed in Operational mode.

If node guarding is being used then the ISM will start transmitting node guarding requests to the node.

During the test cycle, three SDO read accesses are made from the ISM to index 0x1800, subindex 0x01 every 400ms.

At the end of the test cycle the results are analysed and displayed. The times between the transmission of the TPDOs are displayed.

If node guarding was being used and any responses were not received or the toggle bit was not toggled, then a warning will be generated. If heartbeats were being used and no heartbeats were received then a warning will be generated.

If not all the SDO request and/or responses for access to index 0x1800, subindex 0x01 were seen during the test cycle, then a warning will be generated.

If not all the enabled TPDOs were transmitted during each event period or too many were transmitted, then a warning will be generated.

Manual Test

The manual version of the TPDO Event test may be enabled or disabled as desired. When enabled, it is possible to enter exactly which event times should be tested. They are tested in order, allowing the node to be pushed harder and harder up or even beyond the necessary performance level. Simply enter the desired event times into the box, one per line.

At the end of each test cycle, if the alarm option is enabled then the test results for the test cycle are placed in the variables, and the alarm expression is evaluated. It is possible to generate a warning, generate an error or cancel the test if the expression evaluates to true. For more information on expressions, refer to the expressions chapter.

Automatic Limits Test

The automatic limits version of the TPDO Event test may be enabled or disabled as desired. When enabled the test automatically generates test cycles to find a limit in the performance of a node. A modified binary search algorithm is used.

The start and end ranges for the event times are entered along with an expression defining the limit to find. The test then searches for the shortest event time where the expression becomes true inside the range. For more on expressions, refer to the expressions chapter.

The automatic limits test may be enabled at the same time as the manual test. The automatic limits test we start when the manual test has finished.

CiA TR-308 Performance Test

It is possible to run the TPDO Event test in accordance with the CAN in Automation (CiA) Technical Report 308 (TR-308) Performance Test. The report specifies that timing measurements are taken while specific simulated background traffic appears on the bus.

The background traffic is divided into four types:

- Error Control traffic
- SDO traffic
- PDO traffic
- Sync traffic

Each group may be individually turned on or off by checking the relevant box. It is recommended that TR-308 is read before using this mode and the limitations of the test are fully understood. The limitations are:

- The device under test (DUT) and ISM must not have one of the following as node IDs: 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- The device under test must not use any IDs that are in the default connection set of nodes 3, 5, 8, 12, 13, 32, 35, 43, 57, 90

5.7 TPDO Event + RPDOs Test

Description

One event time is used per test cycle.

At the start of each test cycle the error control of the node is configured. If heartbeats are supported then the node is configured to generate a heartbeat every 100ms. If heartbeats are not supported and node guarding is supported, then the node is configured to use node guarding, with a guard time of 100ms and a life time factor of 5.

The columns in the results table will show either the heartbeat response times and deviation from 100ms, or the node guarding response times and difference.

The TPDO Event test optionally configures all enabled TPDOs and RPDOs in the node for transmission type 0xFE (manufacturer defined) or 0xFF (device profile defined, asynchronous change of state transmission) and the event time for the test cycle (TPDOs only) if needed. If PCANopen Inspector fails to configure a TPDO or RPDO then the test will fail.

The node is placed in Operational mode.

If node guarding is being used then the ISM will start transmitting node guarding requests to the node.

During the test cycle, three SDO read accesses are made from the ISM to index 0x1800, subindex 0x01 every 400ms. The RPDOs are transmitted periodically with a period matching the event time.

At the end of the test cycle the results are analysed and displayed. The times between the transmission of the TPDOs are displayed.

If node guarding was being used and any responses were not received or the toggle bit was not toggled, then a warning will be generated. If heartbeats were being used and no heartbeats were received then a warning will be generated.

If not all the SDO request and/or responses for access to index 0x1800, subindex 0x01 were seen during the test cycle, then a warning will be generated.

If not all the enabled TPDOs were transmitted during each event period or too many were transmitted, then a warning will be generated.

Manual Test

The manual version of the TPDO Event test may be enabled or disabled as desired. When enabled, it is possible to enter exactly which event times should be tested. They are tested in order, allowing the node to be pushed harder and harder up or even beyond the necessary performance level. Simply enter the desired event times into the box, one per line.

At the end of each test cycle, if the alarm option is enabled then the test results for the test cycle are placed in the variables, and the alarm expression is evaluated. It is possible to generate a warning, generate an error or cancel the test if the expression evaluates to true. For more information on expressions, refer to the expressions chapter.

Automatic Limits Test

The automatic limits version of the TPDO Event test may be enabled or disabled as desired. When enabled the test automatically generates test cycles to find a limit in the performance of a node. A modified binary search algorithm is used.

The start and end ranges for the event times are entered along with an expression defining the limit to find. The test then searches for the shortest event time where the expression becomes true inside the range. For more on expressions, refer to the expressions chapter.

The automatic limits test may be enabled at the same time as the manual test. The automatic limits test we start when the manual test has finished.

CiA TR-308 Performance Test

It is possible to run the TPDO Event test in accordance with the CAN in Automation (CiA) Technical Report 308 (TR-308) Performance Test. The report specifies that timing measurements are taken while specific simulated background traffic appears on the bus.

The background traffic is divided into four types:

- Error Control traffic
- SDO traffic
- PDO traffic
- Sync traffic

Each group may be individually turned on or off by checking the relevant box. It is recommended that TR-308 is read before using this mode and the limitations of the test are fully understood. The limitations are:

- The device under test (DUT) and ISM must not have one of the following as node IDs: 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- The device under test must not use any IDs that are in the default connection set of nodes 3, 5, 8, 12, 13, 32, 35, 43, 57, 90

5.8 PDO Turnaround Test

Description

The PDO Turnaround test assumes that the node is configured such that reception of an RPDO will result in transmission of one or more TPDOs. The time between the reception of an RPDO and the transmission of the corresponding TPDOs is the TPDO response time.

The node must be configured either by physically connecting inputs to outputs or implementing a logical connection between RPDOs and TPDOs in the application layer.

One RPDO transmission period is used per test cycle. Two phases of RPDOs can be configured. By placing different data in each phase, data received by the node can be continually changed, causing stimulation of the node. Each test cycle is divided up into sub-cycles. There is one sub-cycle for each RPDO supported by the node.

In a sub-cycle the RPDO is transmitted periodically. First the RPDO from phase 1 is transmitted followed by the RPDO from phase 2. The RPDO from phase 1 is then transmitted again and the process repeats until the end of the sub-cycle.

This results in the following pattern:

RPDO1 – phase 1
RPDO1 – phase 2
… repeated
RPDO 2 – phase 1
RPDO 2 – phase 2
… repeated
RPDO 3 – phase 1
RPDO 3 – phase 2
… repeated

The length of a sub-cycle is the length of the test cycle divided by the number of RPDOs supported by the node.

At the start of each test cycle the error control of the node is configured. If heartbeats are supported then the node is configured to generate a heartbeat every 100ms. If heartbeats are not supported and node guarding is supported, then the node is configured to use node guarding, with a guard time of 100ms and a life time factor of 5.

The columns in the results table will show either the heartbeat response times and deviation from 100ms, or the node guarding response times and difference.

The PDO Turnaround test optionally configures all enabled TPDOs and RPDOs in the node for transmission type 0xFE (manufacturer defined) or 0xFF (device profile defined, asynchronous change of state transmission). If PCANopen Inspector fails to configure a TPDO or RPDO then the test will fail.

The node is placed in Operational mode.

If node guarding is being used then the ISM will start transmitting node guarding requests to the node.

During the test cycle, three SDO read accesses are made from the ISM to index 0x1800, subindex 0x01 every 400ms. The RPDOs are transmitted periodically with a period matching the event time.

At the end of the test cycle the results are analysed and displayed. The times between the transmission of an RPDO and the transmission of TPDOs is displayed.

If node guarding was being used and any responses were not received or the toggle bit was not toggled, then a warning will be generated. If heartbeats were being used and no heartbeats were received then a warning will be generated.

If not all the SDO request and/or responses for access to index 0x1800, subindex 0x01 were seen during the test cycle, then a warning will be generated.

If no TPDOs were transmitted during each event period then a warning will be generated.

Manual Test

The manual version of the PDO Turnaround test may be enabled or disabled as desired. When enabled, it is possible to enter exactly which RPDO transmission periods should be tested. They are tested in order, allowing the node to be pushed harder and harder up or even beyond the necessary performance level. Simply enter the desired event times into the box, one per line.

At the end of each test cycle, if the alarm option is enabled then the test results for the test cycle are placed in the variables, and the alarm expression is evaluated. It is possible to generate a warning, generate an error or cancel the test if the expression evaluates to true. For more information on expressions, refer to the expressions chapter.

In order for PCANopen Inspector to know which TPDOs should be transmitted for a specific RPDO, the first test cycle in the test is used to record what was returned by the node for each RPDO. For each subsequent test cycle the number of TPDOs returned is compared to previous test cycles for any difference. Any differences will generate warning. The RPDO transmission period for the first test cycle should, therefore, be long enough for all TPDOs to be transmitted within the RPDO transmission period.

Automatic Limits Test

The automatic limits version of the PDO Turnaround test may be enabled or disabled as desired. When enabled the test automatically generates test cycles to find a limit in the performance of a node. A modified binary search algorithm is used.

The start and end ranges for the RPDO transmission periods are entered along with an expression defining the limit to find. The test then searches for the shortest RPDO transmission period where the expression becomes true inside the range. For more on expressions, refer to the expressions chapter.

The automatic limits test may be enabled at the same time as the manual test. The automatic limits test we start when the manual test has finished.

In order for PCANopen Inspector to know which TPDOs should be transmitted for a specific RPDO, the first test cycle in the test is used to record what was returned by the node for each RPDO. For each subsequent test cycle the number of TPDOs returned is compared to previous test cycles for any difference. Any differences will generate warning. The RPDO transmission period for the first test cycle should, therefore, be long enough for all TPDOs to be transmitted within the RPDO transmission period.

CiA TR-308 Performance Test

It is possible to run the TPDO Event test in accordance with the CAN in Automation (CiA) Technical Report 308 (TR-308) Performance Test. The report specifies that timing measurements are taken while specific simulated background traffic appears on the bus.

The background traffic is divided into four types:

- Error Control traffic
- SDO traffic
- PDO traffic
- Sync traffic

Each group may be individually turned on or off by checking the relevant box. It is recommended that TR-308 is read before using this mode and the limitations of the test are fully understood. The limitations are:

- The device under test (DUT) and ISM must not have one of the following as node IDs: 3, 5, 8, 12, 13, 32, 35, 43, 57, 90
- The device under test must not use any IDs that are in the default connection set of nodes 3, 5, 8, 12, 13, 32, 35, 43, 57, 90

5.9 PDO Analysis Test

Description

The PDO Analysis test reads in the current configuration of the TPDOs and RPDOs in the node and displays the result in a table. A test is performed to determine supported transmission types.

The access (read only, read/write) is determined for the communication parameters and mapping and displayed in the table.

- (RO) = read only
- (RW) = read/write
- (X) = error reading entry

5.10 Object Dictionary Scan Test

Description

The Object Dictionary Scan test scans the Communication Profile area of the Object Dictionary. The results are shown in the table along with the current data in the entry in ASCII, hexadecimal and decimal and the SDO response time for the entry.

Names for the Object Dictionary entries are either obtained from an internal database, or read from the Network Description File.

Chapter 6 – Expressions

6.1 Overview

Expressions are a way of representing mathematical formulas. PCANopen Inspector uses expressions in several places to allow a high degree of configurability using a standard, compact notation.

Expressions in PCANopen Inspector have the same syntax as C programming language expressions, and implement a subset of the functionality present in C.

Examples:

```
msgrate > 52.4
(tpdodiff <= 41) || (tpdomin > 6.2)
(syncdev > 101.2 && (msgrate != 45)) || syncdev > 106.3
```

In many cases an expression is evaluated and the result is checked to see whether it is true or false. For example in the alarms feature an expression is evaluated after each test cycle. If the expression is true then the alarm is signalled and the action chosen (i.e. generate warning) occurs.

If the result of an expression is zero, then the result is regarded as false.
If the result of an expression is non-zero (1, 5.6, -3, etc.) then the result is regarded as true.

6.2 Components

An expression is made up of the following components. Not all components must be present to create a valid expression.

- Variables
- Constants
- Operators

Each component is describe in turn in the following sections.

Variables

PCANopen Inspector defines a collection of variables. The values of these variables are updated at the end of each test cycle before the expression is evaluated. The value of the variables are shown in the test window for that cycle. Not all tests define values for all the variables, and sometimes the value of a variable will be undefined.

Variable names are case sensitive.

Variable: tpdomin
Type: float
Description: The minimum TPDO event or response time in ms

Variable: tpdomax
Type: float
Description: The maximum TPDO event or response time in ms

Variable: tpdodiff
Type: float
Description: The difference in TPDO event or response time in ms

Variable: tpdodev
Type: float
Description: The deviation in TPDO event or response times in %

Variable: tpdoave
Type: float
Description: The average TPDO event or response times in ms

Variable: ecmin
Type: float
Description: The minimum Node Guarding response time or Heartbeat event time in ms

Variable: ecmax
Type: float
Description: The maximum Node Guarding response time or Heartbeat event time in ms

Variable: ecdiff
Type: float
Description: The different in Node Guarding response times or Heartbeat event times in ms

Variable: ecdev
Type: float
Description: The deviation in Node Guarding response times or Heartbeat event times in %

Variable: sdomin
Type: float
Description: The minimum SDO response time in ms

Variable: sdomax
Type: float
Description: The maximum SDO response time in ms

Variable: syncmin
Type: float
Description: The minimum SYNC event time in ms

Variable: syncmax
Type: float
Descripton: The maximum SYNC event time in ms

Variable: syncdev
Type: float
Description: The deviation in SYNC event times in %

Constants

Constants are values written directly into expressions and are most commonly used in a comparison with a variable. Constants may be in decimal (base 10), octal (base 8) or hexadecimal (base 16). Decimal values may be integers or floating point and may be negative or positive.

Octal numbers are prefixed with a zero and may only contain the digits zero to seven.

Hexadecimal numbers are prefixed with "0x" or "0X" and may only contain the characters zero to nine and A to F.

Three suffixes may be used with constants to force them to a specific type.

U unsigned (no negative values)
L long (32-bit)
F float (32-bit)

By default any integers are of type int (16-bit) and any real numbers are of type double (64-bit).

Operators

Operators either perform a comparison between two values (either or both of which can be a variable or constant) or they take one or two values and produce a result.

Unary Operators

These operators work on one value that is to the right of the operator. Standard C operator precedence and associativity is used. If in doubt, use parenthesis.

! NOT
~ One's complement

Binary Operators

These operators work on the values either side of the operator. Standard C operator precedence and associativity is used. If in doubt, use parenthesis.
Comparison operations (i.e. <, >) generate a result of 1 for true and 0 for false.

* Multiplication
/ Division
% Modulus

+	Addition
-	Subtraction
<<	Shift left
>>	Shift right
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
||	Logical OR

Special Characters

(	Open parenthesis
)	Close parenthesis

6.3 Rules

Integral conversion, integer promotion, operator associativity and operator precedence are implemented in the expression system. These are standard C programming language ideas, and are implemented in the same way as they are in C.

Integral Conversion

When a binary operator needs to operate on two values of different types, one of the types must be converted to match the other. In general the smallest common type that can hold both values in use is used.

Integer Promotion

If an int type can hold the value of a char typed value, then the char typed value is converted to an int.

Operator Precedence

Consider the following expression:

```
4 * 2 + 5
```

What is the value of this expression? Is it 13 or 28? Operator precedence is a method of ensuring that the result is always known by giving a priority to operators. In the C programming language (and therefore this expression system), the multiplication operator

has a higher priority (or precedence) than the addition operator. The result therefore is always 13.

Using parenthesis ("(" and ")") the order the operators are processed can be forced. To give a result of 28 for the previous example, it must be written as:

```
4 * (2 + 5)
```

Anything inside parenthesis is processed first. Parenthesis can be nested, for example:

```
4 + (4 * (2 + 5))
```

The expression will evaluate to 32.

There is no penalty for using parenthesis, so if in doubt about the order of operator precedence, use parenthesis.

Operator Associativity

Associativity defines the ordering that the expression is processed in. Consider the following expression

```
4 + 2 – 1
```

Because the associativity of the addition and subtraction operators is left to right, first 4 and 2 are added to give 6, then 1 is subtracted from 6 to give the result.

Unary operators have right associativity, and always process what is to the right of them.

To override associativity use parenthesis as described in the previous section.

6.4 Limitations

The expressions are limited to a total of 200 operators, variable names and parenthesis.
The expressions are limited to a total of 300 characters.

6.5 Reference

For more information on C expressions, especially integral conversion and integer promotion see "The C Programming Language", 2nd Ed., Kernighan and Ritche, Prentice Hall.

Chapter 7 – Overall Ratings

7.1 Overview

The overall ratings sum up the performance of a node into a collection of results. These ratings may be used to compare nodes and set a criteria for whether a node should be used or not.

The ratings are:

Compatability:	How compatible is the node in it's implementation of the CANopen protocol? Result is a value in the range 0 – 100
Performance:	How closely does the node match the desired performance requirements? Result is a value in the range 0 –100
Reliability:	Were all necessary tests passed? Result is a value in the range 0 – 100

Because the calculation of these parameters is highly dependant on the specific node and requirements that are to be met, it is not possible to provide one generic method of calculating these values. Therefore after each test or all tests have finished executing a script is run to calculate the values.

The ratings script is written in a language called Python and can be accessed in the configuration window, under "Global Configuration".

7.2 Script Structure

The script must import the "Inspector" module, gain access to the device summary and fill in the "compatabilityrating", "reliabilityrating" and "performancerating" values. The following illustrates a skeleton script.

```
import Inspector
ds = Inspector.DeviceSummary()
ds.compatibilityrating = 0
ds.reliabilityrating = 0
ds.performancerating = 0
print 'Ratings generation complete'
```

Any print commands appear in the log, allowing additional data (for example the results of calculations) to be performed at the end of testing and included in the test report.

7.3 Example Script

The following is an example script that can be used as the basis for a more complex customized version.

```
# this script calculates the compatibility,
# performance and reliability ratings

import Inspector

ds = Inspector.DeviceSummary()

# calculate compatibility rating - look for
# minimum od entry
if Inspector.ODScan().status != Inspector.NOTSTARTED:
  if Inspector.ODScan().status == Inspector.PASSED and \
     Inspector.ODScanEntry(0x1000, 0x00).found and \
     Inspector.ODScanEntry(0x1001, 0x00).found and \
     Inspector.ODScanEntry(0x1018, 0x00).found and \
     Inspector.ODScanEntry(0x1018, 0x01).found:
    ds.compatibilityrating = 100
    print "Compatibility check passed (100%)"
  else:
    ds.compatibilityrating = 0
    print "Compatibility check failed (0%)"

# calculate reliability rating - look for all
# enabled tests were passed
reliability_max = 0
if Inspector.ODScan().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.PDOAnalysis().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.TPDOSync().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.TPDOSyncRPDOs().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.TPDOEvent().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.TPDOEventRPDOs().status != Inspector.NOTSTARTED:
  reliability_max += 10
if Inspector.PDOTurnaround().status != Inspector.NOTSTARTED:
  reliability_max += 10

reliability_act = 0
if reliability_max > 0:
  if Inspector.ODScan().status == Inspector.PASSED:
    reliability_act += 10
  if Inspector.PDOAnalysis().status == Inspector.PASSED:
```

```
    reliability_act += 10
  if Inspector.TPDOSync().status == Inspector.PASSED:
    reliability_act += 10
  if Inspector.TPDOSyncRPDOs().status == Inspector.PASSED:
    reliability_act += 10
  if Inspector.TPDOEvent().status == Inspector.PASSED:
    reliability_act += 10
  if Inspector.TPDOEventRPDOs().status == Inspector.PASSED:
    reliability_act += 10
  if Inspector.PDOTurnaround().status == Inspector.PASSED:
    reliability_act += 10

  ds.reliabilityrating = 100 * reliability_act / reliability_max
  print "Reliability rating at %d%%" % ds.reliabilityrating

# calculate performance rating
performance_max = 0
performance_act = 0
if ds.heartbeat100ms_hasdata:
  performance_max += 10
  if ds.heartbeat100ms_dev < 20:
    performance_act += 10
if ds.nodeguarding100ms_hasdata:
  performance_max += 10
  if ds.nodeguarding100ms_diff < 40:
    performance_act += 10
if ds.eventtpdo100ms_hasdata:
  performance_max += 10
  if ds.eventtpdo100ms_dev < 40:
    performance_act += 10
if ds.eventtpdo10ms_hasdata:
  performance_max += 10
  if ds.eventtpdo10ms_dev < 20:
    performance_act += 10
if ds.synctpdo100ms_hasdata:
  performance_max += 10
  if ds.synctpdo100ms_diff < 40:
    performance_act += 10
if ds.synctpdo10ms_hasdata:
  performance_max += 10
  if ds.synctpdo10ms_diff < 20:
    performance_act += 10
if ds.bootup_hasdata:
  performance_max += 10
  if ds.bootup_max < 50:
    performance_act += 10
if ds.sdoresponse_hasdata:
  performance_max += 10
  if ds.sdoresponse_max < 10:
```

```
    performance_act += 10

if performance_max > 0:
  ds.performancerating = 100 * performance_act / performance_max
  print "Performance rating at %d%%" % ds.performancerating

print 'Ratings generation complete'
```

7.4 Python Reference

This quick reference describes the custom objects that are defined in Python to access information in the application. For a language reference see the free ebook "Dive into Python" at **www.diveintopython.org**.

Abbreviations used:

RO = read only
RW = read/write
WO = write only

Inspector Module

All Inspector functionality is contained in the Inspector module. Import it before using with:

```
import Inspector
```

The Inspector module defines constants and types that are described below.

Inspector Constants

The Inspector constants for test status are:

NOTSTARTED
RUNNING
PASSED
WARNING
FAILED
CANCELLED

The Inspector constants for Object Dictionary entry accessiblity are:

UNKNOWN
READONLY
READWRITE
UNTESTED

The constants may be accessed as follows:

```
Inspector.PASSED
Inspector.READWRITE
```

Inspector Types

There are one or two types for each test available in the application, and a type for the device summary. The device summary type may be used to read any current device summary data, or set many of the values. The test types can access the test status, whether the test is enabled or not and the number of test cycles (if applicable).
In addition, for tests that have multiple test cycles, there is a test cycle type, which allows access to the results of individual test cycles.

The following sections describe these types.

Device Summary

The device summary type allows read/write access to the device summary. For example:

```
ds = Inspector.DeviceSummary()
ds.reliabilityrating = 5
```

or:

```
Inspector.DeviceSummary().reliabilityrating = 5
```

to set the reliability rating. Boolean values (shown in the main window as yes or no) are treated in Python as integers with the values 0 (for no) and 1 (for yes).

List of properties defined:

devicetype
Device type. RW

vendor
Vendor ID. RW

nodeid
Node ID. RW

baudrate
Baud rate in kbps. RW

bootuptime
Bootup time in ms. RW

nmtmaster
NMT master. 0 (no) or 1 (yes). RW

sdomanager

SDO manager. 0 (no) or 1 (yes). RW

configmanager
Configuration manager. 0 (no) or 1 (yes). RW

autostart
Auto start. 0 (no) or 1 (yes). RW

lss
Layer setting services. 0 (no) or 1 (yes). RW

rpdonum
Number of RPDOs. RW

tpdonum
Number of TPDOs. RW

pdolinking
PDO linking. 0 (no) or 1 (yes). RW

pdomapping
PDO mapping. 0 (no) or 1 (yes). RW

syncconsumer
Sync consumer. 0 (no) or 1 (yes). RW

syncproducer
Sync producer. 0 (no) or 1 (yes). RW

heartbeatconsumer
Heartbeat consumer. 0 (no) or 1 (yes). RW

heartbeatproducer
Heartbeat producer. 0 (no) or 1 (yes). RW

nodeguarding
Node guarding. 0 (no) or 1 (yes). RW

storeparameters
Store parameters. 0 (no) or 1 (yes). RW

emcyconsumer
Emergency consumer. 0 (no) or 1 (yes). RW

emcyproducer
Emergency producer. 0 (no) or 1 (yes). RW

sdoclientnum
Number of SDO clients. RW

sdoservernum
Number of SDO servers. RW

bootup_min
Minimum bootup time in ms. RO

bootup_max
Maximum bootup time in ms. RO

bootup_average
Average bootup time in ms. RO

bootup_numtries
Number of bootups requested. RO

bootup_total
Total number of bootups. RO

bootup_hasdata
Indicates if any bootup data has been recorded. 0 (no) or 1 (yes). RO

sdoresponse_min
Minimum SDO response time in ms. RO

sdoresponse_max
Maximum SDO response time in ms. RO

sdoresponse_average
Average SDO response time in ms. RO

sdoresponse_numtries
Number of SDO requests sent. RO

sdoresponse_total
Total SDO responses. RO

sdoresponse_hasdata
Indicates if any sdo response data has been recorded. 0 (no) or 1 (yes). RO

heartbeat100ms_min
Minimum 100ms heartbeat time in ms. RO

heartbeat100ms_max
Maximum 100ms heartbeat time in ms. RO

heartbeat100ms_dev
100ms heartbeat deviation. RO

heartbeat100ms_hasdata
Indicates if any 100ms heartbeat data has been recorded. 0 (no) or 1 (yes). RO

nodeguarding100ms_min
Minimum 100ms node guarding time in ms. RO

nodeguarding100ms_max
Maximum 100ms node guarding time in ms. RO

nodeguarding100ms_diff
100ms node guarding difference. RO

nodeguarding100ms_hasdata
Indicates if any 100ms node guarding data has been recorded. 0 (no) or 1 (yes). RO

eventtpdo100ms_min
Minimum 100ms TPDO event time in ms. RO

eventtpdo100ms_max
Maximum 100ms TPDO event time in ms. RO

eventtpdo100ms_dev
100ms TPDO event time deviation. RO

eventtpdo100ms_ave
100ms TPDO event time average.RO

eventtpdo100ms_hasdata
Indicates if any 100ms TPDO event time data has been recorded. 0 (no) or 1 (yes). RO

eventtpdo10ms_min
Minimum 10ms TPDO event time in ms. RO

eventtpdo10ms_max
Maximum 10ms TPDO event time in ms. RO

eventtpdo10ms_dev
10ms TPDO event time deviation. RO

eventtpdo10ms_ave
10ms TPDO event time average. RO

eventtpdo10ms_hasdata
Indicates if any 10ms TPDO event time data has been recorded. 0 (no) or 1 (yes). RO

synctpdo100ms_min
Minimum 100ms TPDO sync time in ms. RO

synctpdo100ms_max
Maximum 100ms TPDO sync time in ms. RO

synctpdo100ms_diff
100ms TPDO sync time difference. RO

synctpdo100ms_ave
100ms TPDO sync time average. RO

synctpdo100ms_hasdata
Indicates if any 100ms TPDO sync time data has been recorded. 0 (no) or 1 (yes). RO

synctpdo10ms_min
Minimum 10ms TPDO sync time in ms. RO

synctpdo10ms_max
Maximum 10ms TPDO sync time in ms. RO

synctpdo10ms_diff
10ms TPDO sync time difference. RO

synctpdo10ms_ave
10ms TPDO sync time average. RO

synctpdo10ms_hasdata
Indicates if any 10ms TPDO sync time data has been recorded. 0 (no) or 1 (yes). RO

maxbusload
Maximum bus load. RW

maxburstlength
Maximum burst length. RW

peakmessagerate
Peak message rate. RW

compatibilityrating
Compatibility rating. RW

performancerating
performance rating. RW

reliabilityrating
Reliability rating. RW

OD Scan Test

The ODScan type allows access to the results of the test and status:

```
test = Inspector.ODScan()
s = test.status
```

List of properties defined:

status
Test status. One of the values described in the Inspector Constants section. RO

enabled
0 (for disabled) or 1 (for enabled). RO

The ODScanEntry type allows access to the details of a specific OD entry. Passed is the entry index and subindex:

```
odentry = Inspector.ODScanEntry(0x1000, 0x00)
f = odentry.found
```

List of properties defined:

index
OD entry index. RO

subindex
OD entry subindex. RO

found
0 (not found) or 1 (found). RO

size
Size of OD entry data. RO

accesstime
Access time in ms. RO

value
Value of entry if four bytes or less. RO

PDO Analysis Test

The PDOAnalysis type allows access to the results of the test and status:

```
test = Inspector.PDOAnalysis()
s = test.status
```

List of properties defined:

status
Test status. One of the values described in the Inspector Constants section. RO

enabled
0 (for disabled) or 1 (for enabled). RO

The PDOAnalysisPDO type allows access to the details of a specific PDO. Passed is the index of the PDO communication parameters:

```
pdo = Inspector.PDOAnalysisPDO(0x1800)
i = pdo.index
```

List of properties defined:

index
PDO communication parameters OD entry index. RO

found
0 (not found) or 1 (found). RO

transtype
Transmission type. RO

inhibittime
Inhibit time. RO

eventtime
Event time. RO

mappingnum
Number of mapping entries. RO

cobidaccess
Access type for cobid. One of the values defined in the Inspector Constants section. RO

transtypeaccess
Access type for transmission type. One of the values defined in the Inspector Constants section. RO

inhibittimeaccess
Access type for inhibit time. One of the values defined in the Inspector Constants section. RO

eventtimeaccess
Access type for event time. One of the values defined in the Inspector Constants section. RO

mappingnumaccess
Access type for mapping number. One of the values defined in the Inspector Constants section. RO

mapping(n)
Returns the mapping value for mapping number n. n is 1-indexed. The value returned has the format (all values are big-endian):

<index><subindex><length>

TPDO Sync, TPDO Sync + RPDOs, TPDO Event, TPDO Event + RPDOs and PDO Turnaround

All of these tests have largely the same properties defined in their types. Some tests may not implement all properties. To see what is and isn't implemented, view the window for each test.

To access to the results of the test and status:

```
test = Inspector.TPDOSync()
s = test.status
```

List of properties defined:

status
Test status. One of the values described in the Inspector Constants section. RO

enabled
0 (for disabled) or 1 (for enabled). RO

cycles
Number of test cycles. RO

Replace TPDOSync with TPDOSyncRPDOs, TPDOEvent, TPDOEventRPDOs or PDOTurnaround for the other tests.

To access the details of a specific test cycle, passed is the 1-indexed cycle number:

```
testcycle = Inspector.TPDOSyncCycle(1)
testcycle.time
```

List of properties defined:

cycle
Cycle number. RO

time
Event time or sync time. RO

tpdo_min
TPDO minimum time. RO

tpdo_max
TPDO maximum time. RO

tpdo_diff
TPDO min, max time difference. RO

tpdo_dev
TPDO min, max time deviation. RO

tpdo_ave
TPDO average time. RO

ec_min
Heartbeat or node guarding minimum time. RO

ec_max
Heartbeat or node guarding maximum time. RO

ec_diff
Heartbeat or node guarding min, max time difference. RO

ec_dev
Heartbeat or node guarding min, max time deviation. RO

sdo_min
SDO minimum time. RO

sdo_max
SDO maximum time. RO

sync_min
Sync minimum time. RO

sync_max
Sync maximum time. RO

sync_dev
Sync min, max time deviation. RO

peakmsgrate
Peak message rate reached during test cycle. RO

maxbusload
Maximum bus load during test cycle. RO

maxburstlen
Maximum burst length during test cycle. RO

Replace TPDOSyncCycle with TPDOSyncRPDOsCycle, TPDOEventCycle, TPDOEventRPDOsCycle or PDOTurnaroundCycle for the other tests.

Chapter 8 – Test Reports

8.1 Overview

After the completion of one or more tests, a test report may be generated. The test report is a plain text file containing the node and test configuration information and the results of the tests.

Enough information is included in the test report for someone to identically configure PCANopen Inspector from it and reproduce the tests. Therefore it is ideally suited for formal test reports and sharing of information between different company locations.

8.2 Format

The test report is split into two sections, Human Readable and Machine Readable. Both sections contain the same information, except that in the machine readable section the data is separated by tabs. This allows the machine readable portion of the file to be imported into Excel or other spreadsheet software.

8.3 Generation and Viewing

There are several ways to generate a test report at any time:

- Click on the Generate and View Test Report button on the toolbar
- Choose "Generate and View Test Report" from the Test Reports menu

You will be prompted for a location to save the report to. Once saved the report will be automatically opened in the default view for text files (usually Notepad).

To view the last generated report:

- Click on the View Last Generated Test Report button on the toolbar
- Choose "View Last Test Report" from the Test Reports menu

The test report will be opened in the default viewer for text files (usually Notepad). The location of the last generated report is remembered even when the application is closed, so it is not necessary to generate a report first before using this feature. At least one report in the past must have been generated, however.

8.4 Verification

If a test report is received from another person, it is possible to verify that the report has not been altered since it was generated. Choose "Check Test Report For Changes" from the Test Reports menu and select the test report to verify.

In order to implement this feature, all test reports have a checksum at the bottom of the file. Altering or removing the checksum will render the report unable to be verified by PCANopen Inspector.

Appendix A – CXL Reference

This appendix describes the CXL language used for the Network Description Files. Note that this is a generic file format, and many features may not be applicable to this application.

A.1 File Format

Header and Footer

CXL files are in plain ASCII format. Each file begins with a header:

```
<?xml version="1.0" encoding="UTF-8"?>
<cxl version="1.00" author="Joe Bloggs" date="08-Jul-2004">
```

and ends with a footer:

```
</cxl>
```

Between the header and footer are the top level tags. The top level tags specify sub level tags, which may also in turn specify lower level tags. Refer to the following sections for a full description of each top-level tag.

XML Tag

The XML tag must appear exactly as shown. Even though the encoding is UTF-8, only latin characters are currently supported.

CXL Tag - Version

The version field is mandatory and specifies the CXL file format version. Currently only version 1.00 is defined.

CXL Tag - Author

The author field is optional. The value does not have a defined format.

CXL Tag - Date

The date field is optional. The value has the format:

dd-mmm-yyyy

where *dd* is the day, *mmm* is the first three letters of the month and *yyyy* is the year.

Number Bases

All hexadecimal values in CXL are prefixed with "0x".

Case

Tag names are case sensitive and must appear as they are described in this appendix. The data between the tags is not case-sensitive.

Formatting

Whitespace (spaces, tabs, newlines, carriage returns) are ignored between end and start tags. For example the following are the same:

```
<message>
  <id>0x81</id>
  <name>Node 1 Emergency</name>
</message>
```

```
<message><id>0x81</id><name>Node 1 Emergency</name></message>
```

But whitespace in the data itself is not ignored, so the following are not the same:

```
<message>
  <id>0x81</id>
  <name>Node 1 Emergency</name>
</message>
```

```
<message>
  <id>0x81</id>
  <name>Node 1
  Emergency</name>
</message>
```

A.2 objectdictionary

Defines an Object Dictionary. There may be multiple objectdictionary tags in a symbols file. The subtags are:

- nodeid
  Describes the ID of the node that implements the Object Dictionary. Optional.
- entry
  Describes an Object Dictionary entry

nodeid

If the Object Dictionary relates to a specific node then the nodeid subtag may be included. It specifies the ID of the node in hexadecimal. If it is omitted, then the Object Dictionary entries relate to all nodes.

Example:

```
<nodeid>0x45</nodeid>
```

objectdictionary examples:

```
<objectdictionary>
  <nodeid>0x40</nodeid>
  <entry>
    …
  </entry>
  <entry>
  …
  </entry>
</objectdictionary>

<objectdictionary>
  <entry>
  …
  </entry>
  <entry>
  …
  </entry>
</objectdictionary>
```

entry

Describes a single entry in the Object Dictionary and has the following subtags:

- index
  Describes the index of the entry
- name
  Describes the name of the entry
- subentry
  Describes the subentries in the entry

index

Contains the value in hexadecimal of the index of the entry.

Example:

```
<index>0x1000</index>
```

name

Contains the name associated with the entire Object Dictionary entry.

Example:

```
<name>Device Type</name>
```

subentry

Contains the following subtags that define the subentry:

- subindex
  Describes the subindex of the subentry.
- name
  Describes the name of the subentry. Optional.
- type
  Describes the type of the subentry.

subindex

Contains the value in hexadecimal of the subindex of the subentry.

Example:

```
<subindex>0x00</subindex>
```

name

Contains the name of the subentry. This name is appended to the name of the entry when displayed. If the name subelement is omitted, then no name is appended to the name of the entry. Omitting the name is therefore only useful if the entry has one subentry.

Example:

```
<name>Number of Entries</name>
```

type

Contains the CANopen data type of the subentry. It must be one of the following values (case is ignored):

- BOOLEAN
- INTEGER8
- INTEGER16
- INTEGER24
- INTEGER32
- INTEGER40
- INTEGER48
- INTEGER56
- INTEGER64
- UNSIGNED8
- UNSIGNED16
- UNSIGNED24
- UNSIGNED32
- UNSIGNED40
- UNSIGNED48
- UNSIGNED56
- UNSIGNED64
- REAL32

- REAL64
- VISIBLE_STRING
- OCTET_STRING
- UNICODE_STRING
- TIME_OF_DAY
- TIME_DIFFERENCE
- DOMAIN

Example:

```
<type>UNSIGNED32</type>
```

Examples

The following are examples of an entry:

```
<entry>
  <index>0x1000</entry>
  <name>Device Type</name>
  <subentry>
    <subindex>0x00</subindex>
    <type>UNSIGNED32</type>
  </subentry>
</entry>

<entry>
  <index>0x2000</index>
  <name>Cursor Position</name>
  <subentry>
    <subindex>0x00</subindex>
    <name>Number of Entries</name>
    <type>UNSIGNED8</type>
  </subentry>
  <subentry>
    <subindex>0x01</subindex>
    <name>X</name>
    <type>REAL32</type>
  </subentry>
  <subentry>
    <subindex>0x02</subindex>
    <name>Y</name>
    <type>REAL32</type>
  </subentry>
</entry>
```

A.3 network

Defines network wide information. There may be multiple network elements in a symbols file. The subtags are:

- message
  Describes a message identifier
- abortcode
  Describes an SDO abort code
- errorcode
  Describes an error code
- node
  Describes a node on the network
- devicetype
  Describes a device type
- vendor
  Describes vendors of CANopen nodes

message

Describes a single message and has the following subtags:

- id
  Describes the COB-ID of the message
- name
  Describes the name of the message

id

Describes the COB-ID of the message in hexadecimal.

Example:

```
<id>0x80</id>
```

name

Describes the name of the message. I.e. the name to associate with the ID.

Example:

```
<name>SYNC</name>
```

Examples

The following are examples of message identifier definitions:

```
<message>
  <id>0x80</id>
```

```
  <name>SYNC</name>
</message>
<message>
  <id>0x81</id>
  <name>Node 1 Emergency</name>
</message>
```

abortcode

Describes a single abort code and has the following subtags:

- code
  Describes the abort code.
- name
  Describes the name of the code.

code

Describes the code in hexadecimal.

Example:

```
<code>0x05030000</code>
```

name

Describes the name of the abort code. I.e. the name to associate with the abort code.

Example:

```
<name>Toggle bit not alternated</name>
```

Examples

The following are examples of abort code definitions:

```
<abortcode>
  <code>0x05030000</code>
  <name>Toggle bit not alternated </name>
</abortcode>
```

errorcode

Describes a single error code and has the following subtags:

- code
  Describes the error code.
- name
  Describes the name of the code.

code
Describes the code in hexadecimal.

Example:

```
<code>0x2000</code>
```

name
Describes the name of the error code. I.e. the name to associate with the error code.

Example:

```
<name>Current</name>
```

Examples
The following are examples of error code definitions:

```
<errorcode>
  <code>0x2000</code>
  <name>Current</name>
</errorcode>
```

node
Describes a single node and has the following subtags:

- id
  The ID of the node.
- name
  Describes the name of the node.
- eds
  Describes the Electronic Datasheet for the node.
- simsetupfile
  Describes the setup file for the simulated node. Optional.
- simdll
  Describes the name of the simulation DLL. Optional.
- simio
  Describes the simulation IO file to use when node is simulated. Optional.
- simproduct
  Describes the predefined product to simulate. Optional.

id
Describes the ID of the node in hexadecimal.

Example:

```
<id>0x7F</id>
```

name
Describes the name of the node. I.e. the name to associate with the node.

Example:

```
<name>NMT Master</name>
```

eds
Describes the electronic datasheet of a node by specifying the path to the file. The path may be absolute or relative to the symbol file.

Example:

```
<eds>..\eds\mynode.eds</eds>
```

simsetupfile
Either contains the name of a standard setup file located in the Simulation\Setup Files subfolder (minus the .txt extension), or contains the absolute path to a specific setup file to use. Optional and when used it is assumed that the node is simulated, however it is only used with PCANopen Magic ProDS.

Example:

```
<simsetupfile>Generic IO</simsetupfile>
```

simdll
Describes the name of the DLL to be used to simulate the node. Optional and when used it is assumed that the node is simulated, however it is only used with PCANopen Magic ProDS. The name must the name of a simulation DLL in the Simulation\DLLs subfolder. May also be an absolute path to a specific DLL.

Example:

```
<simdll>MicroCANopen.dll</simdll>
```

simio
Describes the name of the IO file to be used to provide controls when the node is simulated. Optional. Only used with PCANopen Magic ProDS. The name must be the name of a file in the Simulation\IO Files subfolder minus the extension (.sim) or the absolute path to a specific simulation file.

Example:

```
<simio>PCAN_MM_Dig1</simio>
```

simproduct
Describes the product that the node should function like when simulated. Products are defined in the Simulation\Products subfolder, and are a shorthand method of specifying the simsetupfile, simdll and simio parameters. Optional and usually used instead of simsetupfile, simdll and simio. When used it is assumed the node is simulated, however it is only used with PCANopen Magic ProDS.

Example:

```
<simproduct>PEAK MicroMod</simproduct>
```

Examples
The following are examples of node definitions:

```
<node>
  <id>0x01</id>
  <name>Motor Controller</name>
</node>
<node>
  <id>0x7F</id>
  <name>NMT Master</name>
  <eds>..\eds\nmtmaster.eds</eds>
</node>
```

devicetype
Describes a single device type and has the following subtags:

- profile
  The device profile number.
- name
  The name of the device profile.
- Bit
  Describes a single bit in the upper 16 bits of the device type. Optional.

profile
Describes the device profile number.

Example:

```
<profile>0x0191</profile>
```

name
Describes the name of the device profile. I.e. the name to associate with the device profile.

Example:

```
<name>I/O Module</name>
```

bit
Describes a bit in the upper 16 bits of the device type, and has the following subtags:

- position
  The bit position in the range 16 – 31.
- set
  Text to display when the bit is set.
- unset
  Text to display when the but is not set.

position
Defines the position of the bit being described. Must be in the range 16 – 31.

Example:

```
<position>16</position>
```

set
Defines the text to display when the bit is set. May be blank.

Example:

```
<set>Digital inputs</set>
```

unset
Defines the text to display when the bit is unset. May be blank.

Example:

```
<unset>No digital inputs</unset>
```

Examples
The following are examples of device type definitions:

```
<devicetype>
  <profile>0x0191</profile>
  <name>I/O Module</name>
  <bit>
    <position>16</position>
    <set>Digital inputs</set>
    <unset></unset>
  </bit>
</devicetype>
```

vendor
Describes a single CANopen node vendor, and has the following subelements:

- id
  The Manufacturer ID (24-bits).
- name
  The vendor's name
- department
  Describes a department for the vendor (8-bits). Optional.

id

Defines the manufacturer ID for the vendor. This is the lower 24 bits of the Vendor ID.

Example:

```
<id>0x455341</id>
```

name

Defines a name for the vendor.

Example:

```
<name>ESAcademy</name>
```

department

Defines a single department for the vendor. There may be up to 256 department entries for a single vendor. The following subelements are defined:

- id
  The ID of the department.
- name
  The name of the department.

id

Defines the ID for the department. This is the upper 8 bits of the Vendor ID.

Example:

```
<id>0x01</id>
```

name

Defines a name for the department.

Example:

```
<name>USA</name>
```

Examples

The following is an example of a vendor declaration:

```
<vendor>
  <id>0x455341</id>
  <name>ESAcademy</name>
  <department><id>0x00</id><name>Extenal</name></department>
  <department><id>0x01</id><name>USA</name></department>
  <department><id>0x02</id><name>Germany</name></department>
  <department><id>0x03</id><name>ESS Germany</name></department>
</vendor>
```

A.4 processdata

Defines process data that can appear on the bus. There may be multiple processdata elements in a symbols file. The subtags are:

- data
  Defines a single item of process data

data

Defines a single item of process data and has the following subtags:

- name
  The name of the process data
- id
  The COB-ID of the PDO that contains the process data
- startbit
  The starting bit of the process data
- endbit
  The ending bit of the process data
- type
  The type of the process data
- units
  The units of the data. Optional.
- factor
  The scaling factor to apply to the value of the data. Optional.
- offset
  The offset to apply to the value of the data. Optional.

name

The name of the process data. This will be the name displayed.

Example:

```
<name>Temperature</name>
```

id

The message ID of the PDO that contains the process data in hexadecimal.

Example:

```
<id>0x180</id>
```

startbit

The starting bit where the process data can be found inside the PDO's data (0-indexed).

Example:

```
<startbit>8</startbit>
```

endbit

The ending bit where the process data can be found inside the PDO's data (0-indexed).

Example:

```
<endbit>15</endbit>
```

type

The CANopen type of the data. It must be one of the following values (case is ignored):

BOOLEAN
INTEGER8
INTEGER16
INTEGER24
INTEGER32
INTEGER40
INTEGER48
INTEGER56
INTEGER64
UNSIGNED8
UNSIGNED16
UNSIGNED24
UNSIGNED32
UNSIGNED40
UNSIGNED48
UNSIGNED56
UNSIGNED64
REAL32
REAL64
VISIBLE_STRING
OCTET_STRING
UNICODE_STRING
TIME_OF_DAY
TIME_DIFFERENCE
DOMAIN

units

The units for the data. For example, degrees, celcius, flashes, rpm, etc.

Example:

```
<units>RPM</units>
```

offset

An offset applied to the value of the data after the scaling factor has been applied. Allows the data to be given specific ranges. May be a real number. Optional.

value = (value x factor) + offset

The offset is not used for the following types:

BOOLEAN
VISIBLE_STRING
OCTET_STRING
UNICODE_STRING
TIME_OF_DAY
TIME_DIFFERENCE
DOMAIN

Example:

```
<offset>6.252</offset>
```

factor

A scaling factor applied to the value of the data before the offset has been applied. May be a real number. Optional.

value = (value x factor) + offset

The scaling factor is not used for the following types:

BOOLEAN
VISIBLE_STRING
OCTET_STRING
UNICODE_STRING
TIME_OF_DAY
TIME_DIFFERENCE
DOMAIN

Example:

```
<factor>-45.32</factor>
```

Examples
The following are examples of Process Data declarations:

```
<data>
  <name>Temperature</name>
  <id>0x180</id>
  <startbit>0</startbit>
  <endbit>31</endbit>
  <type>REAL32</type>
  <units>degrees C</units>
</data>
<data>
  <name>Data Size</name>
  <id>0x204</id>
  <startbit>8</startbit>
  <endbit>15</endbit>
  <type>UNSIGNED8</type>
  <units>bytes</units>
</data>
```

A.5 comments

Allows comments to be inserted into the file. All contents inside the comments tags are ignored and may be arbitary.

Example:

```
<comments>
Revision 1.00
History: Added new node definition, AA, 06-Jul-2005
</comments>
```

A.6 Example File

The following is an example Network Description File:

```
<?xml version="1.0" encoding="UTF-8"?>
<cxl version="1.00" author="AAyre" date="26-Jan-2004">

  <objectdictionary>
    <entry>
      <index>0x1000</index>
      <name>Device Type</name>
      <subentry>
        <subindex>0x00</subindex>
        <type>UNSIGNED32</type>
      </subentry>
```

```
    </entry>
    <entry>
      <index>0x2000</index>
      <name>Cursor Position</name>
      <subentry>
        <subindex>0x00</subindex>
        <name>Number of Entries</name>
        <type>UNSIGNED8</type>
      </subentry>
      <subentry>
        <subindex>0x01</subindex>
        <name>X</name>
        <type>REAL32</type>
      </subentry>
      <subentry>
        <subindex>0x02</subindex>
        <name>Y</name>
        <type>REAL32</type>
      </subentry>
    </entry>
  </objectdictionary>

  <objectdictionary>
    <nodeid>0x21</nodeid>
    <entry>
      <index>0x2500</index>
      <name>Temperature</name>
      <subentry>
        <subindex>0x00</subindex>
        <type>REAL32</type>
      </subentry>
    </entry>
  </objectdictionary>

  <network>
    <message>
      <id>0x80</id>
      <name>SYNC</name>
    </message>
    <message>
      <id>0x181</id>
      <name>Environment</name>
    </message>
    <abortcode>
      <code>0x05030000</code>
      <name>Toggle bit not alternated</name>
    </abortcode>
    <errorcode>
      <code>0x2000</code>
```

```
    <name>Current</name>
   </errorcode>
   <node>
    <id>0x21</id>
    <name>Environment Monitor</name>
   </node>
   <node>
    <id>0x7F</id>
    <name>NMT Master</name>
   </node>
   <devicetype>
    <profile>0x0191</profile>
    <name>I/O</name>
    <bit>
     <position>16</position>
     <set>Digital inputs</set>
     <unset></unset>
    </bit>
   </devicetype>
   <vendor>
    <id>0x455341</id>
    <name>ESAcademy</name>
    <department><id>0x00</id><name>Extenal</name></department>
    <department><id>0x01</id><name>USA</name></department>
    <department><id>0x02</id><name>Germany</name></department>
    <department><id>0x03</id><name>ESS Germany</name></department>
   </vendor>
  </network>

  <processdata>
   <data>
    <name>Temperature</name>
    <id>0x180</id>
    <startbit>0</startbit>
    <endbit>31</endbit>
    <type>REAL32</type>
    <units>degrees C</units>
   </data>
  </processdata>

</cxl>
```

This example symbols file defines the following:

- There are two nodes on the network, Environment Monitor (ID 0x21) and NMT Master (ID 0x7F).
- The PDO with COB-ID 0x181 contains environmental information, and it's first four bytes contain the temperature in degrees C.

- All nodes define the Device Type at OD entry 0x1000 and a Cursor Position at 0x2000.
- The Environment Monitor node also defines Temperature at OD entry 0x2500.
- The SYNC message has the COB-ID 0x80.
- A Device Type of 0x00000191 is an I/O node. When bit 16 is set, the device supports digital inputs.
- The vendor ESAcademy is defined with four departments within the vendor.
- Values are defined for error and abort codes.