



# NTCAN

## Part 2: Installation, Configuration and Firmware Update



## Installation Guide



---

## NOTE

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document and assumes no responsibility for any errors that may appear in this document. In particular descriptions and technical data specified in this document may not be constituted to be guaranteed product features in any legal sense.

**esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance, or design.

All rights to this documentation are reserved by **esd**. Distribution to third parties, and reproduction of this document in any form, whole or in part, are subject to **esd**'s written approval.

© 2024 esd electronics gmbh, Hannover

esd electronics gmbh  
Vahrenwalder Str. 207  
30165 Hannover  
Germany

Phone: +49-511-372 98-0  
Fax: +49-511-372 98-68  
E-Mail: [info@esd.eu](mailto:info@esd.eu)  
Internet: [www.esd.eu](http://www.esd.eu)

### Trademark Notices

.NET: The Microsoft®.NET logo is a registered trademark of Microsoft Corporation in the United States and/or other countries.

CiA® and CANopen® are registered community trademarks of CAN in Automation e.V..

Linux® is the registered trademark of Linus Torvalds in the United States and/or other countries.

Microsoft®, Windows®, Windows Vista®, and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries.

QNX® and Neutrino® are registered trademarks of QNX Software Systems Limited, and are registered trademarks and/or used in certain jurisdictions.

Solaris™ is a trademark of Sun Microsystems, Inc. in the United States and in other countries.

TENASYS® and INTIME® are registered trademarks of TenAsys Corporation.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

VxWorks® is a registered trademark of Wind River Systems, Inc.

All other trademarks, product names, company names or company logos used in this manual are reserved by their respective owners.

<b>Document file:</b>	I:\Texte\Doku\MANUALS\PROGRAM\CAN\C.2001.21_NTCAN\Installation\NTCAN_Installation_en_48.docx
<b>Date of print:</b>	2024-02-01
<b>Document type number:</b>	DOC0800

## Products covered by this document.

<b>CAN-Driver / SDK</b>	<b>(Driver) Revision</b>
CAN SDK for Windows	4.x.y
Windows 95/98/ME VxD-Driver	1.x.y
Windows NT Device Driver	2.x.y
Windows 2000	2.x.y
Windows XP (32/64-Bit)	3.x.y
Windows Vista (32/64-Bit)	4.x.y
Windows 7 (32/64-Bit)	
Windows 8 / 8.1 (32/64-Bit)	
Windows 10 (32/64-Bit)	
Windows 11 (64-Bit)	
Linux Driver (32-/64-Bit)	3.x.y 4.x.y
LynxOS Driver	1.x.y
PowerMAX OS Driver	1.x.y
Solaris-Driver	3.x.y
SGI-IRIX6.5 Driver	2.x.y
AIX Driver	1.x.y
VxWorks 5.x/6.x (Non-VxBus)	2.x.y
VxWorks 6.x (VxBus)	3.x.y
VxWorks 7.x (VxBus GEN2)	4.x.y
QNX4 Driver	2.x.y
QNX6 / QNX 7 Driver	3.x.y / 4.x.y
RTOS-UH Driver	2.x.y
RTX / RTX64 Driver	3.x.y / 4.x.y
INtime Driver	4.x.y
On Time RTOS-32	3.x.y

<b>CAN-Hardware</b>	<b>Order No.</b>
EtherCAN	C.2050.xx
EtherCAN/2	C.2051.xx
CAN-ISA/200	C.2011.xx
CAN-ISA/331	C.2010.xx
CAN-PC104/200	C.2013.xx
CAN-PC104/331	C.2012.xx
CAN-PCI104/200	C.2046.xx
CAN-PCI/200	C.2021.xx
CAN-PCI/266	C.2036.xx
CAN-PCI/331	C.2020.xx
CAN-PCI/360	C.2022.xx
CAN-PCI/400	C.2048.xx
CAN-PCI/402	C.2049.xx

<b>CAN-Hardware</b>	<b>Order No.</b>
<b>CAN-PCI/402-FD</b>	C.2049.xx
CAN-PCI/405	C.2023.xx
CAN-PCle/200	C.2042.xx
CAN-PCle/400	C.2043.xx
CAN-PCle/402	C.2045.0x
<b>CAN-PCle/402-FD</b>	C.2045.xx
CAN-PCleMini/402	C.2044.xx
<b>CAN-PCleMini/402-FD</b>	C.2044.xx
<b>CAN-PCleMiniHS/402-FD</b>	C.2054.xx
<b>CAN-M.2/402-2-FD</b>	C.2074.xx
PMC-CAN/266	C.2040.xx
PMC-CAN/331	C.2025.xx
PMC-CAN/400	C.2047.xx
<b>PMC-CAN/402-FD</b>	C.2028.xx
PMC-CPU/405	V.2025.xx
CPCI-CAN/200	C.2035.xx
CPCI-CAN/331	C.2027.xx
CPCI-CAN/360	C.2026.xx
CPCI-CAN/400	C.2033.xx
CPCI-CAN/402	I.2332.xx
<b>CPCI-CAN/402-FD</b>	I.2332.xx
CPCI-405	I.2306.xx
CPCI-CPU/750	I.2402.xx
CPCIsreal-CAN/402	I.3001.04
<b>CPCIsreal-CAN/402-FD</b>	I.3001.6x
CAN-PCC	C.2422.xx
CAN-USB/Mini	C.2464.xx
CAN-USB/Micro	C.2068.xx
CAN-USB/2	C.2066.xx
<b>CAN-USB/3-FD</b>	C.2076.xx
CAN-USB/400	C.2069.xx
<b>CAN-USB/400-FD</b>	C.2069.xx
CAN-CBX-AIR/2	C.3051.xx
CAN-CBX-AIR/3	C.3052.xx
VME-CAN2	V.1405.xx
VME-CAN4	V.1408.xx
AMC-CAN4	U.1002.xx
<b>XMC-CAN/402-FD</b>	C.2018.xx

## Document History

The changes in the document listed below affect changes in the software as well as changes in the description of the facts, only.

<b>Rev.</b>	<b>Chapter</b>	<b>Changes versus previous version</b>	<b>Date</b>
4.8	2.9	Moved the content covering Windows 8.x to the chapter for legacy Windows versions.	2024-01-02
	4.4	New chapter which covers the support for the real-time OS INtime®	2024-01-08
	4.6	Moved the content covering On-Time RTOS-32 to the chapter for legacy RTOS versions.	2024-01-02
4.7	1.5.2	Updated USB Hardware ID table	2021-10-22
	2.4.4	New chapter to enforce driver installation.	2021-10-25
	2.8.1	Revised document for the latest changes in the Windows driver code signing policies.	2021-10-22
	2.9	Moved the chapters for Windows XP, Vista and 7 to the legacy Windows versions.	2021-10-22
	3.2	Moved the content for PowerMAX OS, Solaris, SGI-IRIX and AIX to the chapter for legacy UNIX versions.	2021-10-22
	5.1	Extended table 18 for CAN-USB/2V2 and CAN-USB/3-FD	2021-10-25
	N/A	Editorial changes for Windows 11 support.	2021-10-22
4.6	5.1	Revised and extended description of update process	2018-09-19
	N/A	Editorial changes	2019-07-25
4.5	1.5.1	Updated PCI Hardware ID table	2018-01-22
	1.5.2	Updated USB Hardware ID table	2017-10-30
	2.8	Completely revised to cover Microsoft's SHA-1 deprecation policy.	2018-05-28
	4.1	Added VxWorks 7.0 support	2018-06-07
	4.2.1	Updated for QNX7	2018-05-08
	N/A	Editorial changes	2018-06-11
4.4	N/A	Editorial changes	2017-06-08
4.3	2	Description to preinstall device drivers on Windows.	2016-02-08
	2.1	Description of device driver installation on Windows 10.	2016-02-09
	2.2.1	Device Driver Configuration completely revised and extended.	2016-02-12
	2.4	New chapter to troubleshoot Windows driver installation.	2016-03-11
	2.9	Moved Windows 2000 to the legacy Windows versions.	2016-02-09
	N/A	Editorial changes	2016-04-06
4.2	2	Added description of WDF based CAN device driver.	2014-12-15
	2.2.2	Revised description of CAN driver property sheet in device manager with new chapter how to configure IRQ affinity	2015-01-23
	N/A	Added CAN-USB/400 specific information	2015-01-23
	N/A	Editorial changes	2014-12-07
4.1	4.4	Revised RTX installation instructions with regard to RTX64 2014.	2014-11-24
4.0	N/A	The complete document is revised, large parts of the text are rewritten, and the drawings are updated.	2013-08-12

<b>Rev.</b>	<b>Chapter</b>	<b>Changes versus previous version</b>	<b>Date</b>
	1	Introduction completely rewritten.	2013-08-12
	2.1	Description of device driver installation on Windows 8.	2013-08-12
	2.9.1	Description of device driver installation on Windows 7.	2013-08-12
	2.8	New chapter about Windows Digital Signatures.	2013-08-12
	3.1	New introduction for Linux driver.	2013-08-12
	3.1.2.2	Linux CAN-Module IDs: Modules inserted (AMC-CAN4, CAN-PCI/400, CAN-PCIe/400, CPCI-CAN/400, PMC-CAN/400), Modules deleted (CAN-USB/2, VME-CAN2, VME-CAN4)	2013-08-12
	3.1.2.3	Unpacking of Linux TGZ archive for drivers released after 07-2012 inserted	2013-08-12
	-	Chapter 'Installation of Linux drivers < 3.x.x.x deleted'.	2013-08-12
	3.1.3	New chapter with installation for Linux CAN (aka SocketCAN)	2013-08-12
	3.1.4	New chapter for Linux EtherCAN/2 installation.	2013-08-12
	4.1	Installation on VxWorks completely rewritten.	2013-08-12
	4.2.1	QNX6: Program for setting the PCI Class inserted. CAN interface family C400 support inserted. Additional commands for Resource Manager.	2013-08-12
	4.4	Installation on RTX completely revised and extended with installation instructions for RTX64.	2013-08-12
	5	Description of firmware update completely revised and combined in one chapter for all platforms.	2013-08-12

Technical details are subject to change without further notice.

---

## Typographical Conventions

Throughout this manual the following typographical conventions are used to distinguish technical terms.

<i>Convention</i>	<i>Example</i>
File and path names	<code>/dev/null</code> Or <code>&lt;stdio.h&gt;</code>
Function names	<b><i>open()</i></b>
Programming constants	<code>NULL</code>
Programming data types	<code>uint32_t</code>
Variable names	<i>Count</i>

The following indicators are used to highlight noticeable descriptions.



### Note

Notes to point out something important or useful.



### Attention!

Cautions to tell you about operations which might have unwanted side effects.

## Number Representation

All numbers in this document are **base 10** unless designated otherwise. Hexadecimal numbers have a prefix of **0x**, and binary numbers have a prefix of **0b**. For example, 42 is represented as 0x2A in hexadecimal and 0b101010 in binary.

---

## Abbreviations

ABI	Application Binary Interface
AMC	Advanced Mezzanine Cards
API	Application Programming Interface
BSD	Berkley Software Distribution
BSP	Board Support Package
CAN	Controller Area Network
CPU	Central Processing Unit
CiA	CAN in Automation
COTS	Commercial off-the-shelf
CPCI	Compact Peripheral Component Interconnect (Computer Bus)
CPCIs <sub>serial</sub>	Compact Peripheral Component Interconnect Serial (Computer Bus)
CRC	Cyclic Redundancy Check
DLC	Data Length Code
DLL	Dynamic Link Library
DNS	Domain Name Service
EFF	Extended Frame Format
ACC	Advanced CAN Controller
FIFO	First-In-First-Out
FTP	File Transfer Protocol
FW	Firmware
HW	Hardware
I/O	Input/Output
ISA	Industry Standard Architecture (Computer Bus)
IPC	Interprocess Communication
IRIG	Inter-range Instrumentation Group
LSB	Least Significant Bit
LSW	Least Significant Word
MCU	Micro Controller Unit
MSB	Most Significant Bit
MSW	Most Significant Word
NVRAM	Non-Volatile Random Access Memory
N/A	Not Applicable
OS	Operating System
PCI	Peripheral Component Interconnect (Computer Bus)
PCIe	Peripheral Component Interconnect Express (Computer Bus)
PIC	Programmable Interrupt Controller
PMC	PCI Mezzanine Card
PnP	Plug and Play
RAM	Random Access Memory
REC	Receive Error Counter
RTSS	Real-Time Sub-System
SDK	Software Development Kit
SMP	Symmetric Multiprocessor
SoC	System on Chip
SFF	Standard (Base) Frame Format
SW	Software
TEC	Transmit Error Counter
USB	Universal Serial Bus
UP	Uniprocessor
URL	Universal Resource Locator
WDM	Windows Driver Model
WDF	Windows Driver Foundation
WHQL	Windows Hardware Quality Lab
WoW64	Windows 32-bit on Windows 64-bit
XMC	Express Mezzanine Card

---

## Reference

- /1/ esd electronics gmbh, *NTCAN-API Application Developers Manual*, Revision 5.8, 2024
- /2/ ISO 11898-1, *Road vehicles – Controller area network (CAN) – Data link layer and physical signalling*, 2015
- /3/ esd electronic system design gmbh, *EtherCAN/2 Hardware Manual*, Revision 2.0, 2021
- /4/ esd electronic system design gmbh, *CAN-Wiring*, Revision 4.6, 2022



# Table of Contents

1	Introduction .....	14
1.1	Scope .....	14
1.2	Overview .....	15
1.3	Terminology .....	16
1.4	CAN Interface Families .....	18
1.5	Hardware IDs .....	19
1.5.1	PCI / PCIe / PCIe Mini / CPCI / CPCIs serial / PMC / XMC .....	19
1.5.2	USB .....	21
1.5.3	Ethernet .....	21
1.6	Software Deployment .....	22
1.6.1	Windows .....	22
1.6.2	Linux / Unix .....	22
1.6.3	Real-Time Operating Systems .....	22
2	Windows® .....	23
2.1	Windows 10 / 11 .....	27
2.1.1	Hardware-First Driver Installation .....	28
2.1.2	Software-First Driver Installation .....	33
2.1.3	Driver Lifecycle Management .....	33
2.2	Configuration .....	34
2.2.1	Device Driver .....	34
2.2.1.1	Standard Settings .....	34
2.2.1.2	Expert Settings .....	37
2.2.1.3	Device Specific Settings .....	38
2.2.2	System .....	39
2.2.2.1	Power Management .....	39
2.2.2.2	Interrupt Affinity .....	39
2.3	Device Driver Preinstallation .....	40
2.3.1	Driver Staging .....	40
2.3.2	Driver Installation for Non-Administrators .....	41
2.4	Troubleshooting Driver Installation .....	43
2.4.1	Error Code 31 .....	44
2.4.2	Error Code 39 .....	45
2.4.3	Error Code 52 .....	45
2.4.4	Best Driver already Installed .....	46
2.5	Device Driver Lifecycle Management .....	48
2.5.1	Driver Update .....	48
2.5.2	Driver Rollback .....	49
2.5.3	Driver Uninstall .....	50

2.6	EtherCAN and EtherCAN/2 .....	51
2.6.1	Installation .....	51
2.6.2	Configuration .....	52
2.6.3	Uninstall.....	53
2.7	Windows CAN Software Development Kit (SDK) .....	54
2.7.1	Setup Command Line Parameter.....	54
2.7.2	Installation Options .....	55
2.7.3	Uninstall.....	56
2.7.4	IDE Integration.....	56
2.8	Digital Signatures .....	57
2.8.1	Overview.....	57
2.8.2	Driver Installation .....	59
2.8.3	Software Installation .....	60
2.8.4	Digital Signature Verification .....	61
2.9	Legacy Windows Versions .....	63
2.9.1	Windows 7 / 8.x / Server 2008 R2.....	63
2.9.1.1	Hardware-First Driver Installation.....	63
2.9.1.2	Software-First Driver Installation .....	67
2.9.1.3	Driver Lifecycle Management.....	67
2.9.2	Windows Vista® and Server 2008 .....	68
2.9.2.1	Hardware-First Driver Installation.....	68
2.9.2.2	Software-First Driver Installation .....	72
2.9.2.3	Driver Lifecycle Management.....	72
2.9.3	Windows XP and Server 2003 .....	73
2.9.3.1	Hardware-First Driver Installation.....	73
2.9.3.2	Software-First Driver Installation .....	77
2.9.3.3	Driver Lifecycle Management.....	77
2.9.4	Windows 2000 .....	78
2.9.4.1	Hardware-First Driver Installation.....	78
2.9.4.2	Software-First Driver Installation .....	82
2.9.4.3	Driver Lifecycle Management.....	82
2.9.4.4	Non-PnP hardware .....	83
2.9.5	Windows NT 4.0 .....	84
2.9.5.1	Driver Installation .....	84
2.9.5.2	Driver Configuration.....	85
2.9.5.3	Driver Start .....	88
2.9.5.4	Driver Uninstall .....	89
2.9.6	Windows 9x/ME .....	89
2.9.6.1	Installation of PnP CAN modules .....	90
2.9.6.2	Installation of non-PnP CAN modules .....	90
3	Unix® Operating Systems .....	101

3.1	Linux® .....	101
3.1.1	CAN Board Support Overview .....	103
3.1.2	NTCAN Driver .....	105
3.1.2.1	Files of the Linux Package .....	106
3.1.2.2	CAN-Module-ID and Default Parameters of the Driver .....	108
3.1.2.3	Installation .....	109
3.1.3	Linux CAN Driver (aka SocketCAN) .....	115
3.1.3.1	Integration .....	115
3.1.3.2	Installation .....	115
3.1.3.3	Configuration .....	116
3.1.3.4	Restrictions .....	117
3.1.4	EtherCAN and EtherCAN/2 .....	119
3.1.4.1	Installation .....	120
3.1.4.2	Configuration .....	121
3.1.4.3	Miscellaneous .....	122
3.2	Legacy UNIX Versions .....	123
3.2.1	PowerMAX OS Installation .....	123
3.2.1.1	Files of the PowerMAX OS Package .....	123
3.2.1.2	Sequence of Installation Under PowerMAX OS .....	124
3.2.2	Solaris™ Installation .....	126
3.2.2.1	Files of the Solaris Package .....	126
3.2.2.2	Sequence of Installation Under Solaris .....	127
3.2.3	SGI-IRIX6.5 Installation .....	131
3.2.3.1	Files of the SGI-IRIX6.5-Package .....	131
3.2.3.2	Sequence of Installation Under SGI-IRIX6.5 .....	132
3.2.4	AIX Installation .....	133
3.2.4.1	Special Features of the AIX Implementation .....	133
3.2.4.2	Files of the AIX Package .....	133
3.2.4.3	Installation Sequence under AIX .....	134
4	Real-Time Operating Systems .....	136
4.1	VxWorks® .....	136
4.1.1	CAN Board Support Overview .....	137
4.1.2	Driver Integration .....	139
4.1.2.1	VxWorks 5.x .....	139
4.1.2.2	VxWorks 6.x .....	140
4.1.2.3	VxWorks 7.x .....	141
4.1.3	Driver Configuration .....	143
4.1.3.1	VxWorks 5.x .....	143
4.1.3.2	VxWorks 6.x (Non-VxBus) .....	146
4.1.3.3	VxWorks 6.x (VxBus) .....	150
4.1.3.4	VxWorks 7.x (VxBus GEN2) .....	153

4.1.4	Driver Start .....	156
4.1.4.1	VxWorks 5.x .....	156
4.1.4.2	VxWorks 6.x (Non-VxBus) .....	156
4.1.4.3	VxWorks 6.x/7.x (VxBus) .....	157
4.1.5	Miscellaneous .....	158
4.1.5.1	Unresolved Symbols Building the VxWorks Image .....	158
4.1.5.2	Number of Available NTCAN Handles .....	158
4.1.5.3	Test Program 'canTest' .....	158
4.1.5.4	Unexpected Behaviour of Software Timestamps .....	158
4.1.5.5	Correct Interpretation of Error Codes Returned by the Driver .....	159
4.1.5.6	Support of the CAN Extended Frame Format .....	160
4.1.6	Troubleshooting Hints .....	161
4.1.6.1	Where to Implement Needed Configuration Changes .....	162
4.1.6.2	Public Interface of the Version 2.x Driver Core .....	162
4.1.6.3	Public Interface of the VME-CAN4 Driver Core .....	163
4.1.6.4	Address Translation and Board Access Issues .....	163
4.1.6.5	Interrupt Connection Issues .....	169
4.1.6.6	VxBus Driver Prerequisites .....	172
4.2	QNX .....	175
4.2.1	QNX 6 and QNX 7 .....	175
4.2.1.1	Driver Package Content .....	175
4.2.1.2	Sequence of Installation Under QNX6 and QNX7 .....	176
4.2.2	QNX4 .....	180
4.2.2.1	Files of the QNX4 Packages .....	180
4.2.2.2	Sequence of Installation Under QNX4 .....	181
4.3	IntervalZero RTX® and RTX64® .....	184
4.3.1	Driver Integration .....	185
4.3.1.1	RTX .....	185
4.3.1.2	RTX64 .....	186
4.3.2	Driver Installation .....	187
4.3.2.1	RTX .....	187
4.3.2.2	RTX64 .....	188
4.3.3	Driver Start .....	191
4.3.4	Driver Configuration .....	192
4.3.4.1	Command Line Parameter .....	192
4.3.4.2	SMP Support .....	192
4.3.5	Miscellaneous .....	193
4.3.5.1	Application Development .....	193
4.3.5.2	Example Application .....	193
4.4	TenAsys® INtime® .....	194
4.4.1	Driver Integration .....	195

---

4.4.2	Driver Installation .....	196
4.4.2.1	INtime for Windows.....	196
4.4.3	Driver Start .....	200
4.4.4	Driver Configuration .....	201
4.4.4.1	Command Line Parameter .....	201
4.4.4.2	Priority Layout.....	202
4.4.5	Driver Unload.....	203
4.4.6	Miscellaneous.....	203
4.4.6.1	Application Development .....	203
4.4.6.2	Example Application .....	203
4.5	Legacy Support .....	204
4.5.1	LynxOS.....	204
4.5.1.1	Driver Installation .....	205
4.5.2	OnTime RTOS-32 .....	206
4.5.2.1	Overview .....	206
4.5.2.2	Implementation .....	206
4.5.2.3	RTOS-32 Software Requirement .....	206
4.5.2.4	Required (RTOS-32) Libraries .....	206
4.5.2.5	Using the esd NTCAN Library.....	207
4.5.2.6	Compiling the Sample Application “cantest” .....	209
5	Firmware Update Application.....	215
5.1	Updating the Firmware .....	216
5.2	Switch between CAN 2.0A and CAN 2.0B Mode.....	218

# 1 Introduction

This document describes the device driver installation and configuration process of the cross-platform architecture for **esd** Controller Area Network (CAN) hardware as well as the steps to update the firmware of active CAN boards.

Within this document this architecture and its usual implementation as a combination of a device driver and a library is referred to as **NTCAN**. The name has its origin in the initial implementation for Windows NT but it is now the common API for all Operating Systems (OS).

The CAN bus is specified in /2/.



**Attention:**

Before you start with the software installation refer to the CAN board hardware manual how to install the device mechanically and electrically in your system.

Please refer to the *CAN-Wiring* document [/4/](#) for further information on the wiring of the CANbus, cable selection, correct termination, etc.

The documents are either located on the CD which comes with your hardware or can be downloaded from the **esd** web site (<https://www.esd.eu/>)

## 1.1 Scope

This document covers the CAN board and operating system specific installation of the NTCAN architecture which usually consists of an OS and CAN hardware specific device driver and a (shared) library which exports the application interface to integrate CAN I/O into an application as well as required files to use the API in your application (header files, startup code, ...).



The NTCAN architecture and the API (which is identical for all platforms) to develop CAN based applications is described in the first part of the CAN-API documentation called 'NTCAN, Part 1: Application Developers Manual' **/1/**.

The NTCAN architecture is implemented for the following desktop, embedded, real-time and UNIX operating systems.

<i>Windows</i>	<i>UNIX</i>	<i>Real-Time</i>
<b>Active</b>		
Windows 11 (64 Bit) Windows 10 (32-/64-Bit)	Linux (32-/64-Bit)	QNX 6/7 VxWorks 5.x/6.x/7.x RTX64 INttime 6.x/7.x
<b>Legacy</b>		
Windows 9x/ME Windows NT Windows 2000 Windows XP (32-/64-Bit) Windows Vista (32-/64-Bit) Windows 7 (32-/64-Bit) Windows 8 / 8.1 (32-/64-Bit)	AIX PowerMAX OS SGI-IRIX 6.5 Solaris	LynxOS On-Time RTOS-32 QNX4 RTX

**Table 1:** Supported Operating Systems



**Attention:**

*esd electronics gmbh* does no longer provide support and maintenance for operating systems which are considered as **Legacy** according to the table above.

## 1.2 Overview

The document is made up of 5 main chapters with the following topics:

**Chapter 1** contains a general overview on the structure of this manual.

**Chapter 2** describes the installation process on the various version of Windows®.

**Chapter 3** covers the installation process on Unix® based operating systems (inclusive Linux®).

**Chapter 4** describes the device driver integration in various (embedded) real time OS.

**Chapter 5** provides information about the firmware update process of active CAN boards.

## 1.3 Terminology

Within this manual you will encounter the following terms:

<b>Base Frame Format</b>	CAN messages with 11-bit CAN-IDs according to /2/
<b>Board Support Package</b>	<b>A Board Support Package (BSP)</b> is the common name for the hardware specific code which is necessary for an operating system to support an embedded board. It typically consists at least of the code to initialize the hardware to a point to load and start the operating system and all device drivers to support the on-board interfaces.
<b>CAN</b>	<b>Controller Area Network</b> A serial bus system (also known as CAN bus) that was originally designed for use in vehicles but is now also used in automation technology.
<b>CAN Board</b>	A CAN board is a hardware which makes one or more physical CAN ports available for use by an application. This is either an <b>esd CAN Interface</b> or an embedded system with an on-board <i>CAN Controller</i> .
<b>CAN Controller</b>	A chip whose hardware processes the CAN bus protocols. This can be a stand-alone chip which is solely dedicated to this function or a <i>System on Chip (SoC)</i> which integrates one or more CAN controllers as external interface.
<b>CAN Device</b>	The (logical) application view to a physical CAN port.
<b>CAN Family</b>	A CAN family describes the group of CAN boards which are handled with the same device driver (see chapter 1.4).
<b>CAN Handle</b>	Logical link between the application and a physical CAN port. An application can open several CAN handles to the same or to different CAN ports.
<b>CAN-ID</b>	Identifier of a CAN message either in the <i>Standard Frame Format</i> (11-bit) or the <i>Extended Frame Format</i> (29-bit)
<b>CAN Interface</b>	A CAN interface is a dedicated <b>esd</b> hardware which is either connected to a local bus (PCI, USB, PC/104, etc.) of a CPU or remotely (Ethernet, Wireless, etc.) to a host system.
<b>CAN Node</b>	All hardware connected to the CAN bus. This can be any hardware with a CAN port ranging from a simple sensor up to a complex control system.
<b>CAN Message</b>	Logical unit which consists of a CAN-ID and a payload either as data frame or as remote request frame.
<b>CAN Port</b>	The physical connector to a CAN bus which is handled by a CAN controller.
<b>Driver Store</b>	Repository for device driver introduced with Windows Vista.
<b>Extended Frame Format</b>	CAN messages with 29-bit CAN-IDs according to /2/
<b>Hot Plugging</b>	Hot Plugging describes the capability of a hardware to be connected or replaced without shutting down the system.
<b>INTx</b>	<b>INTx</b> or Legacy Interrupts describe the method to signal interrupts with the help of dedicated pins in contrast to <b>MSI</b> .
<b>Legacy Interrupt</b>	See <b>INTx</b> .



<b>IRIG B</b>	Time code format used to provide time-of-day information to communication systems which have to correlate data (reception) with time.
<b>MSI</b>	<b>Message Signaled Interrupts</b> are defined in the PCIe standard as a method to signal interrupts as an in-band message instead of using dedicated pins. One of the advantages compared to traditional INTx signaling is that MSIs are never shared.
<b>Plug and Play (PnP)</b>	Hardware capability to support automatic device detection and configuration without user intervention. Examples for PnP capable buses are PCI, PCIe, CPCI, PMC, PCI104 and USB.
<b>RTSS</b>	<b>Real-Time Sub-System</b> for Windows provide by RTX/RTX64.
<b>Standard Frame Format</b>	Same as <i>Base Frame Format</i> .
<b>UAC</b>	<b>User Account Control</b> is a security infrastructure introduced with Microsoft Windows Vista.
<b>VxBus</b>	Software infrastructure to integrate device drivers in the real time operating system VxWorks with minimal BSP support.
<b>VxD</b>	Device driver model of Windows 9x/ME.
<b>WDM</b>	The <b>Windows Driver Model</b> was introduced with Windows 98/2000 as a common driver model to replace VxD and the Windows NT Driver Model with a unified API.
<b>WDF</b>	The <b>Windows Driver Foundation</b> is a WDM based driver model (framework) introduced in 2006 which provides a robust object-based interface for device drivers.
<b>WHQL</b>	<b>Windows Hardware Quality Labs (WHQL) Testing</b> is Microsoft's testing process which involves running a series of tests on third-party hardware/software. For device driver passing the WHQL tests, Microsoft creates a digitally signature.
<b>WoW64</b>	<b>WoW64 (Windows 32-bit on Windows 64-bit)</b> is a subsystem included on all 64-bit versions of Windows that is capable of running 32-bit applications.

## 1.4 CAN Interface Families

A single **esd** NTCAN device driver supports in most cases more than one CAN interface. For this reason, the operating system specific installation and configuration instructions of a device driver in this manual are usually intended for complete CAN interface family covering several CAN interfaces. Each CAN family device driver is assigned a unique signature which is used in log messages, etc. to distinguish drivers of different CAN interface families on the same host system and a device driver name which follows one of two possible naming conventions.

The table below gives an overview on the CAN interfaces or boards that are covered by a CAN family device driver, their assigned signatures and device driver names:

<b>CAN Interface Families</b>	<b>Family Name</b>	<b>Signature</b>	<b>Driver Name (Naming Convention I)</b>	<b>Driver Name (Naming Convention II)</b>
CAN-PCI/200, CAN-PCI/266 CAN-PCIe/200, CAN-PCI104/200 CPCI-CAN/200, PMC-CAN/266	C200	C200	c200	pci200-sja1000
CAN-PCI/331, CPCI-CAN/331, PMC-CAN/331	C331	C331	c331	pci331-i20
CAN-PCI/360, CPCI-CAN/360	C360	C360	c360	pci360-i20
CAN-PCI/400, CAN-PCIe/400, CPCI-CAN/400, PMC-CAN/400	C400	C400	c400	pci400-esdacc
CAN-PCI/402, CAN-PCIe/402, CAN-PCIMini/402, CPCI-CAN/402, CPCISerial-CAN/402, PMC-CAN/402, XMC-CAN/402, CAN-M.2/402-2-FD, CAN-PCIeMiniHS/402 + CAN FD enabled derivatives	C402	C402	c402	pci402-esdacc
CAN-PCI/405	C405	C405	c405	pci405-pcimsgx
CAN-USB/Mini	USB1	U331	usb331	N/A
CAN-USB/Micro, CAN-USB/2 CAN-USB/2V2, CAN-AIR/2	USB2	U2292	usb2292	N/A
CAN-USB/3-FD	USB3	U3FD	usb3fd	N/A
CAN-USB/400 CAN-USB/400-IRIG-B	U400	U400	u400	N/A
CAN-ISA/200, CAN-PC104/200, Memory mapped NXP SJA1000	C200I	C200I	c200i	isa200-sja1000
CAN-ISA/331, CAN-PC104/331	C331I	C331I	c331i	isa331-i20
CAN-VME2	CAN2	ICAN2	ican2	N/A
CAN-VME4	CAN4	ICAN4	ican4	N/A

**Table 2:** Overview of the CAN Interface Families

## 1.5 Hardware IDs

To identify and **esd** (P'n'P) CAN interface in a system or to troubleshoot an installation it is sometimes helpful to know it's bus specific hardware ID. This chapter gives an overview on the hardware IDs for the **esd** Commercial off-the-shelf (COTS) CAN interfaces.

The knowledge about the hardware IDs helps troubleshooting installations and interpreting log messages.

### 1.5.1 PCI / PCIe / PCIe Mini / CPCI / CPCIs serial / PMC / XMC

An **esd** CAN interface attached to the PCI, PCIe, PCIe Mini, CPCI, CPCIs serial, PMC or XMC bus can be unambiguously identified by the ID pair for the main chip (Vendor ID and Device ID) and a vendor specific ID pair for the device (Subsystem Vendor ID and Subsystem ID). Each ID is 16-bit and the Vendor and Subsystem Vendor IDs are assigned by the PCI SIG.

The PCI SIG also defines a ID pair of of class and subclass for the device type. The class and subclass ID are 8-bit each. As a class dedicated to CAN interfaces was not introduced before PCI Local Bus Specification Revision 2.3 you will find **esd** CAN interfaces utilize three different classes.

<i><b>PCI Class</b></i>	<i><b>PCI Subclass</b></i>	<i><b>Description</b></i>
0x06	0x80	Other Bridge Device / Other System Peripheral
0x02	0x80	Other Network Controller
0x0C	0x09	CAN bus

**Table 3:** PCI device classes assigned to esd CAN interfaces



**Attention:**

Some operating systems and BIOS versions are known to not assign I/O resources to PCI devices with the class 0x06 and subclass 0x80 (Other Bridge Device). Please contact **esd** in these cases for a solution.

## Introduction

The table below gives an overview on the COTS **esd** CAN interfaces, their local bus IDs and device classes.

<b>CAN Interface</b>	<b>Vendor ID</b>	<b>Device ID</b>	<b>Subsystem Vendor ID</b>	<b>Subsystem ID</b>	<b>Class / Subclass</b>
CAN-PCI/200	0x10B5	0x9050	0x12FE	0x0004	0x06 / 0x80
CAN-PCI/266	0x10B5	0x9056	0x12FE	0x0009	0x06 / 0x80
CAN-PCIe/200	0x10B5	0x9056	0x12FE	0x0200	0x0C / 0x09
CAN-PCI104/200	0x10B5	0x9030	0x12FE	0x0501	0x0C / 0x09
CPCI-CAN/200	0x10B5	0x9030	0x12FE	0x010B	0x02 / 0x80
PMC-CAN/266	0x10B5	0x9056	0x12FE	0x000E	0x06 / 0x80
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	0x10B5	0x9050	0x12FE	0x0001	0x06 / 0x80
PMC-CAN/331 (3.3 V)	0x10B5	0x9030	0x12FE	0x000C	0x06 / 0x80
CAN-PCI/360	0x10E3	0x0860	0x12FE	0x0000	0x06 / 0x80
CPCI-CAN/360	0x10E3	0x0860	0x12FE	0x0007	0x06 / 0x80
CAN-PCI/400-2 CAN-PCI/400-4	0x10B5	0x9056	0x12FE	0x0200	0x0C / 0x09
CAN-PCIe/400-2 CAN-PCIe/400-4	0x10B5	0x9056	0x12FE	0x0201	0x0C / 0x09
CPCI-CAN/400-2	0x10B5	0x9056	0x12FE	0x0141	0x0C / 0x09
CPCI-CAN/400-4	0x10B5	0x9056	0x12FE	0x0142	0x0C / 0x09
CPCI-CAN/400-4I	0x10B5	0x9056	0x12FE	0x0143	0x0C / 0x09
CPCI-CAN/400-2-PXI	0x10B5	0x9056	0x12FE	0x0144	0x0C / 0x09
CPCI-CAN/400-4-PXI	0x10B5	0x9056	0x12FE	0x0145	0x0C / 0x09
CPCI-CAN/400-4I-PXI	0x10B5	0x9056	0x12FE	0x0146	0x0C / 0x09
PMC-CAN/400-4	0x10B5	0x9056	0x12FE	0x04C2	0x0C / 0x09
PMC-CAN/400-4I	0x10B5	0x9056	0x12FE	0x04C3	0x0C / 0x09
CAN-PCIe/402, CAN-PCI/402, CPCI-CAN/402, CPCIsreal-CAN/402, CAN-PCIMini/402	0x12FE	0x0402	0x12FE	0x0401 0x0402 0x0403	0x0C / 0x09
CAN-PCI/402-FD, CAN-PCIe/402-FD, CAN-PCIMini/402-FD, CPCIsreal-CAN/402-FD, PMC-CAN/402-FD, XMC-CAN/402-FD CAN-M.2/402-2-FD CAN-PCIeMiniHS/402	0x12FE	0x0402	0x12FE	0x1402 0x1403	0x0C / 0x09
CAN-PCI/405	0x1014	0x0156	0x12FE	0x0008	0x02 / 0x80

**Table 4:** Hardware IDs for CAN PCI, PCIe, PCIe Mini, CPCI, CPCIsreal and PMC bus interfaces

## 1.5.2 USB

An **esd** CAN interface attached to the USB bus can be unambiguously identified by an ID pair which consists of a Vendor ID which is assigned by the USB committee and a vendor specific Device ID. Each ID is 16-bit numerical value.

The table below gives an overview on the COTS **esd** USB CAN interfaces and their unique Ids.

<b>CAN Interface</b>	<b>Vendor ID</b>	<b>Device ID</b>
CAN-USB/Mini	0x0AB4	0x0001
CAN-USB/Micro	0x0AB4	0x0011
CAN-USB/2 CAN-USB/2V2	0x0AB4	0x0010
CAN-USB/3-FD	0x0AB4	0x0014
CAN-AIR/2	0x0AB4	0x0018
CAN-CBX-AIR/2	0x0AB4	0x0019
CAN-CBX-AIR/3	0x0AB4	0x001B
CAN-USB/400	0x0AB4	0x0400
CAN-USB/400-IRIG-B	0x0AB4	0x0401
CAN-USB/400-FD	0x0AB4	0x0402
CAN-USB/400-FD-IRIG-B	0x0AB4	0x0403

**Table 5:** Hardware IDs for CAN USB bus Interfaces

## 1.5.3 Ethernet

An **esd** CAN interface attached to the Ethernet has a unique 48-bit MAC address which consists of a 24-bit vendor ID assigned by the Internet Assigned Number Authority (IANA) and a 24-bit vendor specific part. All Ethernet CAN Gateways use a unique MAC address with the format below:

<b>esd Vendor ID</b>	<b>Device ID</b>
00-02-27	xx-xx-xx

**Table 6:** MAC addresses of esd Ethernet devices

# 1.6 Software Deployment

The necessary (device driver) software is usually shipped on a CD/DVD together with the CAN interface or can be downloaded (Windows/Linux) as an archive from the **esd** website ([www.esd.eu](http://www.esd.eu)). The scope of delivery consists at least of a device driver and the NTCAN-API library. For most supported platforms the necessary files to develop NTCAN based applications (header files, example source, documentation, etc.) is also part of the software package.

A CD/DVD usually contains the (device driver) software and HW/SW manuals for all CAN interfaces supported on a platform, the downloadable software distributions only for a CAN interface family.

## 1.6.1 Windows

If you install from CD/DVD you have all necessary software and can continue reading chapter 2 which describes the installation process for the various versions of the Windows OS.

If you want to download the software from the **esd** website for Windows 10 and later, you need the CAN interface family (see chapter 1.4) and CPU architecture specific (32-/64-bit) device driver package. If you want to develop NTCAN based software on the target PC you also have to download the CAN SDK for Windows before you proceed reading chapter 2.

Device driver for legacy versions of Windows 9x/ME/NT/2000/XP/Vista/7/8.x can also still be downloaded but they are no longer technically supported.

## 1.6.2 Linux / Unix

The (device driver) software for all Unix based systems (except Linux) is shipped exclusively on CD/DVD with the hardware and cannot be downloaded from the **esd** website. So usually you have all necessary software and can continue reading chapter 3 which describes the installation process for the various Unix systems.

Because of the rapid change in the Linux kernel source tree which often leads to problems compiling the device driver for the latest kernel versions the Linux drivers can also be download from the **esd** website. You need a CAN interface family (see chapter 1.4), Linux kernel version (2.4.x/2.6.x/3.x) and CPU architecture specific (32-/64-bit) device driver package which also contains all necessary files to develop NTCAN based software on the target PC before you proceed reading chapter 3.

Some CAN interfaces are supported directly by Linux CAN (aka SocketCAN) which is part of the Linux kernel since version 2.6.25. As the SocketCAN has an individual API **esd** provides a NTCAN wrapper library for SocketCAN so you can use NTCAN based applications with this driver, too (see chapter 3.1.3).

Device driver for other Unix versions but Linux are no longer technically supported.

## 1.6.3 Real-Time Operating Systems

The (device driver) software for all real-time operating systems is shipped exclusively on CD/DVD with the hardware and cannot be downloaded from the **esd** website. So you have all necessary software and can continue reading chapter 4 which describes the installation process for the various supported systems.

## 2 Windows®

This chapter describes the necessary steps to install the (kernel mode) device driver for **esd Plug and Play** (PnP) capable CAN hardware which are connected to a local PnP capable bus (PCI, USB, ...) on the various versions of Microsoft Windows®. There is no device driver support for non PnP **esd** CAN modules on legacy buses (ISA, PC104, Parallel Port,...).

**Note:**

The standard driver installation is based on the *hardware-first* installation mechanism which involves that the installation of the device driver is triggered by plugging in the **esd** CAN hardware to the system. Starting with Windows Vista Microsoft also integrated the tools for an in-box *software-first* installation mechanism which allows a driver preinstallation without the presence of the hardware during this process.

The CAN device drivers for Windows are either based on the **WDM** (Windows Driver Model) or the **WDF** (Windows Driver Foundation). The WDM was introduced with Windows 2000 and WDF was introduced by Microsoft in 2006 as a robust object-based interface for device drivers on top of WDM. The WDF consists of a Kernel Mode Driver Framework (**KMDF**) and User Mode Driver Framework (UMDF). All WDF based CAN device driver are KMDF driver.

The KMDF comes as a library which is either already part of Windows or is installed together with the CAN device driver once per system with the help of a WDF co-installer. The WDF based CAN driver currently use the KMDF library version 1.9 which is part of Windows since Windows 7.

The installation procedure for WDM and WDF based driver described for the various Windows versions in chapter 2.1 to 2.9.4 is identical.

**Note:**

The KMDF library version 1.9 does not support Windows 2000 so a **WDF** based CAN device driver requires Windows XP or later to run. Please refer to the WDF driver release notes for further differences between these two driver types.

**Attention:**

The **WDM** based device drivers for the C400, C402 and C405 family do not support a change to a low power state (standby or hibernation) and you have to disable this on your Windows system. Please refer to chapter 2.2.2 for details.

The tables on the next pages give you an overview on the various files which are part of a WDM/WDF driver for 32- or 64-bit Windows.

A **WDM** based kernel mode device driver package for 32-bit Windows contains the following files where <drvname> is the device family specific driver name following driver naming convention I (see chapter 1.4).

<b>Filename</b>	<b>Description</b>
x86/<drvname>.sys	The 32-bit <b>WDM</b> based device driver.
x86/ntcan.dll	The 32-bit NTCAN library
x86/canui32.dll	The 32-bit <i>Device Manager</i> property sheet extension
x86/calcan32.dll	The 32-bit protocol helper library
<drvname>.inf	The driver's INF file.
<drvname>.cat	The driver's catalogue file with cryptographic hashes.

**Table 7:** Files of 32-bit WDM based driver package

A **WDM** based kernel mode device driver package for 64-bit Windows contains the following files where <drvname> is the device family specific driver name following driver naming convention I (see chapter 1.4).

<b>Filename</b>	<b>Description</b>
amd64/<drvname>a.sys	The 64-bit <b>WDM</b> based device driver.
amd64/ntcan64.dll	The 64-bit NTCAN library (Renamed to ntcan.dll during install)
amd64/canui64.dll	The 64-bit <i>Device Manager</i> property sheet extension
amd64/calcan64.dll	The 64-bit protocol helper library
x86/ntcan.dll	The 32-bit NTCAN library (for WoW64)
x86/calcan32.dll	The 32-bit protocol helper library (for WoW64)
<drvname>a.inf	The driver's INF file.
<drvname>a.cat	The driver's catalogue file with cryptographic hashes.

**Table 8:** Files of 64-bit WDM based driver package



A **WDF** based device driver package for 32-bit Windows contains the following files where **<drvname>** is the device family specific driver name following driver naming convention I (see chapter 1.4).

<b>Filename</b>	<b>Description</b>
<b>x86/&lt;drvname&gt;k.sys</b>	The 32-bit <b>WDF (KMDF)</b> based device driver.
<b>x86/WdfCoInstallerMMmmm.dll</b>	The 32-bit WDF co-installer.*
<b>x86/ntcan.dll</b>	The 32-bit NTCAN library
<b>x86/canui32.dll</b>	The 32-bit <i>Device Manager</i> property sheet extension
<b>x86/calcan32.dll</b>	The 32-bit protocol helper library
<b>&lt;drvname&gt;k.inf</b>	The driver's INF file.
<b>&lt;drvname&gt;k.cat</b>	The driver's catalogue file with cryptographic hashes.

**Table 9:** Files of 32-bit WDF based driver package

A **WDF** based device driver package for 64-bit Windows contains the following files where **<drvname>** is the device family specific driver name following driver naming convention I (see chapter 1.4).

<b>Filename</b>	<b>Description</b>
<b>amd64/&lt;drvname&gt;ak.sys</b>	The 64-bit <b>WDF (KMDF)</b> based device driver.
<b>amd64/WdfCoInstallerMMmmm.dll</b>	The 64-bit WDF co-installer.*
<b>amd64/ntcan64.dll</b>	The 64-bit NTCAN library (Renamed to ntcan.dll during install)
<b>amd64/canui64.dll</b>	The 64-bit <i>Device Manager</i> property sheet extension
<b>amd64/calcan64.dll</b>	The 64-bit protocol helper library
<b>x86/ntcan.dll</b>	The 32-bit NTCAN library (for WoW64)
<b>x86/calcan32.dll</b>	The 32-bit protocol helper library (for WoW64)
<b>&lt;drvname&gt;ak.inf</b>	The driver's INF file.
<b>&lt;drvname&gt;ak.cat</b>	The driver's catalogue file with cryptographic hashes.

**Table 10:** Files of 64-bit WDF based driver package

\*MM is the major version and mmm is the minor version number of the WDF co-installer. For the WDF version 1.09 this would result in the filename WdfCoInstaller01009.dll.

Chapter 2.2 covers the device driver configuration with the Windows *Device Manager* for Windows 2000 and later.

The mechanisms for a device driver preinstallation introduced by Microsoft with Windows Vista are described in chapter 2.3 and chapter 2.5 covers the post installation aspects of device driver update, device driver rollback and device driver uninstall.

The installation and configuration of the EtherCAN/2 driver software (for all supported versions of Windows) is described in the separate chapter 2.6 as this CAN interface is using a user mode device driver and a hardware specific configuration tool.

Development of NTCAN based applications (on any hardware) requires the installation of the CAN Software Development Kit (SDK) for Windows which comes as separate package. The installation of this package is described for in chapter 2.7.

Chapter 2.8 contains information on device driver and software signing.

The final chapter 2.9 is intended as reference for the installation and configuration of the device driver for the legacy versions of Windows (9x/ME/NT/2000) which are no longer actively supported by Microsoft as well as **esd**.



**Note:**

This chapter sometimes refers to Microsoft keyboard short-cuts in combination with the *Windows Key* or *WinKey*. This is the key with the Windows logo shown on it and it is usually found between the *Ctrl* and *Alt* keys on your keyboard. A simultaneous key press with another letter is written in this manual as e.g. **WinKey + R** (which does not mean that you have to capitalize the letter).

## 2.1 Windows 10 / 11

This chapter covers the device driver installation for Windows 10 and Windows 11.

The installation procedure is identical for the 32-bit and the 64-bit version of Windows 10 but different driver binaries are required. On the 64-bit OS versions all libraries to run 32-bit NTCAN based applications in the *WoW64* subsystem are installed automatically.

**Attention!**

The device driver for the CAN hardware of the C400, C402 and C405 family currently do not support the *Hybrid Shutdown* mechanism which is the default for these Windows versions. Please refer to chapter 2.2.2 to disable it before you install the device driver.

**Note:**

Please read the current `Release Notes` file that comes with the software!

Please note the drivers delivered on CD are most likely outdated. We rather recommend checking the `esd electronics gmbh` website for newer driver releases and use these ones to circumvent any problems caused by known and fixed issues before they occur.

### 2.1.1 Hardware-First Driver Installation

To initiate the device driver installation process, you have to connect the CAN module to your system. Depending on the *Hot Plugging* capability of the hardware you might have to shut down Windows for this. Please refer to the CAN module specific hardware manual for advises.

**Attention!**

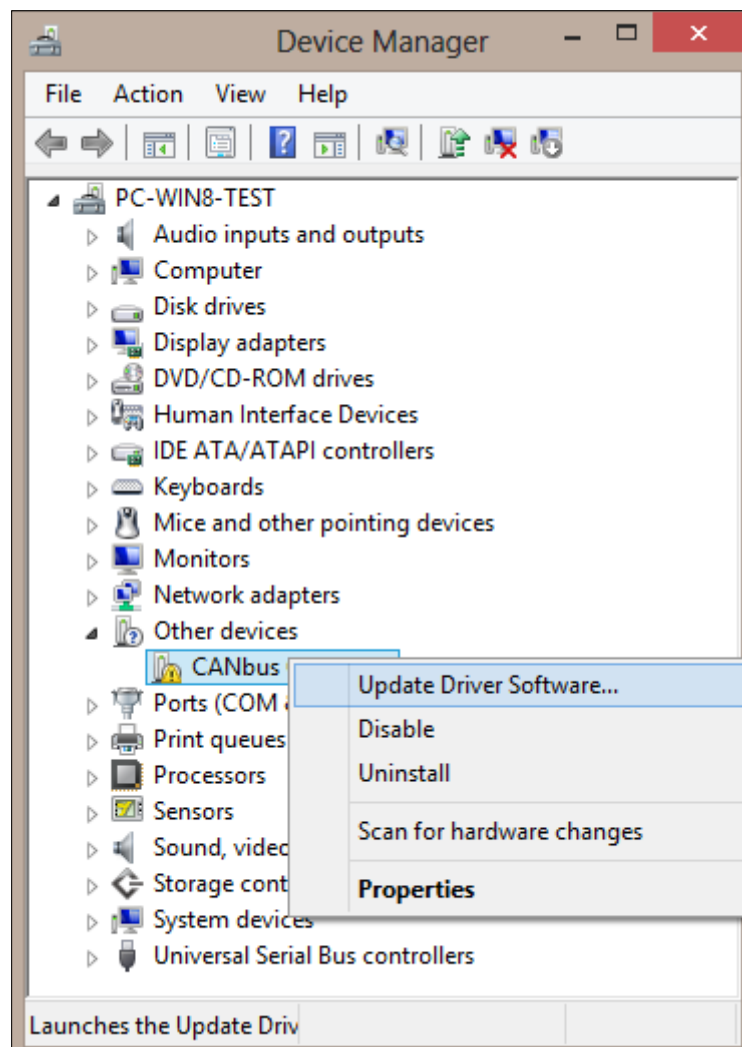
A user which wants to install a device driver must be member of the Administrators group.

Starting with Windows 7 the presence of a new hardware does not automatically start the *Found New Hardware Wizard* to locate and install a driver for the new device with user interaction as in previous versions of Windows. To initiate the interactive device driver installation you now have to open the *Device Manager*. One of the fastest way to do this is to press **WinKey + Pause/Break** and to select *Device Manager* or your local translation of this tool in the newly opened dialogue box.

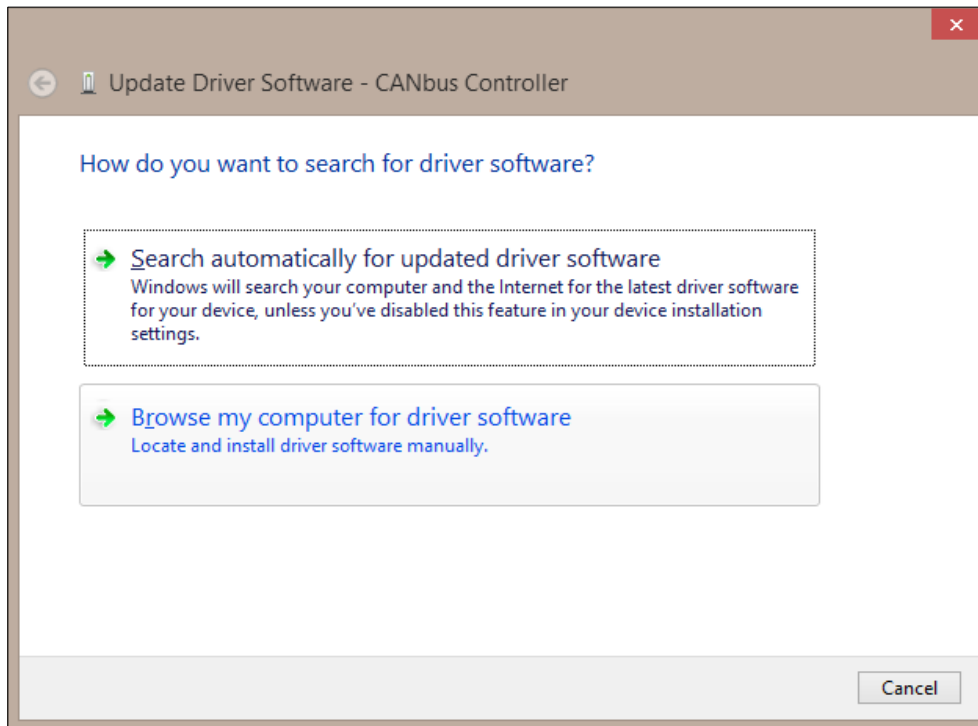


In the *Device Manager* windows there will be a device under *Other Devices* with a yellow exclamation point next to the icon to indicate that there is no device driver installed yet. The text next to the device will depend on the CAN module attached.

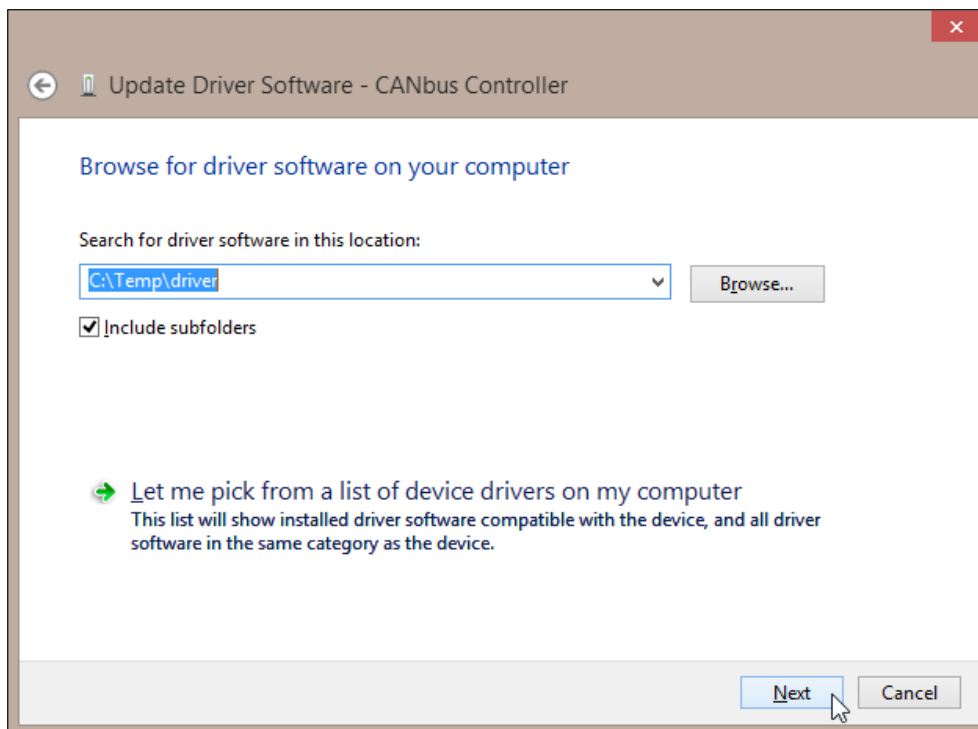
Right click on the device to bring up the context menu as shown above and select the menu item **Update Driver Software...** which opens the following dialogue box.



Right click on the device to bring up the context menu as shown above and select the menu item **Update Driver Software...** which opens the following dialogue box.

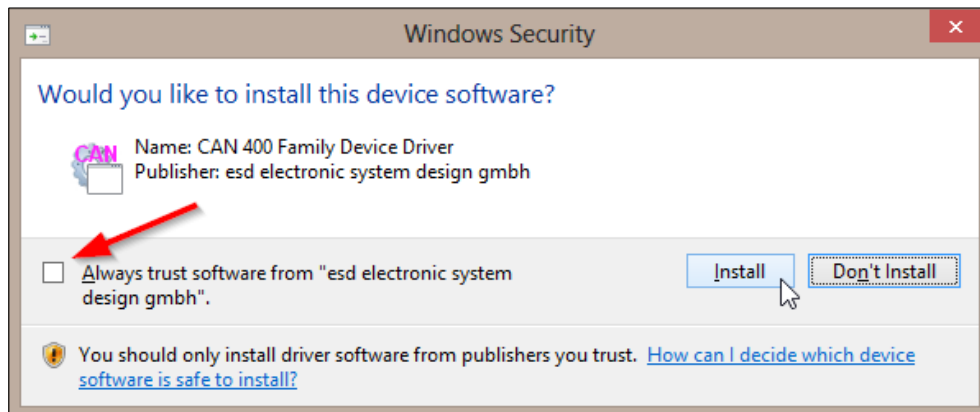


Select the second option to “Browse the computer for driver software” and the following dialogue box will appear.



Press the **Browse...** button to define the location of the driver files. This might either be the drive letter of your optical driver if you want to use the CD which accompanied the delivery of your CAN module or is the location on your hard disk where you have extracted a driver archive downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu)). If the driver files are in a sub directory of the configured path do not forget to check the “Include subfolders” option in the dialogue before you press the **Next** button to start copying the files to your system which may take some time.

During driver installation you will see a security message similar to in the dialogue below.



All CAN device drivers are digitally signed to give you as end user who is installing this software the possibility to verify that **esd** is really the publisher of this driver package and that the binaries are not tampered. Please refer to chapter 2.8 for more details about *Digital Signatures*.

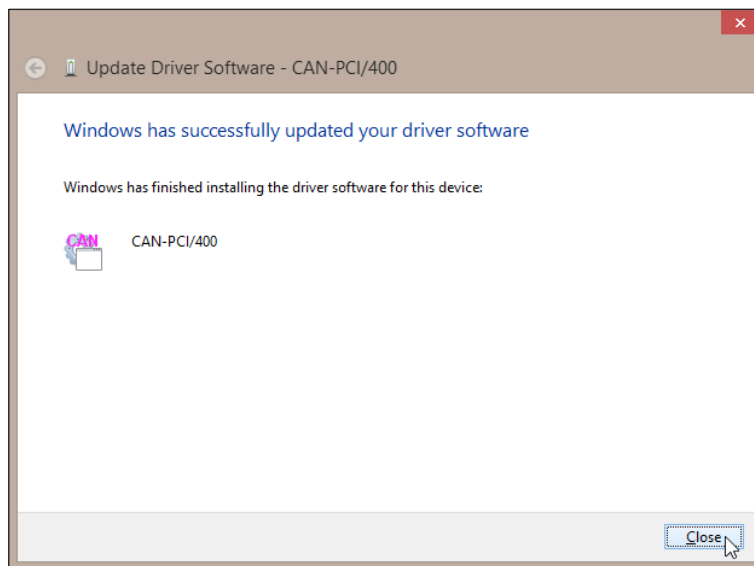


**Note:**

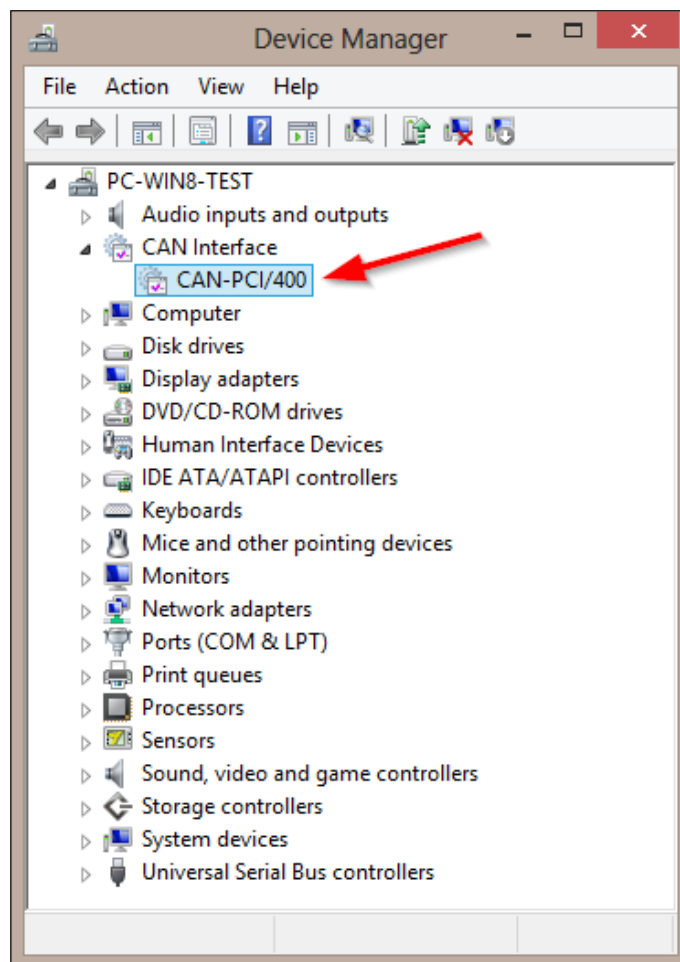
If you activate the check box “Always trust software from *esd electronic system design gmbh*” you will not have to confirm this dialogue in the future during the installation of another digitally signed driver for an **esd** device.

Press the **Install** button to continue.

When the installation is finished a completion dialogue as above is displayed and the driver is now started automatically with every Windows start-up. The displayed device name depends on the CAN module. Press the **Close** button to complete the installation.



If you return to the *Device Manager* window you will see that the CAN module is now listed below the new device class "CAN Interface".





If you have installed several **esd** CAN modules attached to a local bus of your system you will find all of them here. By double clicking the device you will open the *Properties* dialogue where you can configure the device specific options described in chapter 2.2 via the *Settings* tab.

**Note:**

If you just want to run NTCAN based application on the system you are done.

If you intend to develop NTCAN based applications on this system you also must install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

## 2.1.2 Software-First Driver Installation

The process for a software-first driver installation is similar on all versions of Windows since Vista and is covered in chapter 2.3.

## 2.1.3 Driver Lifecycle Management

The process of updating, rolling back or uninstalling a device driver package is very similar on all Windows versions and covered in chapter 2.5.

## 2.2 Configuration

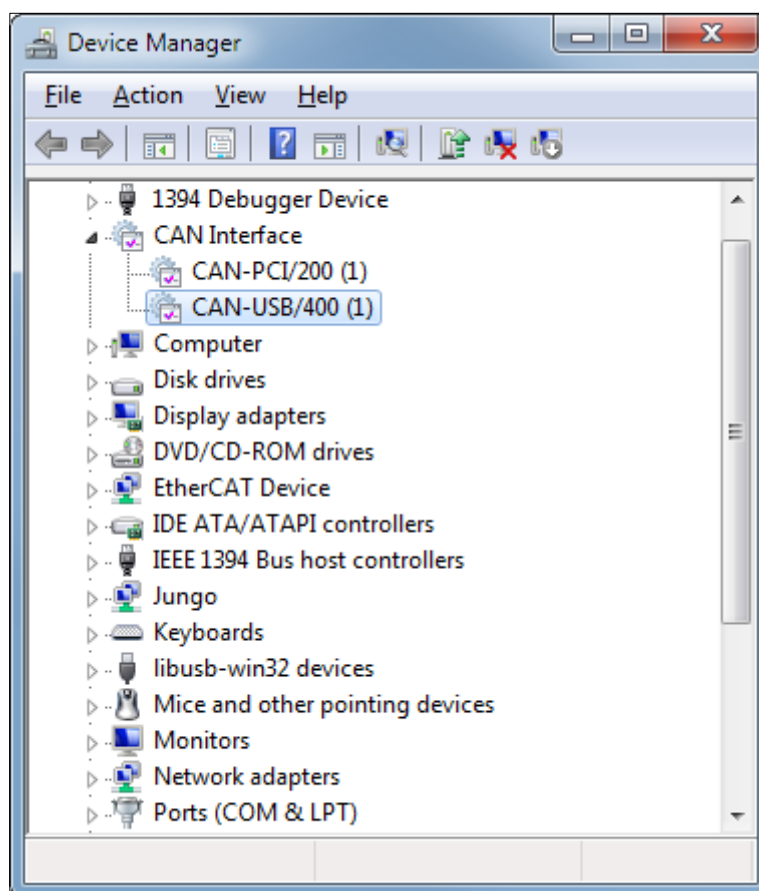
This chapter contains a description of configuration options of the device driver and Windows itself.

### 2.2.1 Device Driver

#### 2.2.1.1 Standard Settings

This chapter covers the CAN device driver configuration for Windows 2000 and later versions.

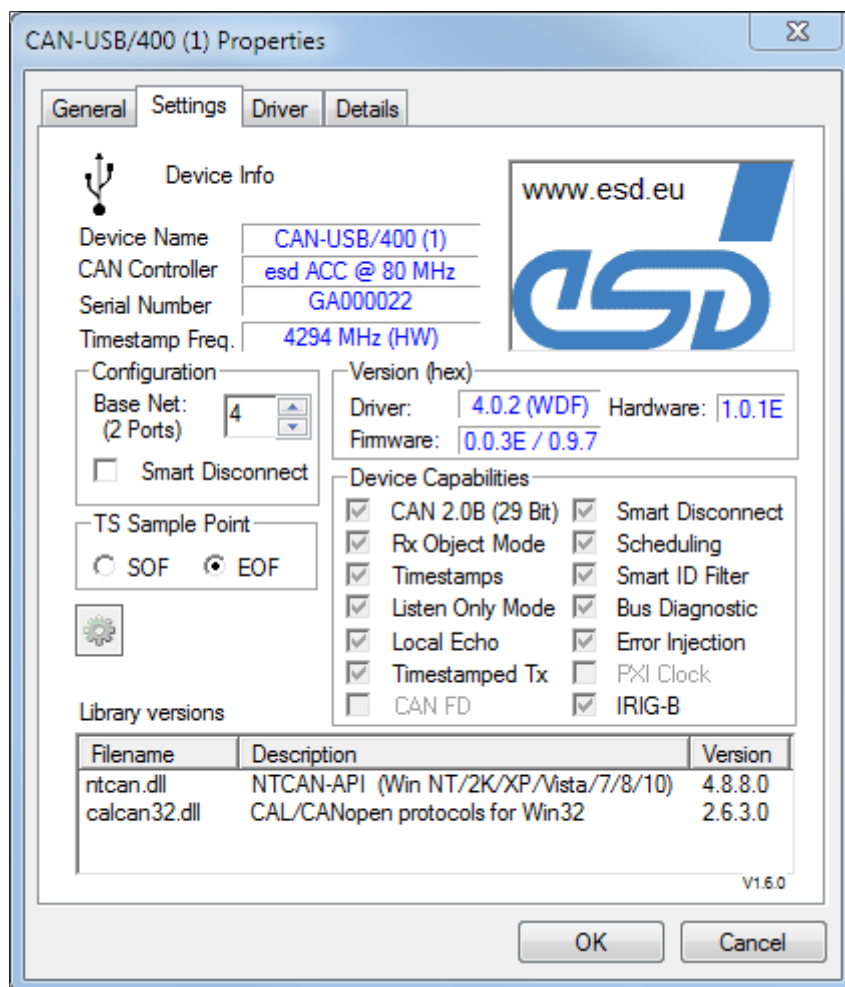
To configure several CAN driver related settings or to check the version of the installed components you must open the Windows Device Manager. All device drivers for **esd** CAN modules are installed as a **CAN Interface** class as shown below.



To change the device driver configuration parameter, double-click the device instance to open the *Properties* dialogue of the device and select the *Settings* tab.

The settings dialogue (Rev. 1.6.x or later) contains the following read only information:

- Device name with CAN controller type and CAN controller frequency.
- Device driver revision and type (WDM / WDF), the firmware revision(s) (if applicable) and the hardware revision.
- The serial number (if serial number access is supported by the hardware).
- The timestamp frequency (if timestamping is supported) and the information if this is a hardware (HW) or software (SW) timestamp.
- The CAN interface and driver related device capabilities.
- The version of the libraries which are installed together with the driver.



Nearly all elements in the configuration dialogues will present a context sensitive (English) help text if you move the mouse on them and wait for a while.

The text 'Device Info' at the top the dialogue may be replaced by a problem notification with increased font size to ease troubleshooting.

If you click on the **esd** logo your default web browser is launched with the URL of the **esd** website where you can check if device driver updates for your CAN hardware are available for download.

The dialogue also allows to configure several driver parameters on a per device basis.

**Attention!**

Parameter changes are applied with the next start of the device driver and not immediately after the dialogue is closed.

### Base Net

Via *Base Net* a logical net number is assigned to the CAN module which is used by NTCAN to distinguish between several physical CAN ports. The number of available physical ports is indicated here. If a CAN interface has more than one physical CAN port, the logical net number entered in *Base Net* is assigned to the first physical port and further ports will be assigned consecutive increasing net numbers. The default value for the first instance of a hardware is always 0.

**Attention!**

If there is more than one CAN module in the system, the user has to make sure that the logical net numbers which are assigned to the physical ports do not overlap!

### Smart Disconnect

The *Smart Disconnect Feature* to disable a port after the last handle is closed can be enabled or disabled, if supported by the CAN hardware. The default after installation is disabled.

### Timestamp sample point

An ESDACC based CAN hardware (C400,C402 and U400 CAN device family) allows to configure if the timestamp of a CAN frame is captured at the Start of Frame (SOF) or at the End of Frame (EOF) which is the default after driver installation. For other CAN device families this configuration option is not available.

### CPU Affinity

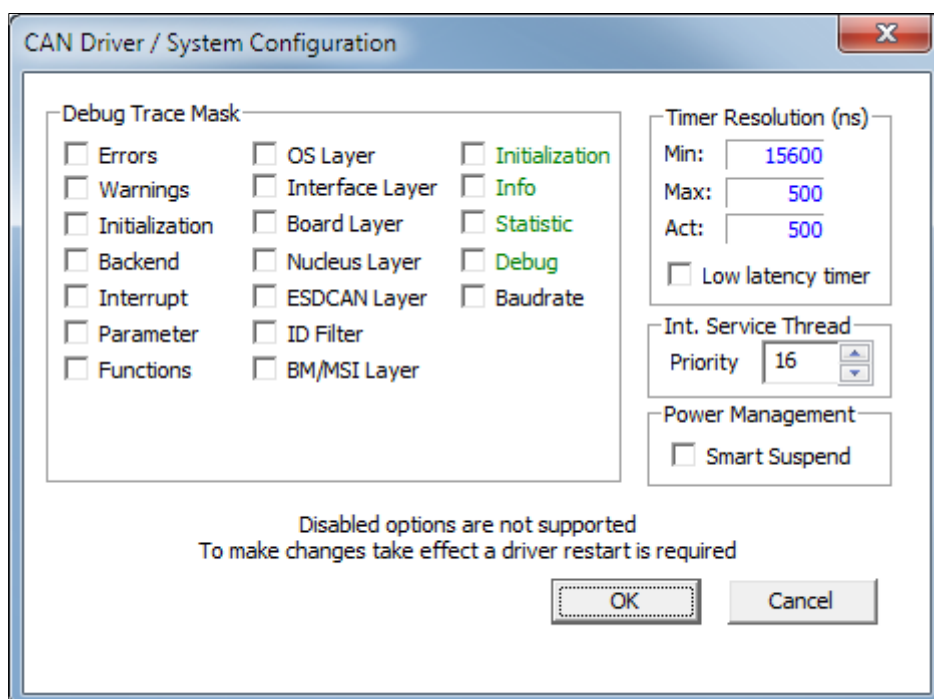
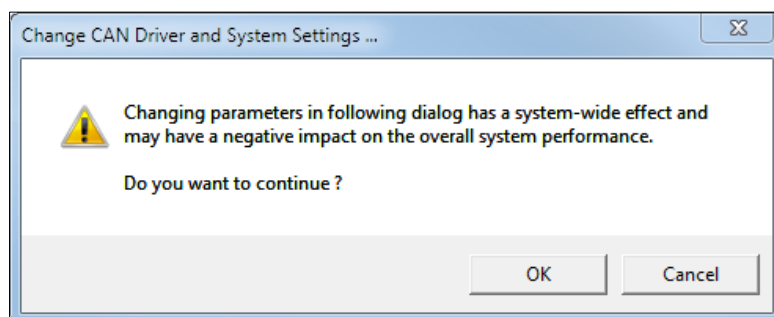
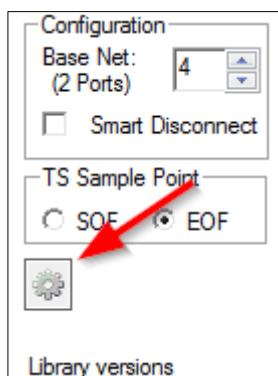
Earlier versions of this dialogue allowed to configure the *CPU Affinity* of the devices interrupt handler. This option has been removed (from the dialogue and the driver) as modifying the CPU affinity mask within the driver caused unwanted side effects if the interrupt was shared with another device. Please refer to chapter 2.2.2.2 for this topic.

### Smart Suspend

Earlier versions of this dialogue allowed to configure the *Smart Suspend* option here which is now moved into the *Expert Settings* dialogue described in the next chapter.

### 2.2.1.2 Expert Settings

Via the cogwheel icon you can reach an expert settings dialogue after you have confirmed the warning that changing parameters in this dialogue may have (negative) system-wide effects.



#### Debug Trace Mask

For all device driver with a version greater than 3.10.x you can enable a trace mask which causes the driver to send trace messages via the Windows kernel debugger API. For the release build of the device driver only the options marked green have an effect. From Microsoft TechNet you can download the tool DebugView to capture these messages without setting up a kernel debugger.

## Timer Resolution

In Windows the clock interrupt frequency can be changed within certain limits. The dialogue shows the actual active clock interrupt resolution in nanoseconds as well as the system specific minimum and maximum value. A higher value for the clock interrupt frequency decrease in some cases the I/O latency and improves the granularity of configured timeouts.

If you enable the **Low Latency Timer** option the device driver will configure the interrupt resolution to 1 ms with the next driver start.



### Attention!

Note that the result of changing the clock interrupt frequency is system-wide and can also have a severely negative effect on system performance. Also note that higher clock interrupt frequencies can shorten a system's battery life.

## Service Thread Priority

All device driver V4.x.x and later process CAN messages on passive level instead on DPC level to reduce the overall system latency on multicore CPUs. The thread priority of the passive level worker thread which handles the CAN messages can be configured here.

## Smart Suspend

If the device driver supports this feature a change into a lower power state (Sleep/Hibernate) is rejected by the device driver as long as there is an application with an open CAN handle.

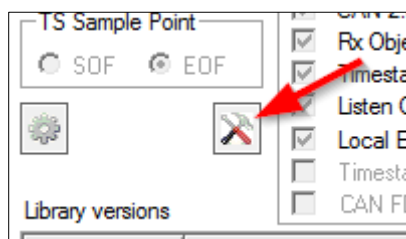


### Note:

A device driver is only able on Windows 2000/XP to prevent a state into a low power state completely. On Windows Vista and later a device driver cannot prevent such a change if it is explicitly requested by a user but it can prevent changes if they result on configurations in the powerplan.

### 2.2.1.3 Device Specific Settings

If a device specific configuration tool is available for a CAN hardware (CAN-AIR/2, CBX-AIR/2, CBX-AIR/3) the common *Settings* dialogue will show a tools icon which opens this hardware specific configuration tool if clicked.

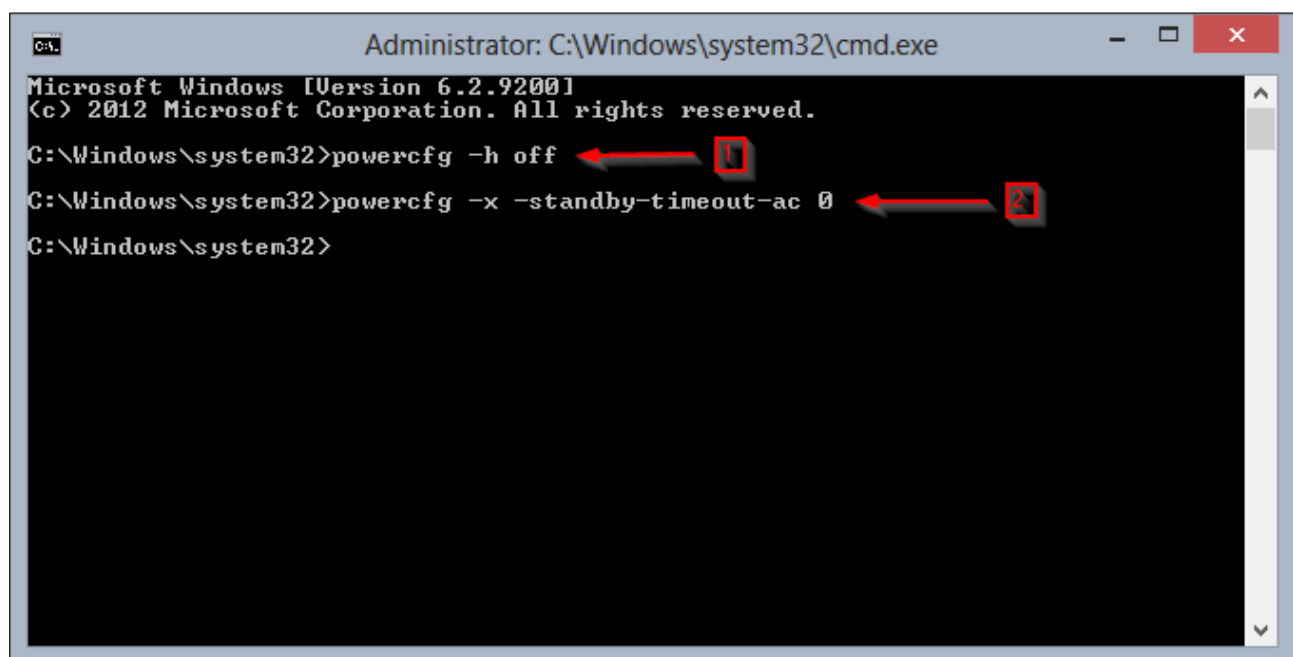


## 2.2.2 System

### 2.2.2.1 Power Management

The WDM device drivers (3.x.y) for the C400, C402 and C405 family (see 1.4) of CAN devices do not support a change to a low power state which includes the support of the *Hybrid Shutdown* feature introduced with Windows 8. With *Hybrid Shutdown* (which is enabled by default) the states of drivers and services are saved into the hiberfile on shutdown for a faster reboot.

To prevent problems using hardware from this driver families on Windows you have to make sure that the change to a low power state is disabled on your system. This can be achieved via various power management related dialogues in Windows but the easiest method is to use the *powercfg* command line utility which was introduced with Windows XP SP2. Powercfg must be run from an elevated command prompt and requires administrator rights.



Use the command `powercfg -h off` (1) to disable hibernation and implicitly the *Hybrid Shutdown* feature of Windows 8. If you want to restore hibernation you have to use 'on' instead of 'off'.

Use the command `powercfg -x -standby-timeout-ac 0` (2) to disable Windows standby mode. If you want to restore the standby behaviour use a positive value in minutes instead of 0 as parameter. On Windows XP you have to write `-change` instead of the abbreviation `-x`.

### 2.2.2.2 Interrupt Affinity

The interrupt affinity is the set of processors/cores that should service an interrupt in a multi-processor/multi-core architecture. After the device driver installation each PCI / PCIe / CPCI / PMC based CAN device has the default interrupt affinity policy that any processor/core can handle its interrupt which usually need not to be changed.

In cases you want to assign the affinity to dedicated (set) of processors or cores you should use the *intfiltr.exe* tool which is part of the *Windows 2003 Resource Kit* for Windows 2000/XP or the *Microsoft Interrupt-Affinity Policy Tool* for Windows Vista and later.

## 2.3 Device Driver Preinstallation

This chapter describes the process of (pre-)installing a device driver before the hardware is present (*software-first* installation) for Windows Vista and later so the device driver for **esd** CAN hardware is installed as soon as the PnP manager detects a (new) instance of the hardware without further user interaction like devices which are supported by Windows in-box drivers.

### 2.3.1 Driver Staging

Starting with Windows Vista, Microsoft introduced a repository for device drivers which is called the *Driver Store* and split up the device installation process into two steps:

- **Driver Staging:** The process of adding driver packages to the *Driver Store*.
- **Driver Installation:** The process of installing drivers from the *Driver Store*.

During a hardware-first installation the PnP Manager performs the step of driver staging implicitly so a device driver is always installed from the driver store but the user experience remains similar to the Windows versions before Vista. You can find a detailed description of the steps during driver installation in this [Microsoft TechNet article](#).

The central repository which even keeps several versions of a device driver offers some advantages compared to earlier versions of Windows:

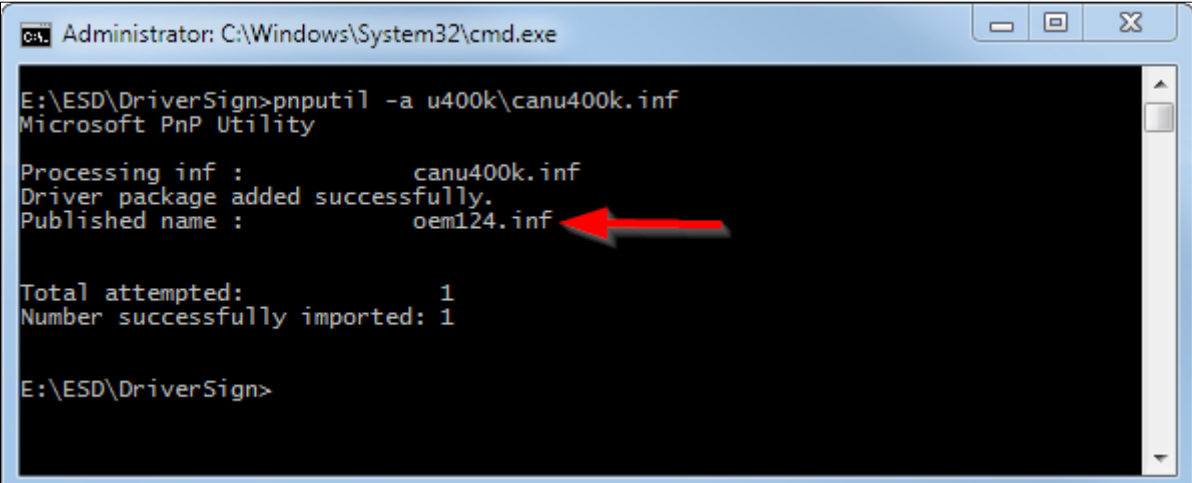
- Driver Repair or Re-installation no longer requires the source media.
- Managing a driver rollback (see chapter 2.5.2) is easily possible.
- The driver store can be preloaded with OEM drivers and the device hardware does not need to be present during this staging process.

The option to stage a device driver is the precondition for a software-first installation process. The in-box (console) staging tool for Windows is **PnPUTil** which usage requires administrative privileges on the system. During driver staging, the driver files are verified and copied into the driver store. They are not installed on the system until the device hardware is detected by the PnP manager.

The staging process is triggered with the command

```
pnputil -a <Path\drvname.inf>
```

as shown in the picture below for a CAN-USB/400 device driver. The directory with the INF file has to contain all driver files in the hierarchy of the distribution media.



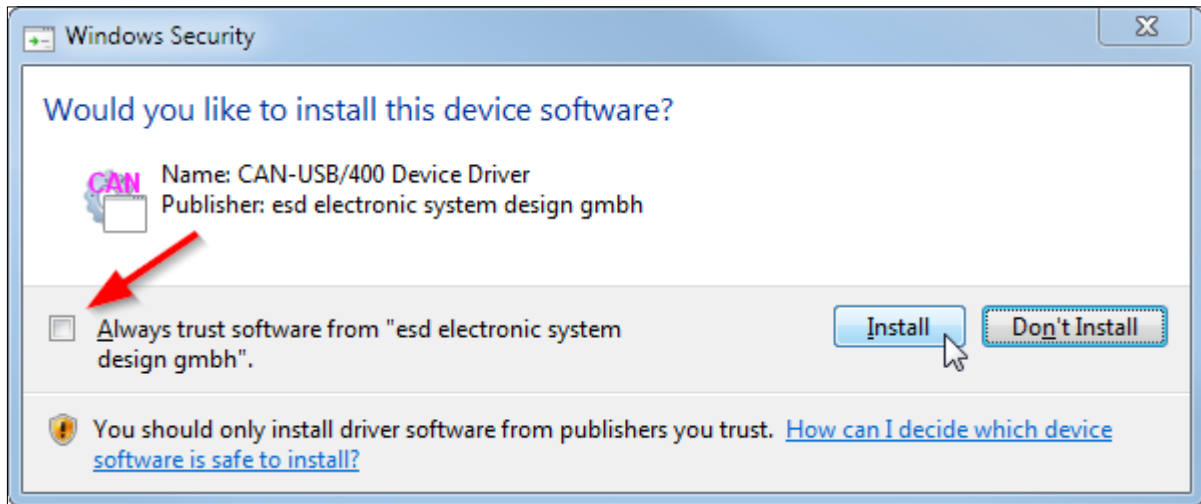
```
Administrator: C:\Windows\System32\cmd.exe
E:\ESD\DriverSign>pnputil -a u400k\canu400k.inf
Microsoft PnP Utility

Processing inf :          canu400k.inf
Driver package added successfully.
Published name :          oem124.inf
Total attempted:          1
Number successfully imported: 1

E:\ESD\DriverSign>
```



During the staging process you have to complete the *Windows Security* dialogue with the **Install** button in the same way you have to do it for hardware-first installation process. Optionally you may choose to accept **esd** as trustworthy software publisher so this dialogue will not appear in further device driver staging or installation tasks (refer to chapter 2.8.2 for more information about digital signatures).



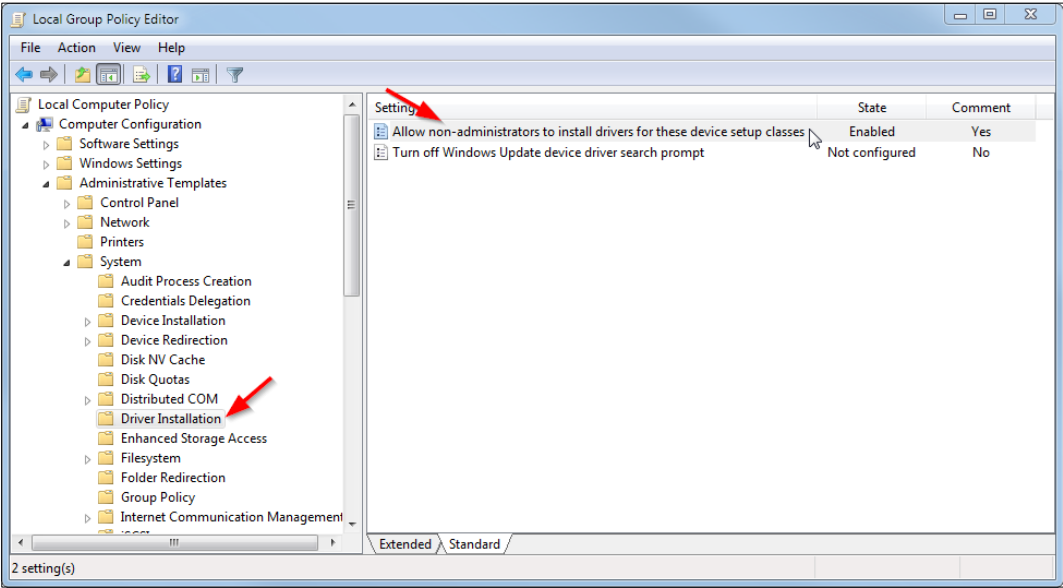
Successfully staged drivers can also be removed from the driver store. For this purpose you have to note the *Published Name* which is indicated during the staging process in the console window. This name consists of the common prefix '**oem**' followed by a digital number and the suffix '**.inf**'. In the example above the name is **oem124.inf** and you can remove this staged driver with the command:

```
pnputil -d oem124.inf
```

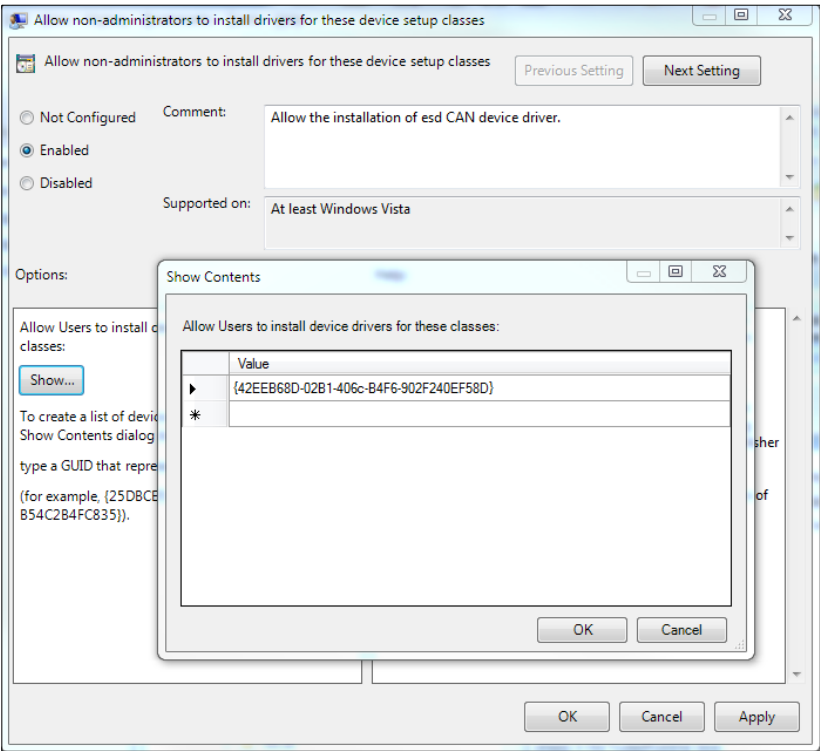
## 2.3.2 Driver Installation for Non-Administrators

Without further administrative action the installation of a device driver, which has been staged as described in the previous chapter, is only possible for users which belong to the local Administrators group.

This [Microsoft TechNet article](#) describes the required steps to configure a policy with the *Local Group Policy Editor* to allow users which do not belong to the local Administrators group to install previously staged device drivers. If you follow the steps of this article you have to double click the **“Allow non-administrators to install drivers for these device setup class”** rule as shown in the following picture:



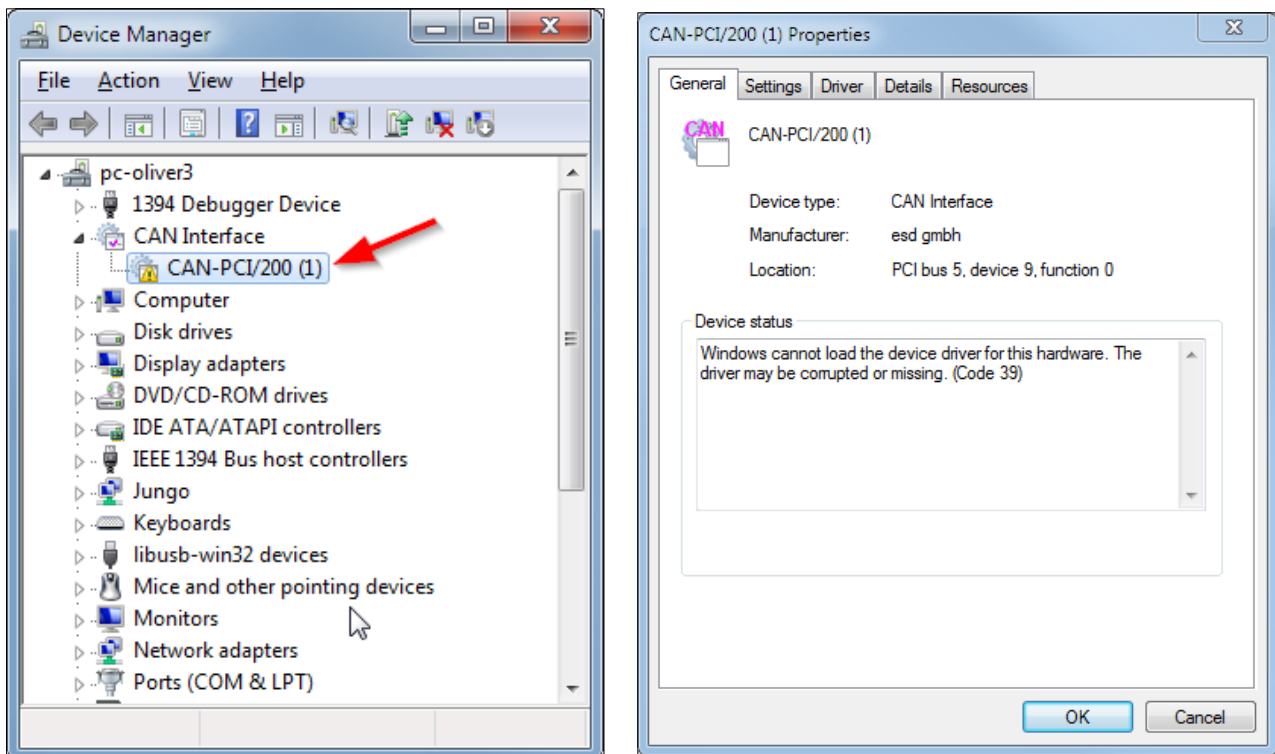
The device setup class for **esd** CAN devices is **{42EEB68D-02B1-406c-B4F6-902F240EF58D}** which has to be added in the dialogue which is opened if you click the “**Show**” button.



## 2.4 Troubleshooting Driver Installation

If the installation of a device driver fails, the Windows *Device Manager* is the central starting point for troubleshooting the problem. One of the fastest ways to open the *Device Manager* (which works on any supported Windows version) is pressing the key combination **WinKey + Pause/Break**. This opens the system settings dialogue where you can open the *Device Manager* with a further mouse click.

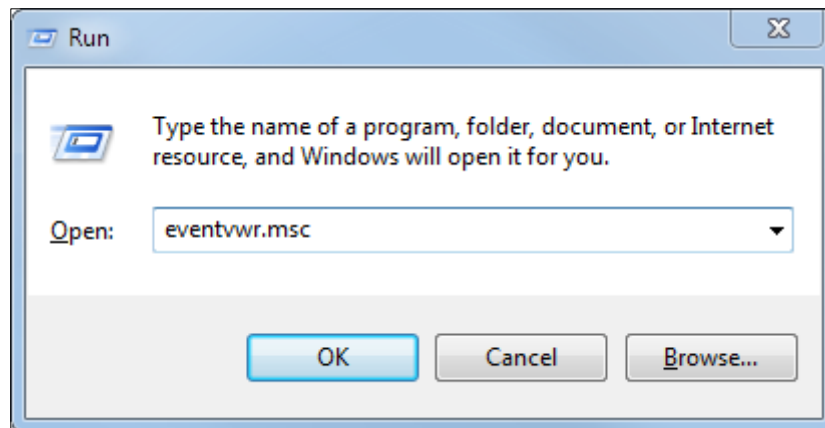
In case of a problem the CAN hardware is marked with a yellow exclamation point and you can double-click on the device to view more information about the problem.



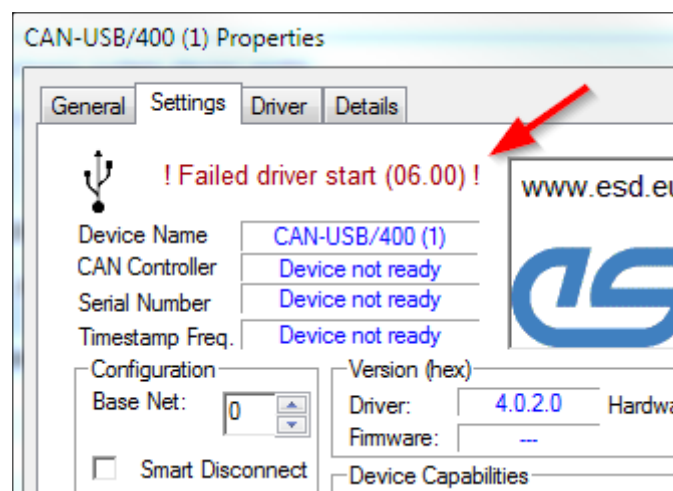
Follow this link for a list of possible error codes and reasons. The next two chapters deal with the two most common error situations:

### 2.4.1 Error Code 31

This is an indication that Windows starts the device driver but something went wrong during the startup process. In this case in many error situations the CAN device driver will store additional information in the Windows *System Event Log*. You can open the Windows Event Viewer by pressing the key combination **WinKey + R** and starting *eventvwr.msc*.



WDM based device driver from the driver package 2.6.8 and later as well as all WDF based device driver can indicate the error reason in the upper left corner of the device's *Settings* tab.



The error reason is indicated as a CAN device driver specific major and minor error code separated by a colon. If you double click on this number the Windows *Event Viewer* will be opened with a filter configured for this device driver.

The following table contains a list of common major error codes which are reported during device initialization.

<b>Error Code</b>	<b>Description</b>
1	Failed to map physical address.
2	Failed to attach the interrupt handler.
3	Failed to verify the CAN hardware.
6	Failed to create the logical base net as it is already in use by another CAN device.
7	Out of resources.
8	No interrupt assigned to the device.
11	Firmware and device driver are incompatible. Update firmware or driver.
12	Hardware and device driver are incompatible. Update device driver.
13	Unrecoverable PCI bridge bug detected.
14	Failed to write into the PCI configuration space.
16	This is an engineering release of a debug driver without CAN I/O functionality.
19	Internal error during initialization.
20	Failed to attach to a lower level device driver.
21	Failed to register CAN interface class.
22	Failed to configure an USB device.
25	Bootloader update required (CAN-PCI/405).
28	Error returning from standby or hibernate.
29	Failed to create a logical net as it is already in use by another CAN device.

**Table 11:** Windows Device Driver Installation Error Codes

### 2.4.2 Error Code 39

This is an indication that the device driver is missing, corrupted or especially on 64-bit system the validation of the digital signature has failed (see chapter 2.8.2).

If re-installing the device driver does not solve the problem you can check a plaint text log file where Windows Vista and later versions of Windows store information about device driver installation process especially if a signing problem exists. You will find this log file in

**%SystemRoot%\inf\setupapi.dev.log**

### 2.4.3 Error Code 52

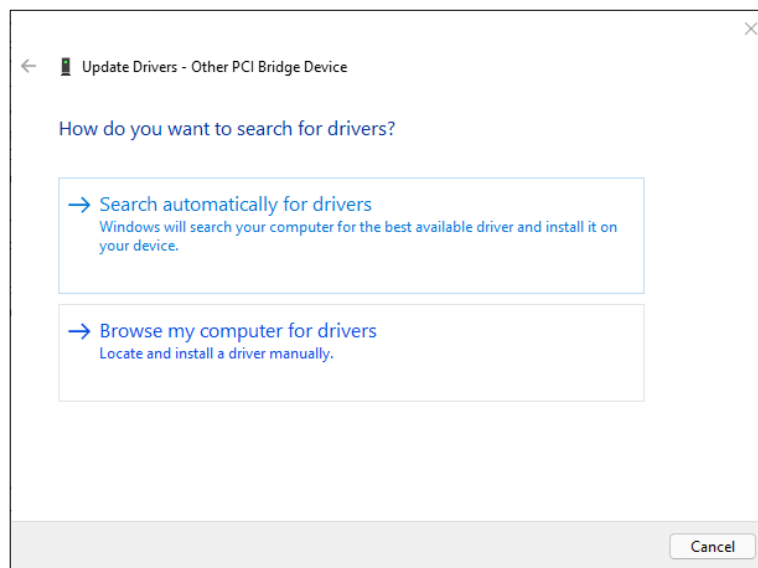
This is an indication that the device driver may be unsigned or corrupted. (see chapter 2.8.2).

## 2.4.4 Best Driver already Installed

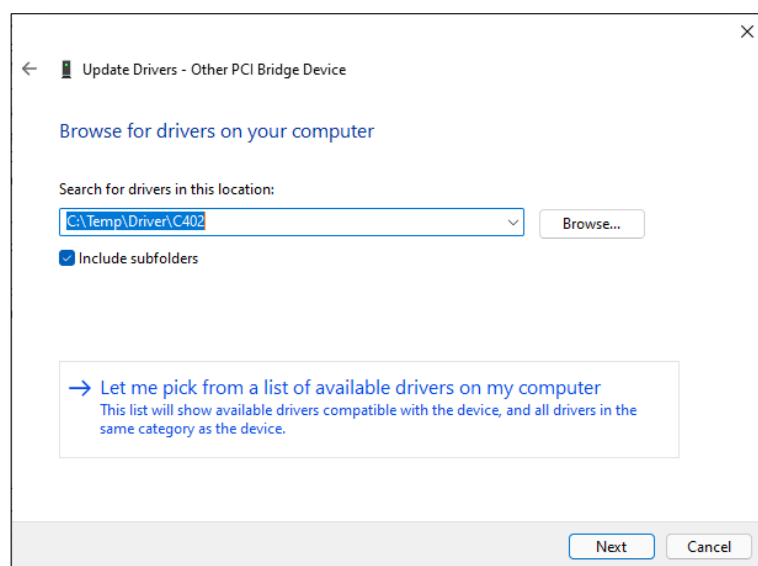
Windows implements an internal mechanism of driver ranking. If there is already a driver installed which is ranked higher than the one you want to install Windows will cancel the installation process with the message that the “Best Driver is already Installed”. If the driver has to be installed anyway, there are two possibilities to overcome this situation:

1. Uninstall existing drivers until Windows reports that there is no driver available for this device (see chapter 2.5.3).
2. Enforce the installation of the driver installation as follows:

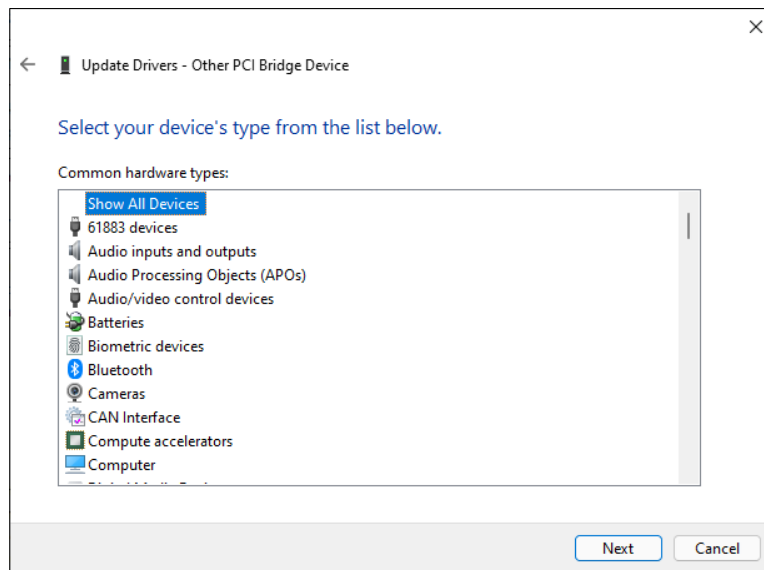
Open the Device Manager, select “Update Driver” in the context menu of the device (see chapter 2.5.1) and choose “Browse my computer for drivers” in the dialogue which is opened by Windows.



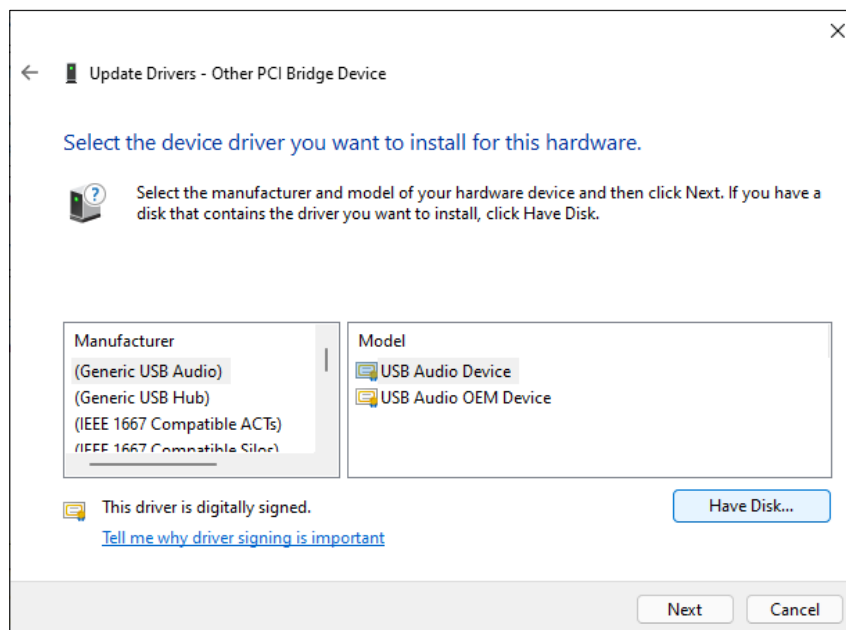
Choose “Let me pick...” in the following dialogue.



Select “All available devices” in the following dialogue:



Choose “Have Disk...”, change to the location of your driver installation package and start installing the driver.



## 2.5 Device Driver Lifecycle Management

This chapter describes the process of updating a device driver to a newer version, rolling back a device driver to a previous version or uninstall a device driver. This process is similar for all Windows versions.

**Attention!**

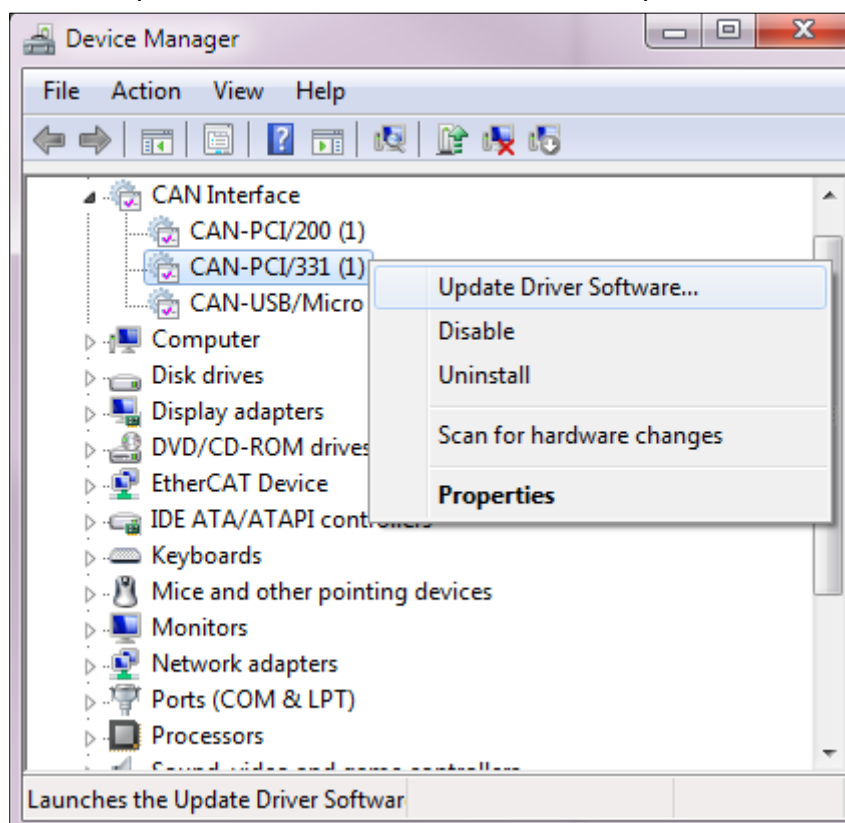
Close all applications which are using the NTCAN-API before you start updating or rolling back a device driver.

**Note:**

As most device driver packages are family device driver which support more than one CAN device type (see chapter 1.4) all devices covered by this driver are affected by any change.

### 2.5.1 Driver Update

To update the device driver to a newer version open the *Device Manager* and select in the context menu of the CAN device *Update Driver Software* as shown in the picture below for Windows 7.



Follow the steps of the wizard which will guide you through the device driver update process which is like the initial installation of a device driver.

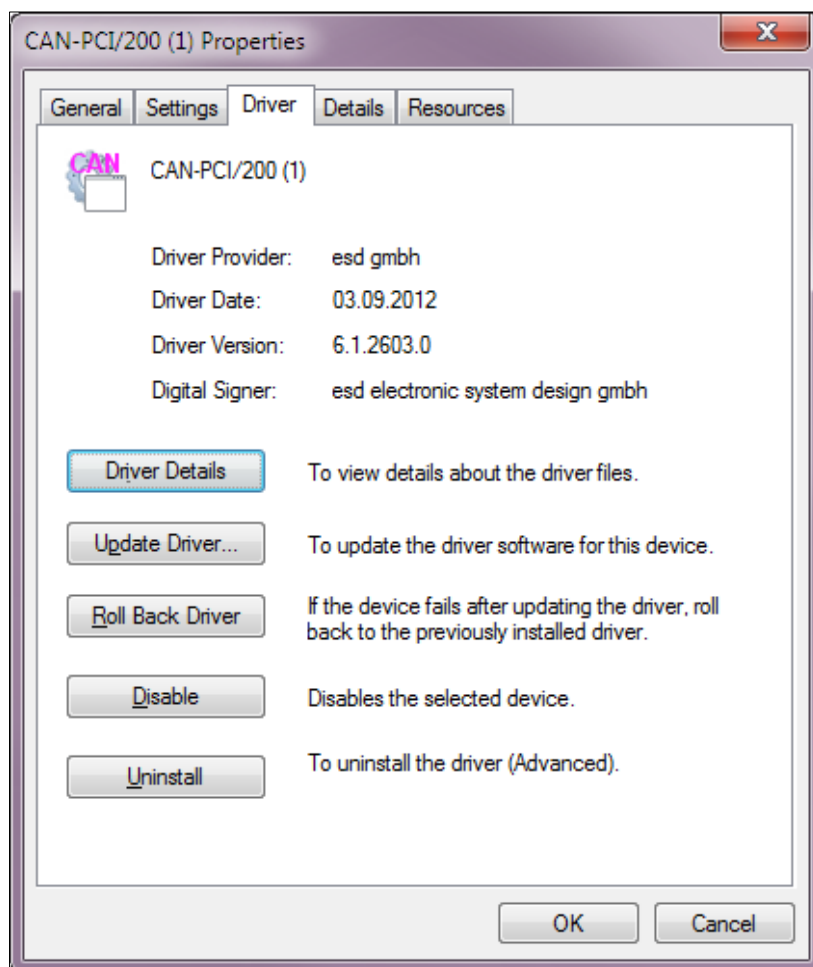
**Note:**

Depending on the files which are updated a restart of the system might be required. The configuration of the device driver is not affected by the driver update. If an update introduces new configuration parameter, they will have the default value.



## 2.5.2 Driver Rollback

If you have updated the initially installed driver to a newer one, starting with Windows XP you can roll back your driver to the previous version. Double click the CAN device in the *Device Manager* and select in the *Properties* dialogue the *Driver* tab as shown in the picture below for Windows 7.



To roll back to a previous version of the driver press the **Roll Back Driver** button and confirm the following security dialogue.

**Note:**

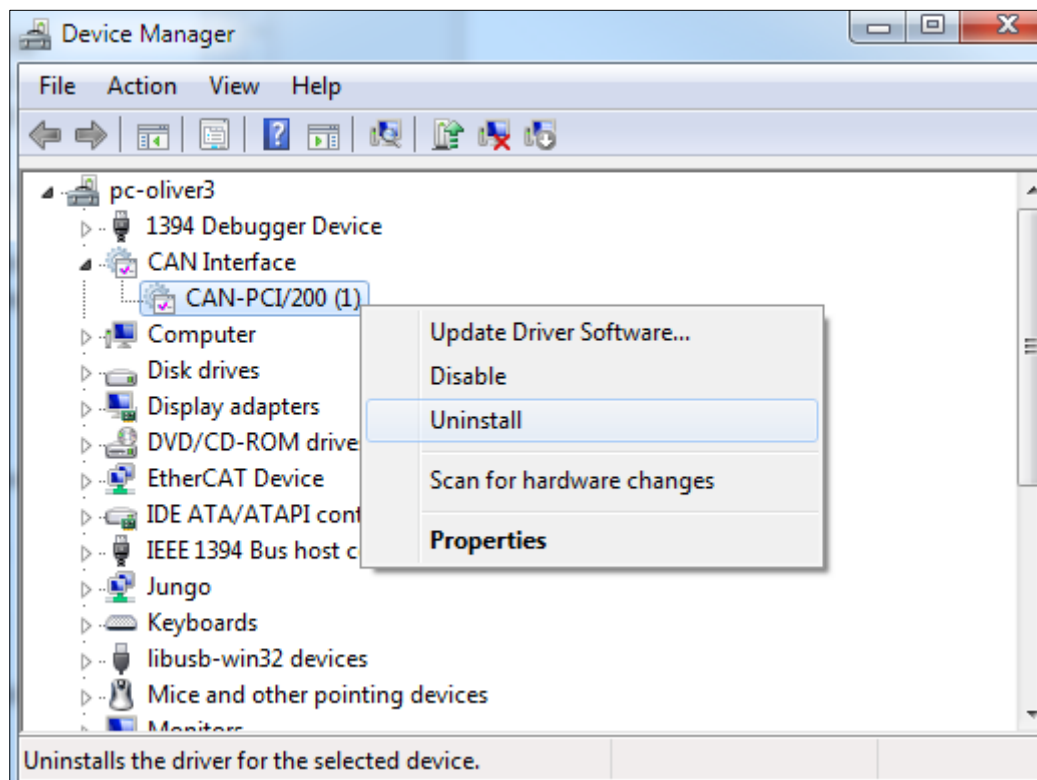
If the installed device driver is the only one in the Windows driver store for this hardware the **Roll Back Driver** button is not enabled.

**Attention!**

A newer version of a device driver often adds functionality and fixes problems that were discovered in earlier versions. Rolling back a driver can cause the loss of that new functionality and can reintroduce the problems that were addressed with the newer version. Furthermore, the DLLs are shared between device driver packages for different **esd** CAN device families so that a roll back of one driver might also affect files of another driver.

### 2.5.3 Driver Uninstall

Starting with Windows Vista you can completely remove an installed driver package from the driver store if you start a device **Uninstall** operation in the device manager



and check the **Delete the driver software for this device** in the confirmation dialogue box.



If an older version of the device driver is available in the driver store Windows will choose this one the next time the device is enumerated so basically this driver uninstall operation is a driver rollback as described in the previous chapter without and implicit re-enumeration.

## 2.6 EtherCAN and EtherCAN/2

In comparison to CAN interfaces connected to a local PC bus (PCI, USB, ...) supported with a Windows kernel mode device driver the EtherCAN and EtherCAN/2 interfaces are supported with a user mode device driver which integrates this remote CAN hardware into the NTCAN architecture in the same way as a local interface. This user mode device driver supports the EtherCAN/2 as well as the legacy EtherCAN hardware but for reasons of simplicity this chapter only refers to the EtherCAN/2.

The EtherCAN/2 driver software comes as a digitally signed Windows installer (see chapter 2.8.3) which supports Windows 2000 and later versions (32-/64-bit).

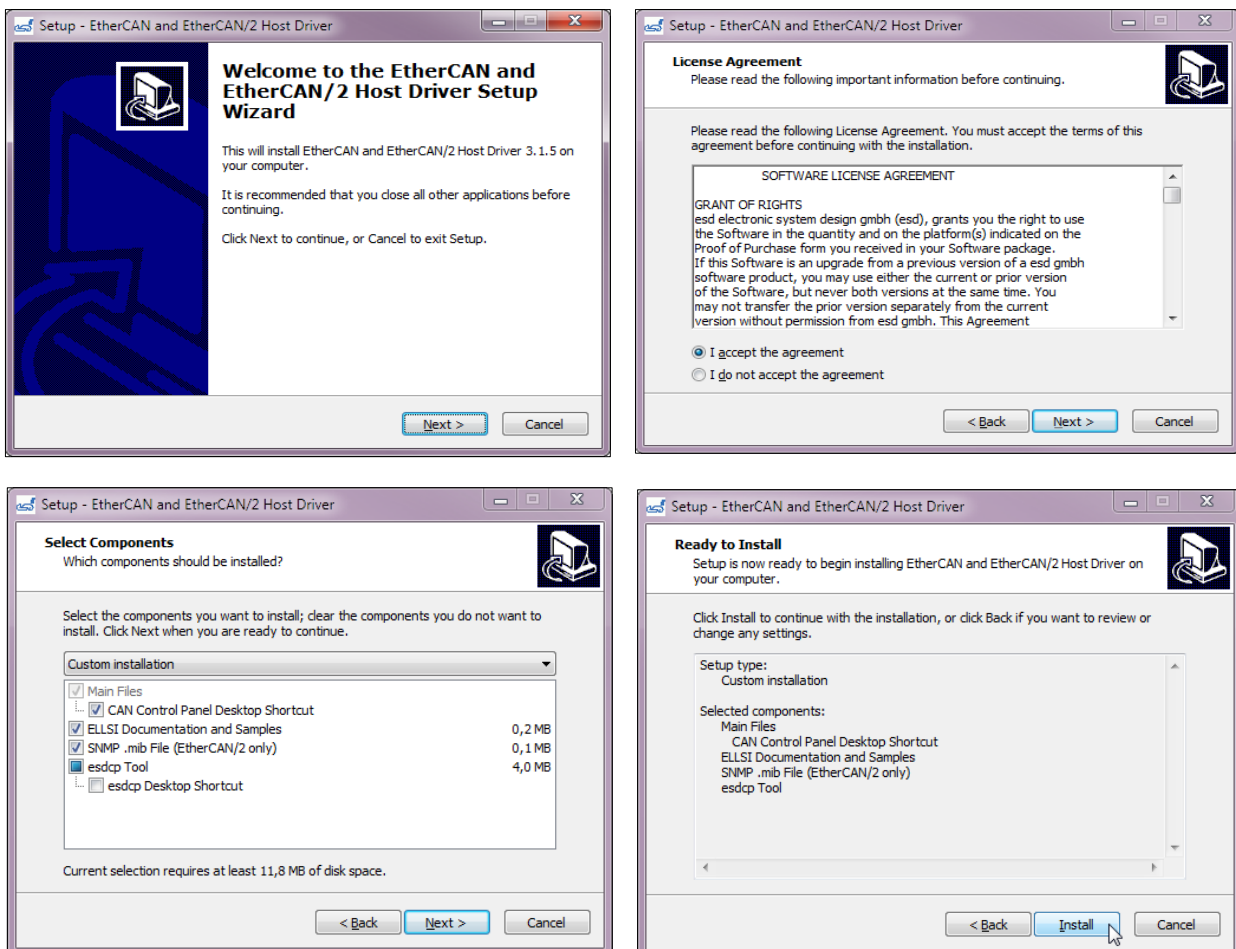


### Attention!

A user which wants to install/uninstall the software must be member of the Administrators group.

### 2.6.1 Installation

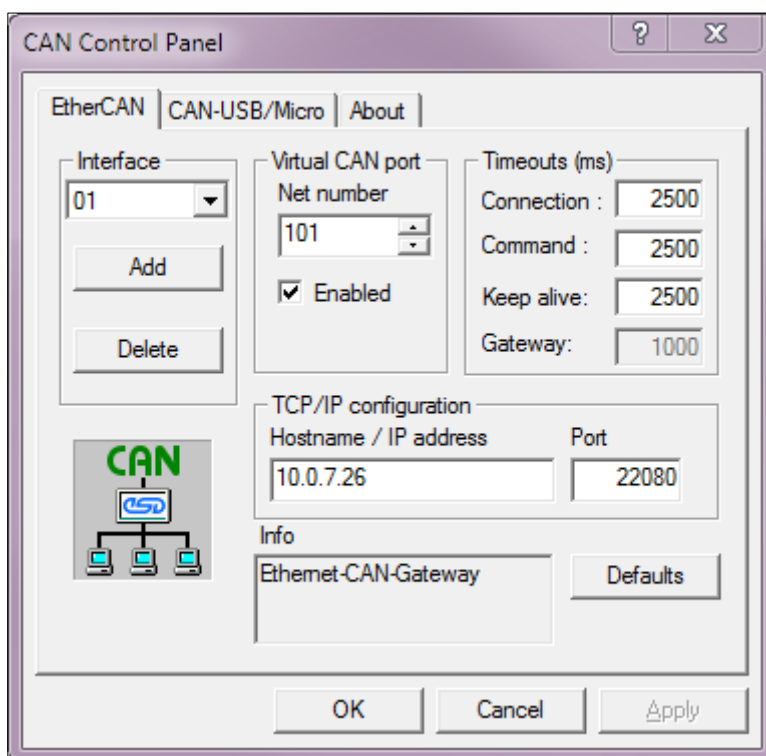
Start the installer application received on CD/DVD or downloaded from the **esd** website and follow the dialogue based setup process shown in the picture below.



## 2.6.2 Configuration

For configuration of the EtherCAN/2 please refer to the EtherCAN/2 hardware manual /3/.

The local driver is configured with the help of the *CAN Control* application that can be started by clicking on “**esd/EtherCAN/CAN Control Panel**” in the Windows Start Menu which opens a dialogue similar to the picture below:



The following driver configuration options are available:

### *Interface*

In the drop-down box you can choose the EtherCAN/2 interface instance. With the Add button you can create additional instances. With the Delete button you can remove the currently selected instance. The default after installation are 5 EtherCAN/2 instances which can not be deleted.

### *Virtual CAN port*

You must assign a CAN network number between 0 and 255 and have to enable it before this net number is available in the NTCAN environment to be used by your application.



### **Attention!**

Please make sure that the assigned logical net number is not already in use by another EtherCAN/2 or another CAN interface attached to a local bus.

### *TCP/IP Configuration*

Configure the IP address or hostname (if registered in the DNS server) of the EtherCAN/2 interface. A change of the default port 22080 is not supported at the moment. The IP address must be identical with the one that is assigned to the EtherCAN/2 (please refer to /3/ for details)

### Timeouts

Currently three separate timeouts can be configured.

The **connection timeout** defines the time the EtherCAN/2 driver waits for a response during the initial connection before the client software returns with a timeout.

The **command timeout** defines the time after which a request to the EtherCAN/2 interface must be replied before the client software returns with a timeout.

The **keep alive timeout** defines the time after which a keep alive request must be replied by the EtherCAN/2 interface before the host driver tries to reset and re-establish the connection.

### Defaults

The Default button restores all driver defaults for the timeout parameter.

**Note:**

If you just want to run NTCAN based application on the system you are done.

If you intend to develop NTCAN based applications on this system you also have to install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

## 2.6.3 Uninstall

To uninstall the *EtherCAN/2 driver* from your computer you must open the **Add/Remove Programs** (Windows 2000/XP) or **Programs and Features** (Windows Vista and later) dialogue via the **Control Panel** and uninstall the *EtherCAN and EtherCAN/2 host driver*.

The process of uninstallation is described in more detail for the *CAN SDK* in chapter 2.7.3.

## 2.7 Windows CAN Software Development Kit (SDK)

After driver installation you can proceed with the installation of the **esd CAN SDK** which supports software development on 32- and 64-bit Windows versions. It contains the necessary files, documentation, and tools to develop, debug and test NTCAN-API based applications.

Please refer to the release notes which gets installed with the CAN SDK for a complete list of supported programming languages and development environments.



**Note:**

If you have installed an older version (before V 2.x) of the CAN SDK uninstall this version before you install the new version.

If you have already installed a CAN SDK revision 2.x or newer you can overwrite the older installation or the uninstall process of a previous version is triggered implicitly if necessary.

The CAN SDK is deployed either on the CAN driver CD you receive with your CAN hardware or can be downloaded as an archive from the **esd** website. To install the **CAN SDK**, start **CAN\_SDK.exe** located in the directory **CAN\_SDK** of the CAN driver CD or start the installer after unpacking the downloaded archive. The installer is digitally signed so you can verify its integrity (see chapter 2.8.4) before you start installation.



**Attention!**

A user which wants to install/uninstall the software must be member of the Administrators group.

Follow the steps of the setup wizard to complete the installation.

### 2.7.1 Setup Command Line Parameter

The following table contains the most important command line parameters which are supported by the setup application to automate the installation process:

<i><b>Parameter</b></i>	<i><b>Description</b></i>
/HELP, /?	Shows a summary of all available parameters.
/SP	Disables the This will install... Do you wish to continue? prompt at the beginning of Setup.
/LOG	Causes Setup to create a log file in the user's TEMP directory detailing actions taken during the installation process. The log file is created with a unique name based on the current date (It will not overwrite or append to existing files). The information contained in the log file is technical in nature and therefore not intended to be understandable by end users.
/LOG="filename"	Same as /LOG, except it allows you to specify a fixed path/filename to use for the log file. If a file with the specified name already exists it will be overwritten. If the file cannot be created, Setup will abort with an error message
/LANG=language	Specifies the language to use (en or de). When a valid /LANG parameter is used, the Select Language dialog will be suppressed.

## 2.7.2 Installation Options

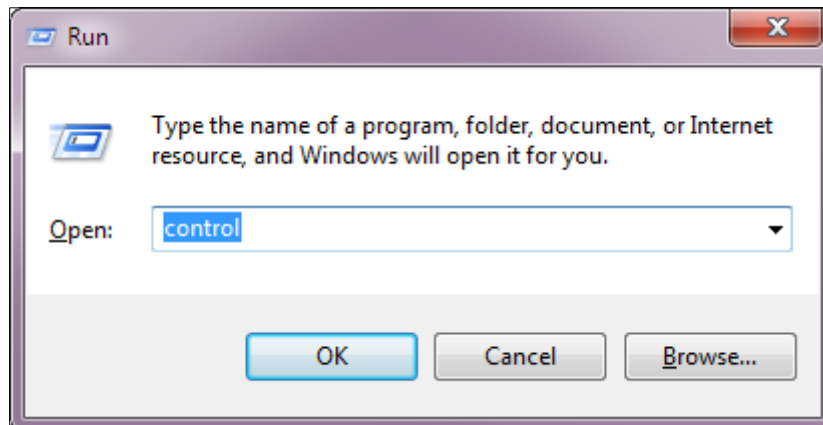
At the start of the installation, you are asked for the installation language. Currently an installation in English and German is supported. The installation language also defines the language of the installed documentation (if available in both languages).

The installer allows to choose between a *full installation*, a *compact installation* and a *custom installation*. The *full installation* installs everything. The *compact installation* installs only the files which are necessary for software development. The *custom installation* lets you choose which components of the SDK are to be installed. The categories currently available are *Tools*, *Documentation*, *Sample Code* and *Software Development* files.

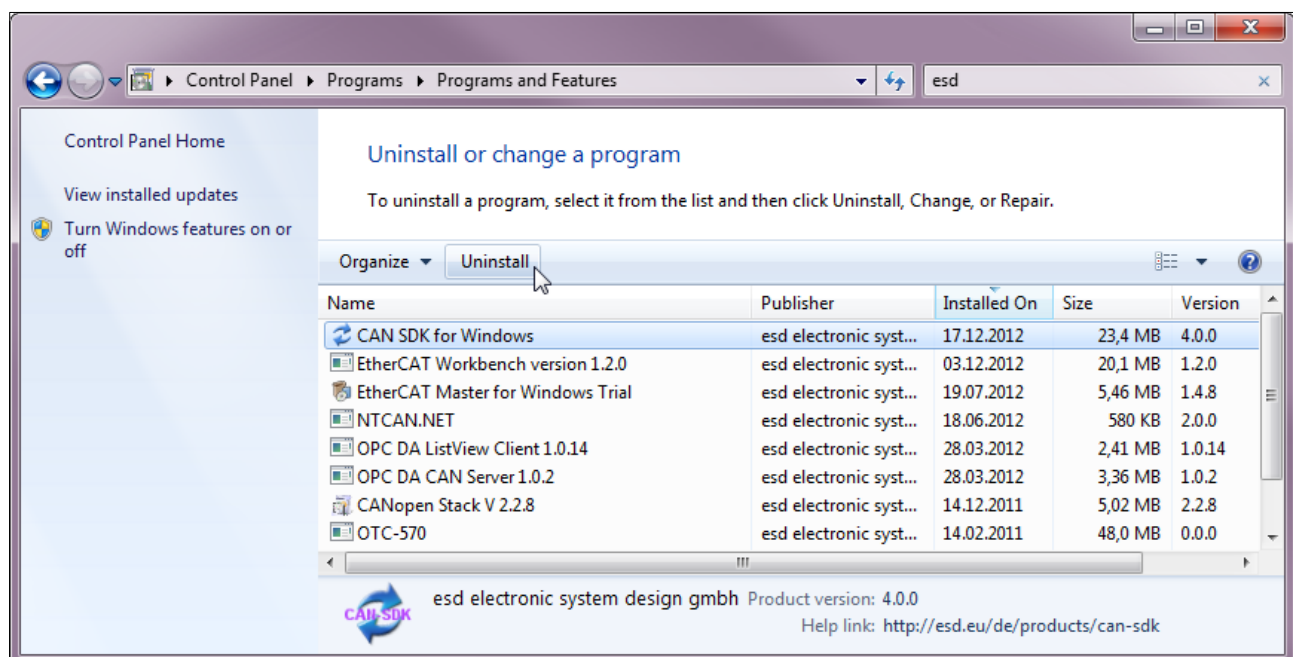
### 2.7.3 Uninstall

To uninstall the *CAN SDK* from your computer you must open the **Add/Remove Programs** (Windows 2000/XP) or **Programs and Features** (Windows Vista and later) dialogue via the **Control Panel**.

To open the **Control Panel** in all versions of Windows open the run dialogue by pressing the keys **WinKey + R** and type *control* as shown below for Windows 7.



Choose in the **Programs and Features** dialogue as shown below for Windows 7 the *CAN SDK*, click **Uninstall** and follow the steps of the wizard.



### 2.7.4 IDE Integration

Many Integrated Development Environments allow to define paths relative to an environment variable. For this reason, during the installation the environment variable `CanSdkDir` is created which is set to the installation directory of the *CAN SDK*. Using this environment variable in paths makes a project independent of the installation directory of the *CAN SDK*.



## 2.8 Digital Signatures

### 2.8.1 Overview

All binaries (device drivers, libraries (DLLs) and installers) of the current **esd** CAN device driver packages and the package itself are digitally signed. A digital signature is an electronic security mark that can indicate the publisher of the software, as well as whether someone has changed the original contents of the files. The code-signing technology built in Microsoft Windows operating system for this is called *Authenticode*.

Certificates issued by certification authorities (CA) trusted by Windows for the initial implementation of *Authenticode* used SHA-1 cryptographic hash functions. These have been deprecated by the National Institute of Standards and Technology (NIST) in 2011 because of significant mathematical weaknesses according to the collision resistance which allows brute force attacks to circumvent the security. For this reason, in 2015 Microsoft published a SHA-1 code signing certificate deprecation policy with the aim to migrate to SHA-2 code signing certificates in future and at this time still supported Windows versions. This SHA-1 deprecation policy distinguishes in the level of support between kernel mode code (device driver) and user mode code for the different major versions of Windows.

The table below gives an overview about the SHA-2 support for the different major Windows desktop versions. This table and everything said below also applies for the respective server versions.

OS	Windows XP	Windows Vista	Windows 7	Windows 8	Windows 10/11
User Mode	Yes	Yes	Yes	Yes	Yes
Kernel Mode	No	No	Yes	Yes	Yes

**Table 12:** Windows Support for SHA-2 Code Signing Certificates



**Note:**

SHA-2 user mode code signing support for Windows XP requires the installation of SP3. Windows 7 requires the KB3033929 update to be installed for SHA-2 signed kernel driver support. SHA-2 signed kernel driver support is not published by Microsoft for earlier versions of Windows.

Before Windows 10, device driver packages are signed with a code signing certificate and the cross-certificate of a CA which issued the certificate used for signing. This CA must belong to the group of CAs which are trusted by Microsoft. The mechanism of signing a device driver that way is referred to as **Cross Signing**.

Starting with Windows 10, Microsoft changed its general policy for kernel mode code signing by making it mandatory that all Windows kernel mode drivers must be submitted to and digitally signed by the Windows Hardware Developer Centre Dashboard Portal instead of performing an (in-house) cross signing. This portal only accepts (device driver) submissions with a valid EV Signing Certificate which are, since 2016, only submitted by the CAs based on SHA-2. This mechanism of signing device driver (which is only applicable for Windows 10 and later) is referred to as **Attestation Signing**.

Windows 10 enforces the policy of Attestation Signing after a grace period which ended with the release of Windows 10 1607 (aka *Anniversary Update*) and if the following two conditions are met:

- Fresh installation of Windows (no upgrade from a version before 1607)
- Secure Boot enabled

All cross-certificates trusted by Microsoft expired in July 2021 so Cross Signing device driver is no longer possible.

For CAN device driver released by **esd** the Microsoft kernel mode code signing policy has the following impact:

- All device drivers released before March 6, 2017 are “cross-signed” with an SHA-1 certificate issued to *esd electronic system design gmbh* and they should be accepted by all versions of Windows XP and later.
- All device drivers released after March 6, 2017 are “cross-signed” with an SHA-2 EV certificate issued to *esd electronic system design gmbh* and they should be accepted by all versions of Windows 7 with installed KB3033929 and later. **esd** can no longer provide device driver which install flawlessly on Windows versions before Vista.
- All device drivers released between May 29, 2019 and March 19, 2021 are “attestation-signed” with an SHA-2 EV certificate issued to *Microsoft Windows Hardware Compatibility Publisher* as well as “cross-signed” with an SHA-2 EV certificate issued to *esd electronics gmbh* and should be accepted by all versions of Windows 7 with installed KB3033929 and later.
- All device driver released after March 19, 2021 are “attestation-signed” with an SHA-2 EV certificate issued to *Microsoft Windows Hardware Compatibility Publisher* and should be accepted by all versions of Windows 10 and later.



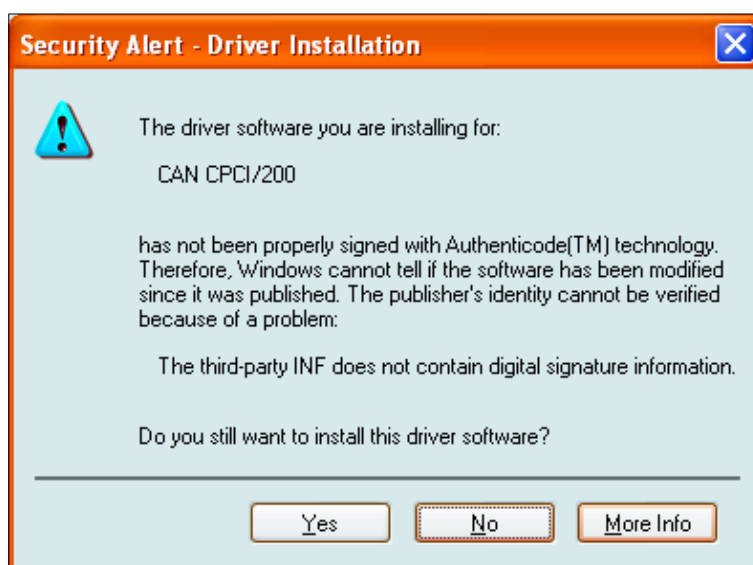
**Note:**

As Microsoft ended the support to allow in-house cross-signing of device driver code, it is technically no longer possible for esd to release new or updated device drivers for Windows versions before Windows 10.

## 2.8.2 Driver Installation

Windows indicates an invalid signature of a driver package in the following ways:

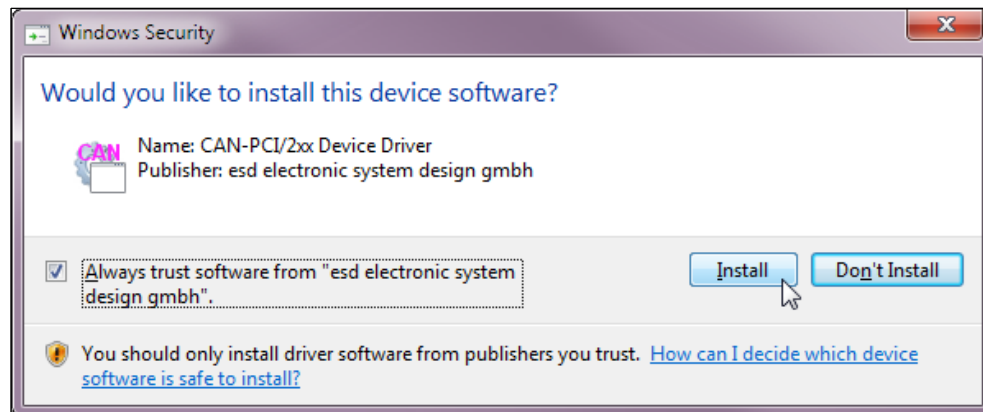
- **Windows rejects to install a device driver.** This can occur if one of the following conditions is met:
  - **Any 64-Bit Windows version:** The operating system fails to validate the digital signature of the driver package because for example one or more files were altered after the driver package has been digitally signed.
  - **Windows 10 1607 (32-/64-Bit) or later:** The device driver was not signed by the Windows Hardware Developer Centre Dashboard Portal on fresh installations of the operating system (no upgrade from a version before 1607) with enabled secure boot.
  - **Windows XP/Vista (64-Bit):** Device driver released after March 6, 2017 which are signed with an SHA-2 certificate which is not supported.
- **Windows can't verify the publisher of the device driver.** This occurs if a device driver either has no digital signature or it has been signed with a digital signature that could not be verified by a certification authority. On Windows versions before Vista during the process of driver installation, depending on the system configuration, a dialogue box may indicate that the *Authenticode* signature is invalid.



Windows versions before Windows Vista cannot validate a correct SHA-1 signature because the trusted chain of the certificates is not completely stored in them and they accept only a signature created by Microsoft during a WHQL certification process. In this special case you can ignore the warning and continue with the installation.

- **The device driver has been altered.** This occurs if files in the driver package are altered on 32-bit Windows versions after it was digitally signed by **esd**.

On Windows Vista and later after a successful validation of the certificate a message shows the name of **esd** as vendor that has signed the driver package.



You can optionally decide to add the **esd** certificate in the *Trusted Publishers* certificate store which will prevent this dialogue for other digitally signed **esd** software and for device driver updates

If you get an indication about an invalid signature (with the exception described for Windows versions before Vista) please contact **esd**.

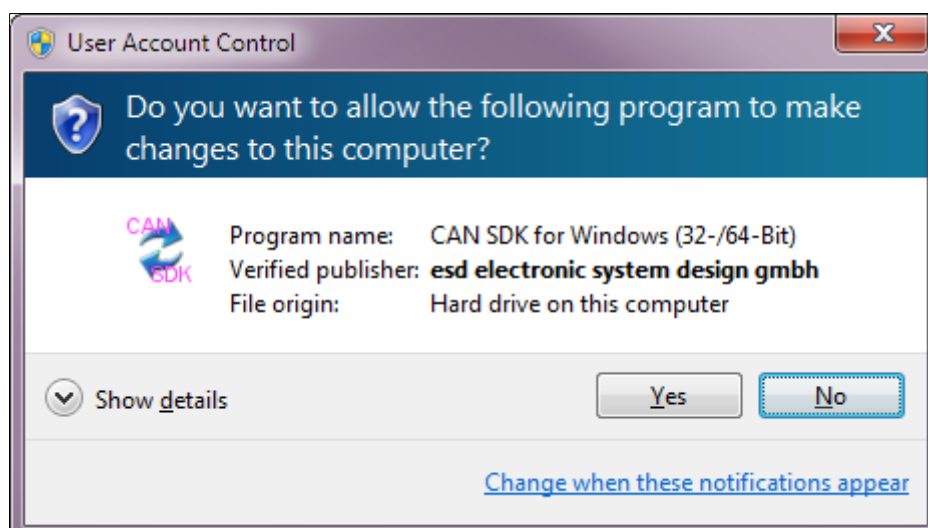


**Note:**

The 64 bit versions of Windows do not allow to install and use device drivers which are not digitally signed by the vendor which means that it is also not possible to just replace the driver .sys file with a newer version which is still possible on the 32 bit versions of Windows.

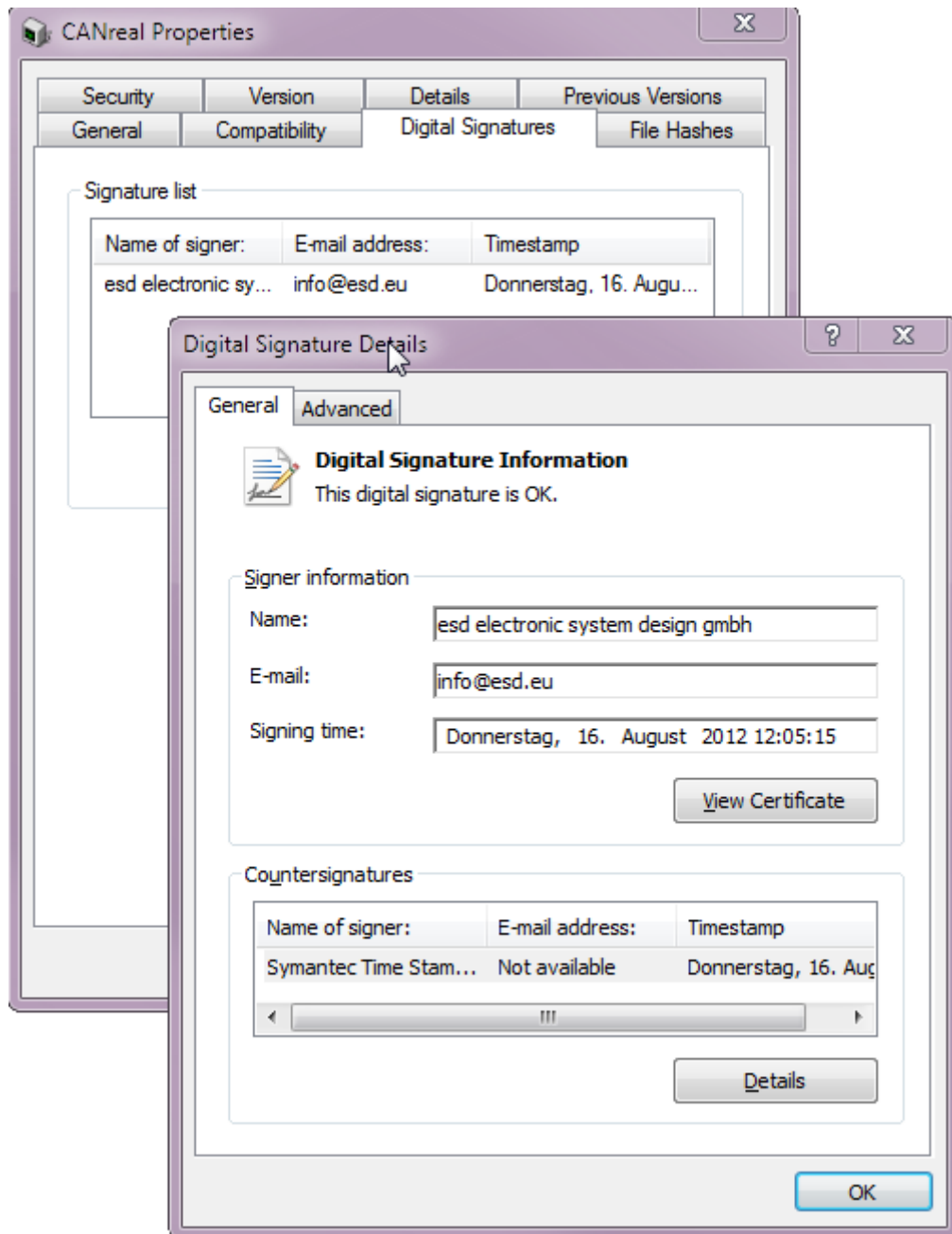
### 2.8.3 Software Installation

Windows installer based software by **esd** is also digitally signed and on UAC enabled systems (Windows Vista and later) you will see an UAC dialogue similar to the picture below which indicates that the **esd** is the publisher of the software and the binary was not altered by third parties since it was signed.



## 2.8.4 Digital Signature Verification

To view digital signatures of a binary, open the context menu of the file and select **Properties**, then go to the **Digital Signatures** tab.

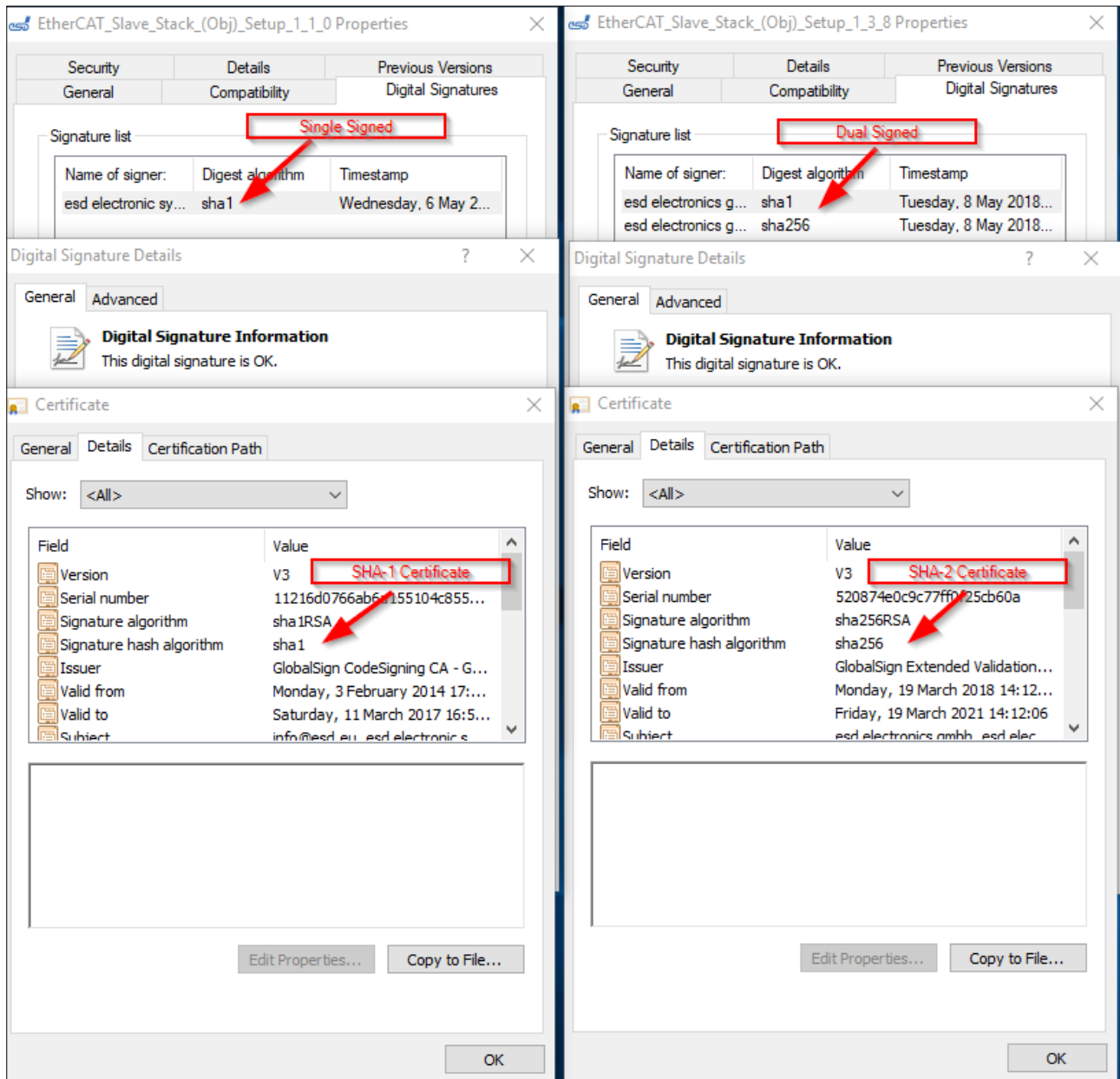


The dialogue box provides information that **esd** is the software publisher, about the certification authority that issued the certificate and the date the code was signed (timestamp). Finally it indicates that the digital signature for the file is valid.

## Windows®

As described in chapter 2.8.1 the Windows XP SP3 with KB3033929 and Windows Vista support SHA-2 certificates at least for user mode code but they do not support the improved cryptographic hash functions which come with SHA-2 for the digest algorithm. For this reason, all binaries released by **esd** after March 6, 2017 are dual signed with the SHA1 and the SHA256 algorithm.

The picture below shows the difference between a single signed binary with SHA-1 certificate and a dual signed binary with SHA-2 certificate.



## 2.9 Legacy Windows Versions

This chapter covers installation and configuration of the device driver for the older versions of Windows operating systems (Windows 9x/ME, Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7) which have not been maintained by Microsoft for several years (which means that the end of the mainstream as well as the extended support according to the Microsoft product life-cycle management has been reached).

**Attention!**

Active technical support by **esd** and development for these versions of Windows have stopped but the latest version of the device driver files is kept available.

### 2.9.1 Windows 7 / 8.x / Server 2008 R2

This chapter covers the device driver installation for Windows 7, Windows 8.x and Windows Server 2008 R2 (which is implicitly also referenced if the following text refers to the Windows desktop versions).

**Attention!**

The expiration of cross-signing certificates by Microsoft in July 2021 (see 2.8.1) which is required for (in-house) Authenticode signing makes it impossible for **esd** to provide updated device drivers for all legacy Windows versions.

The installation procedure is identical for the 32-bit and the 64-bit version of these operating systems, but different driver binaries are required. On the 64-bit version all libraries to run 32-bit NTCAN based applications in the *WoW64* subsystem are installed automatically.

**Attention!**

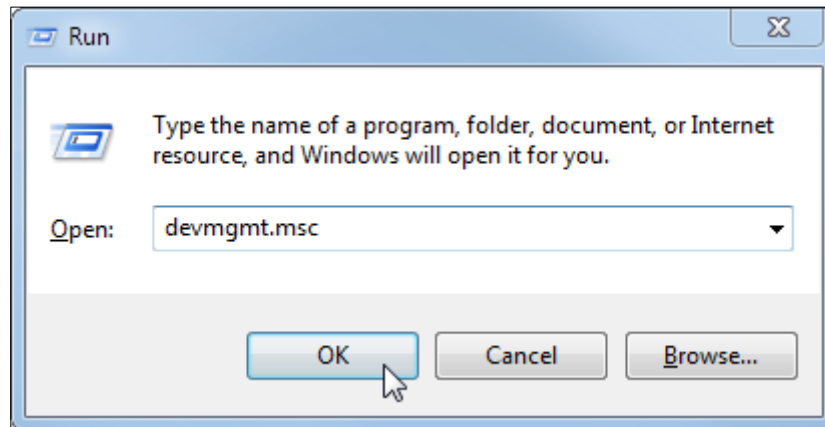
A user which wants to install a device driver must be member of the Administrators group.

#### 2.9.1.1 Hardware-First Driver Installation

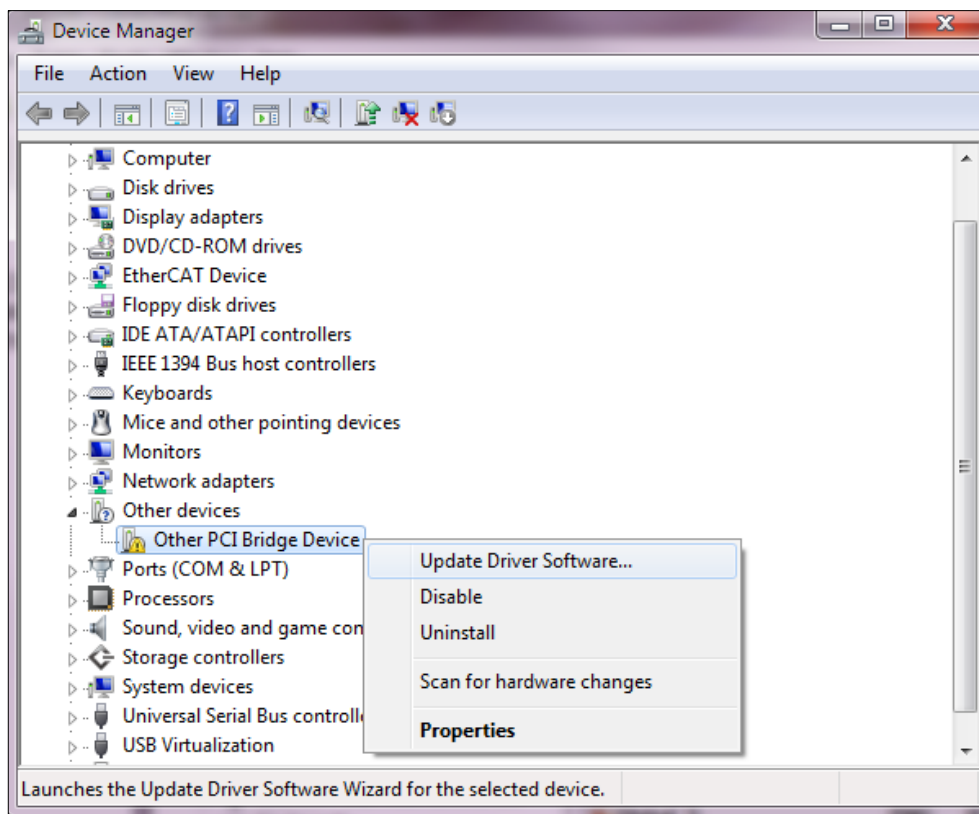
To initiate the device driver installation process, you have to connect the CAN module to your system. Depending on the *Hot Plugging* capability of the hardware you might have to shut down Windows for this. Please refer to the CAN module specific hardware manual for advises.

Starting with Windows 7 the presence of a new hardware does not automatically start the *Found New Hardware Wizard* to locate and install a driver for the new device with user interaction as in previous versions of Windows. To initiate the interactive device driver installation, you now must open the *Device Manager*. Open the *Device Manager* dialogue by pressing the key combination **WinKey + Pause/Break** and type in *devmgmt.msc* into the search box followed by the *Enter* key.





In the *Device Manager* windows there will be a device under *Other Devices* with a yellow exclamation point next to the icon to indicate that there is no device driver installed yet. The text next to the device will depend on the CAN module attached.



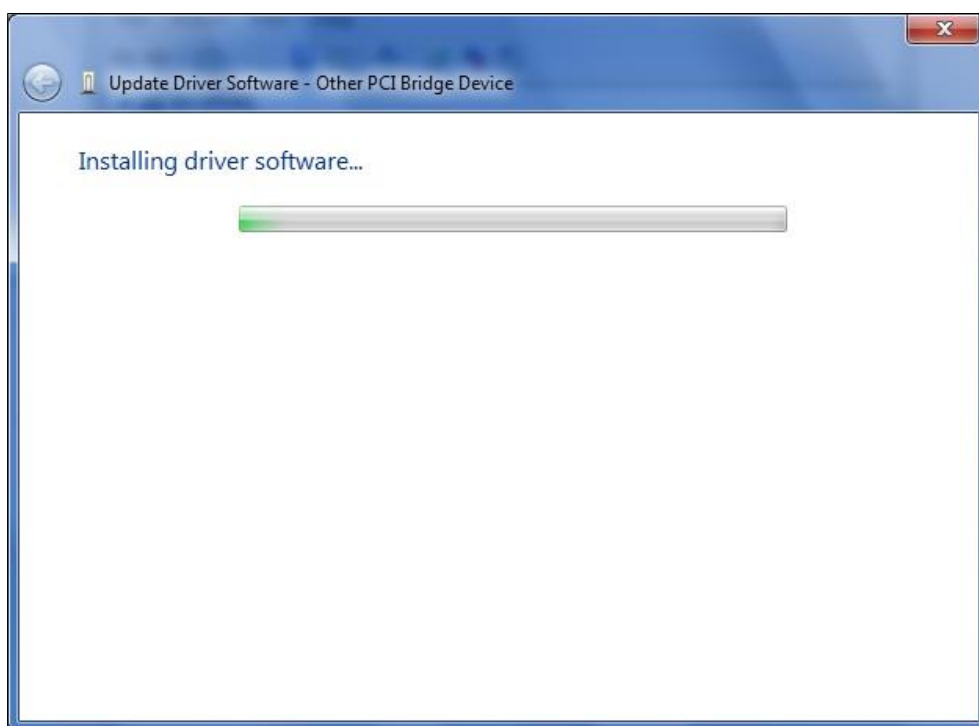
Right click on the device to bring up the context menu as shown above and select the menu item **Update Driver Software...** which opens the following dialogue box.



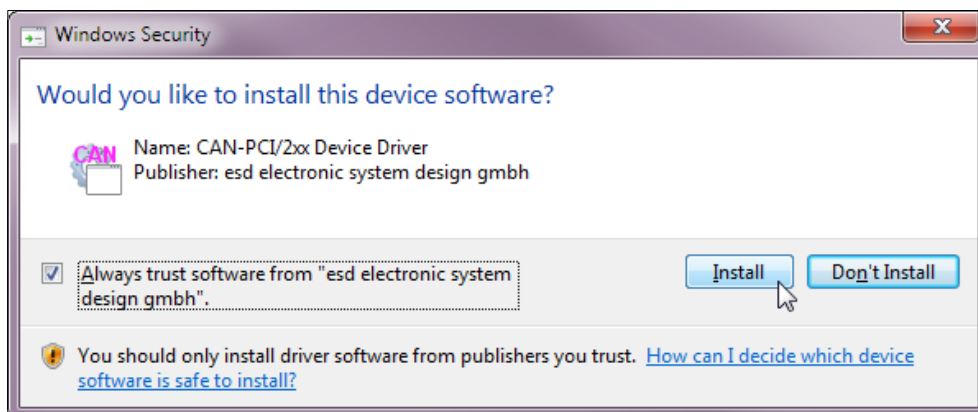
Select the second option to “Browse the computer for driver software” and the following dialogue box will appear.



Press the **Browse...** button to define the location of the driver files. This might either be the drive letter of your optical driver if you want to use the CD which accompanied the delivery of your CAN module or is the location on your hard disk where you have extracted a driver archive downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu)). If the driver files are located in a sub directory of the configured path do not forget to check the “Include subfolders” option in the dialogue before you press the **Next** button to start copying the files to your system which may take some time.



During driver installation you will see a security message similar to in the dialogue below.



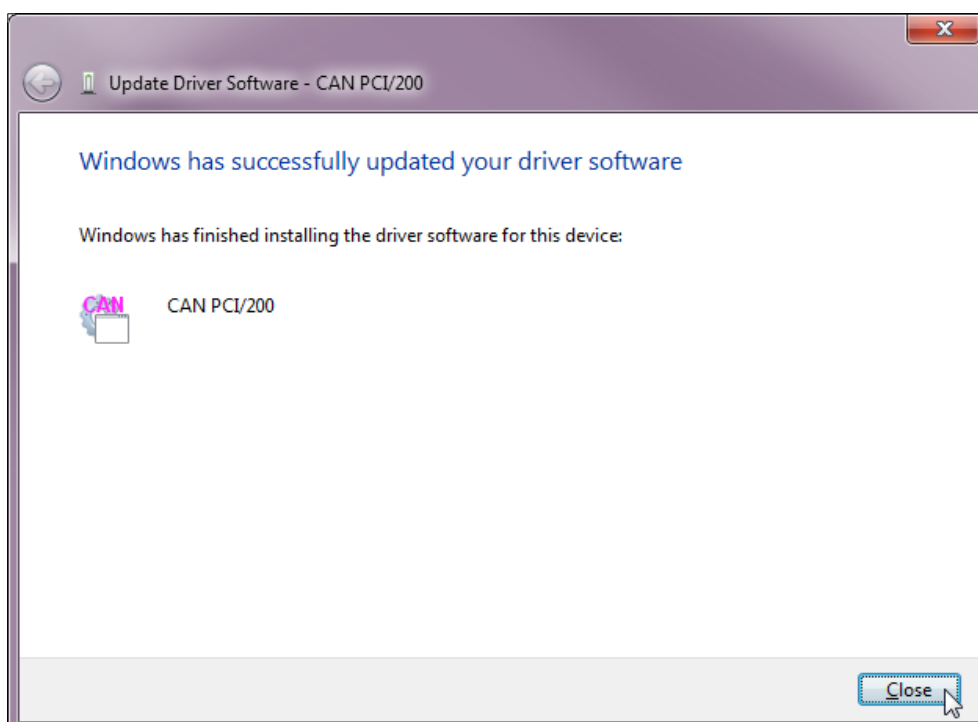
All CAN device drivers are digitally signed to give you as end user who is installing this software the possibility to verify that **esd** is really the publisher of this driver package and that the binaries are not tampered. Please refer to chapter 2.8 for more details about *Digital Signatures*.



**Note:**

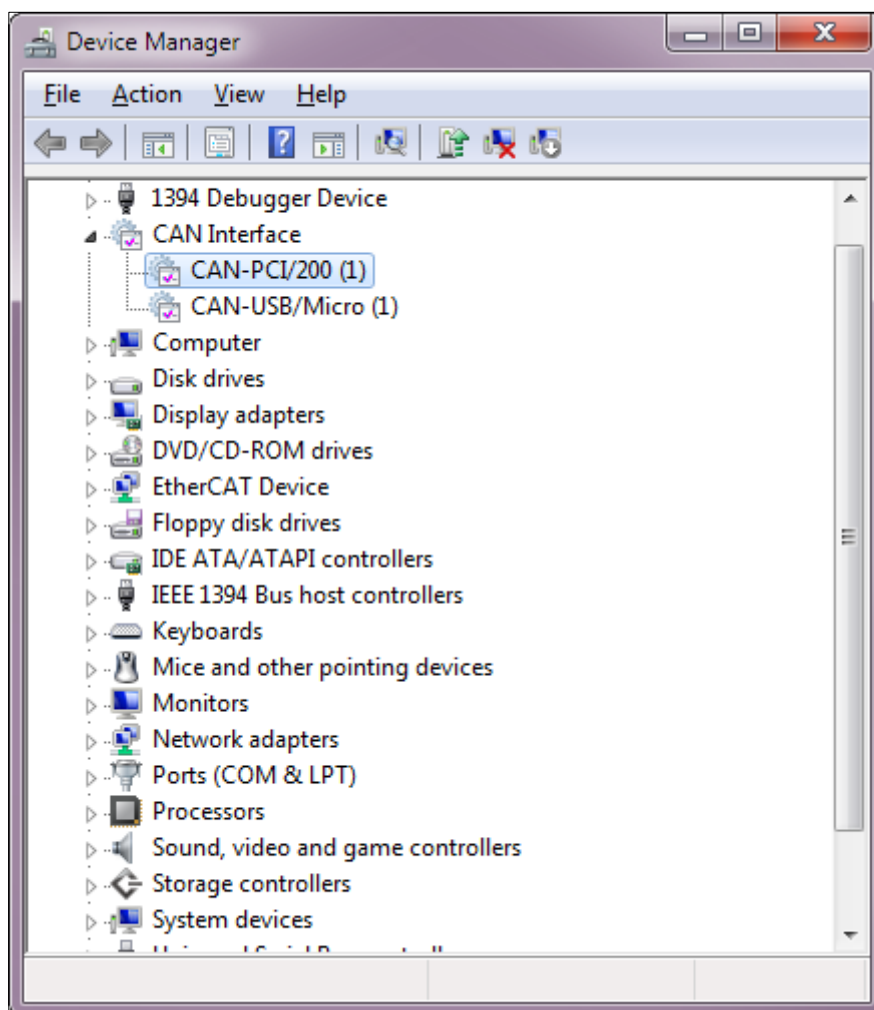
If you activate the check box "Always trust software from *esd electronic system design gmbh*" you will not have to confirm this dialogue in the future during the installation of another digitally signed driver for an **esd** device.

Press the **Install** button to continue.



When the installation is finished a completion dialogue as above is displayed and the driver is now started automatically with every Windows start-up. The displayed device name depends on the CAN module. Press the **Close** button to complete the installation.

If you return to the *Device Manager* window you will see that the CAN module is now listed below the new device class “CAN Interface”.



If you have installed several **esd** CAN modules attached to a local bus of your system you will find all of them here. By double clicking the device you will open the *Properties* dialogue where you can configure the device specific options described in chapter 2.2 via the *Settings* tab.

**Note:**

If you just want to run NTCAN based application on the system you are done.

If you intend to develop NTCAN based applications on this system you also must install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

### 2.9.1.2 Software-First Driver Installation

The process for a software-first driver installation is similar on all versions of Windows since Vista and is covered in chapter 2.3.

### 2.9.1.3 Driver Lifecycle Management

The process of updating, rolling back or uninstalling a device driver package is very similar on all Windows versions and covered in chapter 2.5.

## 2.9.2 Windows Vista® and Server 2008

This chapter covers the device driver installation for Windows Vista and Windows Server 2008 (which is implicitly also referenced if the following text refers to Windows Vista). The installation procedure is identical for the 32-bit and the 64-bit version of these operating systems but different driver binaries are required. On the 64-bit version all libraries to run 32-bit NTCAN based applications in the *WoW64* subsystem are installed automatically.

**Attention!**

A user which wants to install a device driver must be member of the Administrators group.

### 2.9.2.1 Hardware-First Driver Installation

To initiate the device driver installation process, you have to connect the CAN module to your system. Depending on the *Hot Plugging* capability of the hardware you might have to shut down Windows for this. Please refer to the CAN module specific hardware manual for advises.

After Windows Vista has detected the new hardware, it will start the interactive driver installation process of the *Found New Hardware Wizard* with the dialogue below where the hardware name is not “CANbus Controller” as in this example for all **esd** CAN modules.

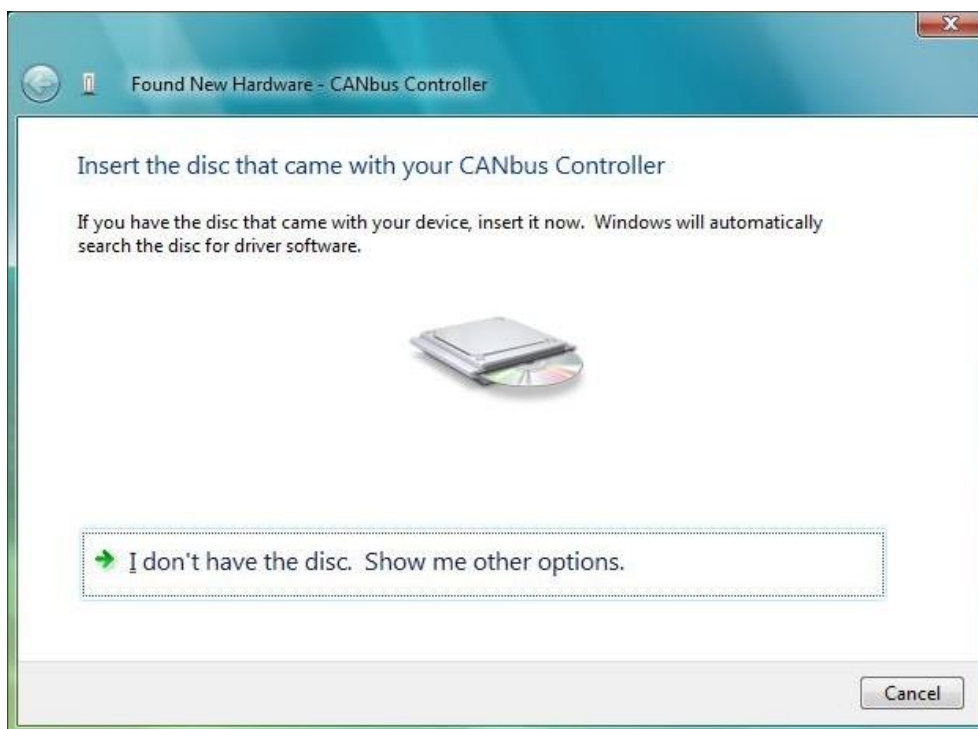


Choose **Locate and install driver software** to continue.

Depending on the system configuration an UAC (User Account Control) dialogue has to be confirmed before the installation can be started as administrator privileges are required for this.

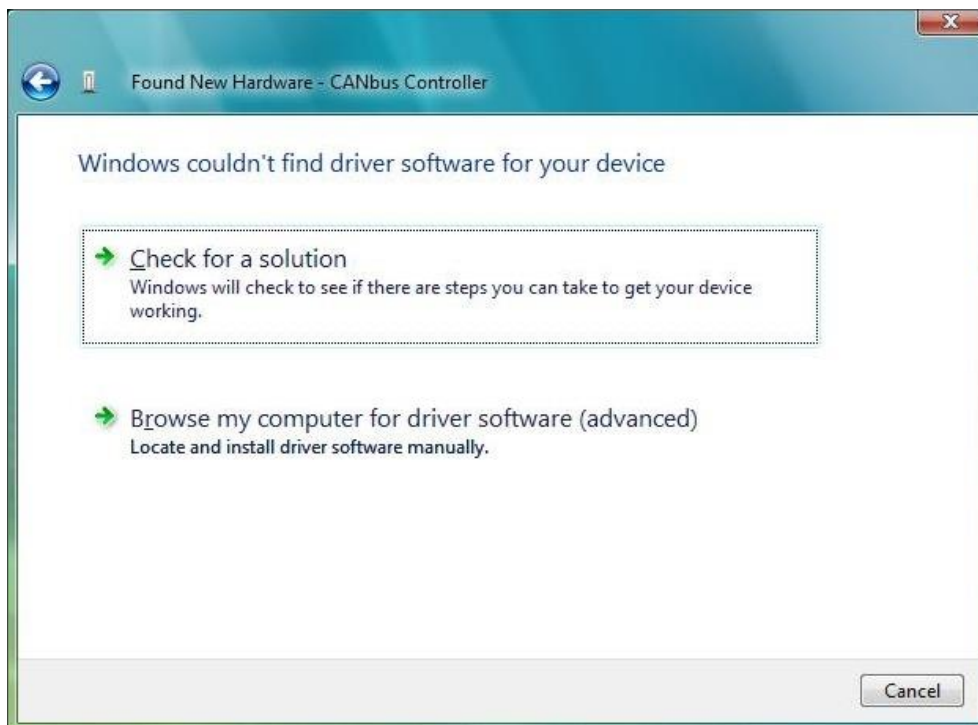
Depending on the configuration of the *Windows Update Driver Settings*, which can be changed via the *Hardware* tab of the *System Properties* dialogue which is opened if you select *Control Panel/System/Advanced System Settings*, Windows Vista will ask you if you want to search for a suitable driver online. If such a dialogue appears choose *Don't search online*, otherwise wait until the online search on Windows Update is completed which may take up to one minute.

Now insert your **esd** CAN CD which accompanied the delivery of your CAN module and the installation process will continue.

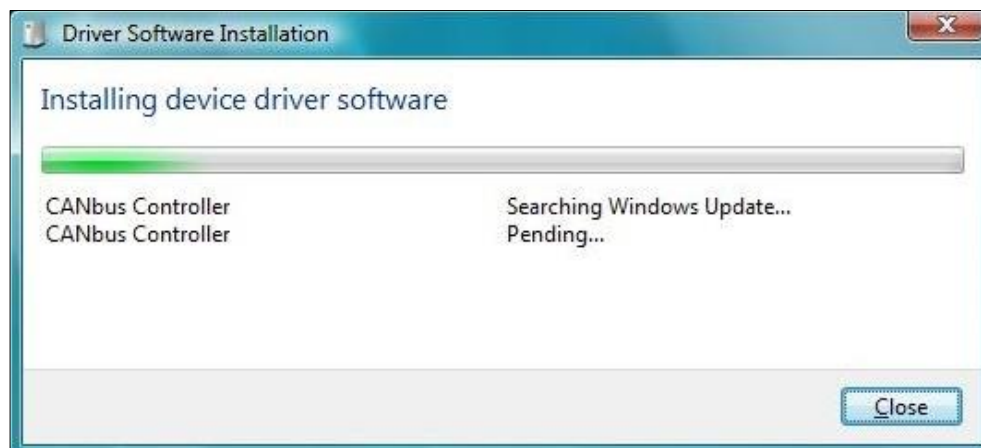


If you have no CD but the driver files are located on your hard disc where you have extracted a driver archive downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu)) choose *I don't have the disc. Show me other options*.

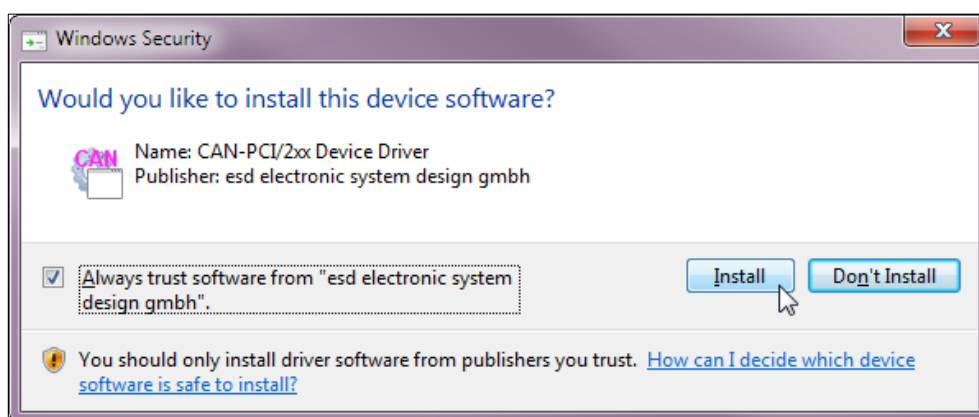
In the following dialogue choose *Browse my computer for driver software* which is followed by a file dialogue to locate the driver files.



Now the files are copied to your system which may take some time.



During driver installation you will see a security message similar to in the dialogue below.

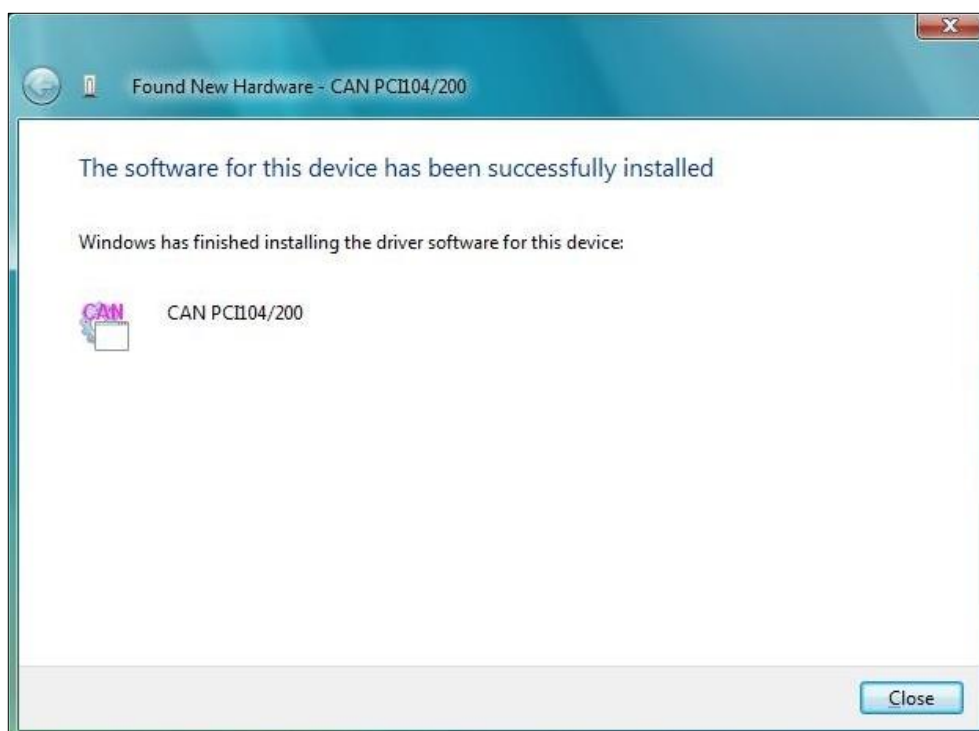


All CAN device drivers are digitally signed to give you as end user who is installing this software the possibility to verify that **esd** is really the publisher of this driver package and that the binaries are not tampered. Please refer to chapter 2.8 for more details about *Digital Signatures*.

**Note:**

If you activate the check box "Always trust software from *esd electronic system design gmbh*" you will not have to confirm this dialogue in the future during the installation of another digitally signed driver for an **esd** device.

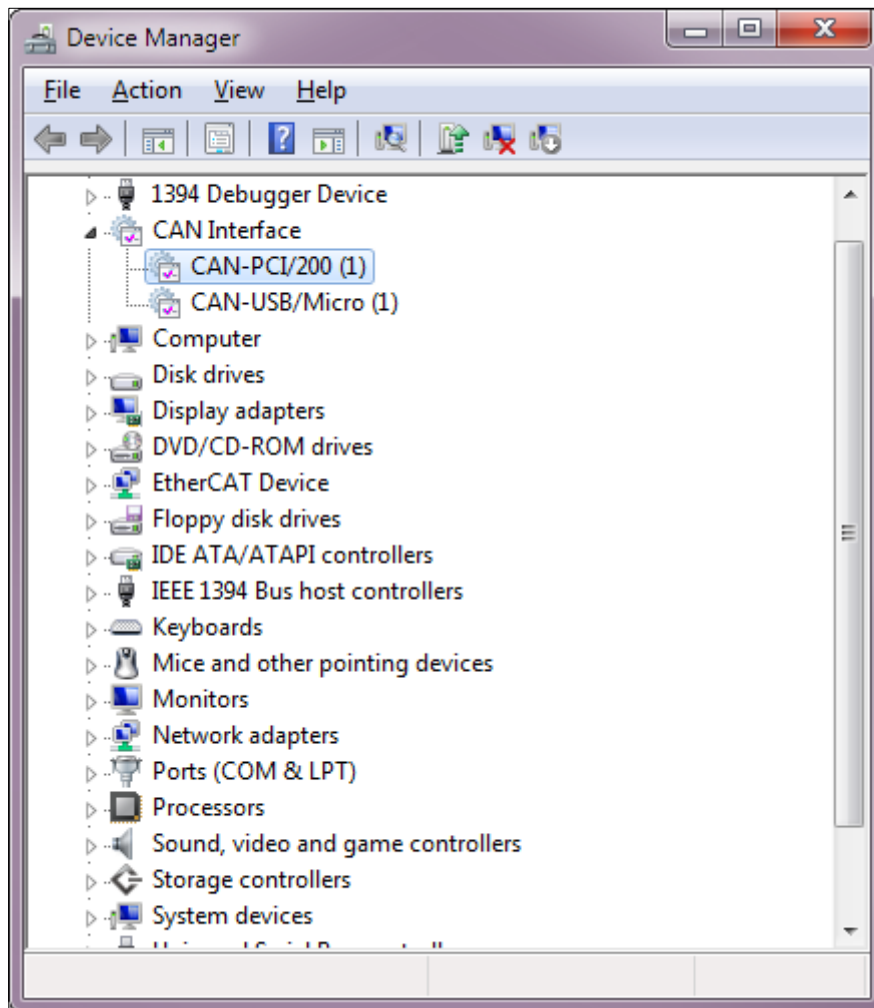
Press the **Install** button to continue.



When the installation is finished a completion dialogue is displayed and the driver is now started automatically with every Windows start-up. The displayed device name depends on the CAN module. Press the **Close** button to complete the installation.



If you now open the *Device Manager* window you will see that the CAN module is now listed below the new device class **CAN Interface**.



If you have installed several **esd** CAN modules attached to a local bus of your system you will find all of them here. By double clicking the device you will open the *Properties* dialogue where you can configure the device specific options described in chapter 2.2 via the *Settings* tab.



**Note:**

If you just want to run NTCAN based application on the system you are done.  
If you intend to develop NTCAN based applications on this system you also have to install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

### 2.9.2.2 Software-First Driver Installation

The process for a software-first driver installation is similar on all versions of Windows since Vista and is covered in chapter 2.3.

### 2.9.2.3 Driver Lifecycle Management

The process of updating, rolling back or uninstalling a device driver package is very similar on all Windows versions and covered in chapter 2.5.



## 2.9.3 Windows XP and Server 2003

This chapter covers the device driver installation for Windows XP and Windows Server 2003 (which is implicitly also referenced if the following text refers to Windows XP). The installation procedure is identical for the 32-bit and the 64-bit version of these operating systems, but different driver binaries are required. On the 64-bit version all libraries to run 32-bit NTCAN based applications in the *WoW64* subsystem are installed automatically.

**Attention!**

A user which wants to install a device driver must be member of the Administrators group.

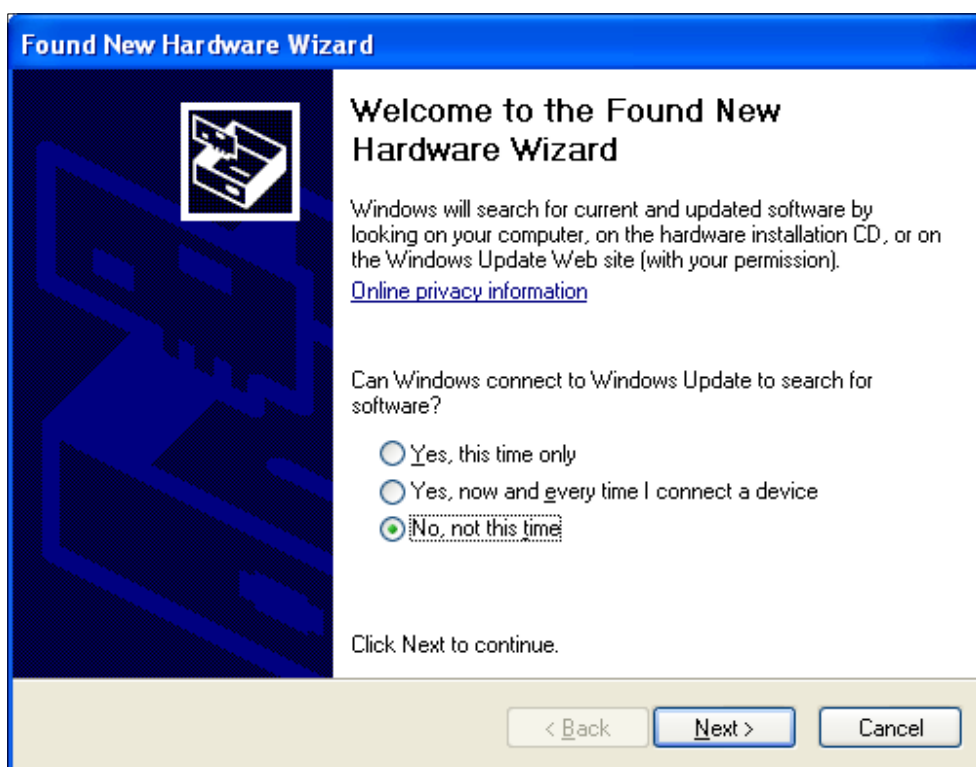
### 2.9.3.1 Hardware-First Driver Installation

To initiate the device driver installation process, you have to connect the CAN module to your system. Depending on the *Hot Plugging* capability of the hardware you might have to shut down Windows for this. Please refer to the CAN module specific hardware manual for advises.

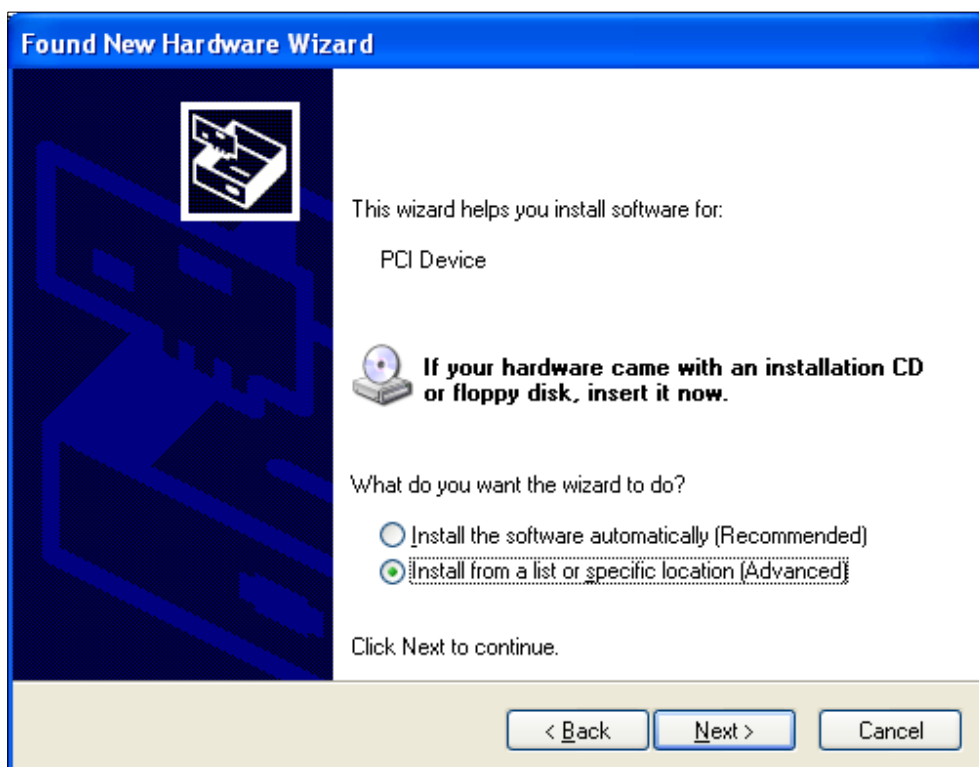
**Note:**

This chapter covers the installation of the device driver for *Plug and Play* capable CAN modules which are connected to a local bus (PCI, USB, ...) of your system. Non PnP **esd** CAN modules for legacy buses (ISA, Parallel Port,...) are supported only for 32-bit Windows XP in the same way as for Windows 2000 (see chapter 2.9.4.4 for details).

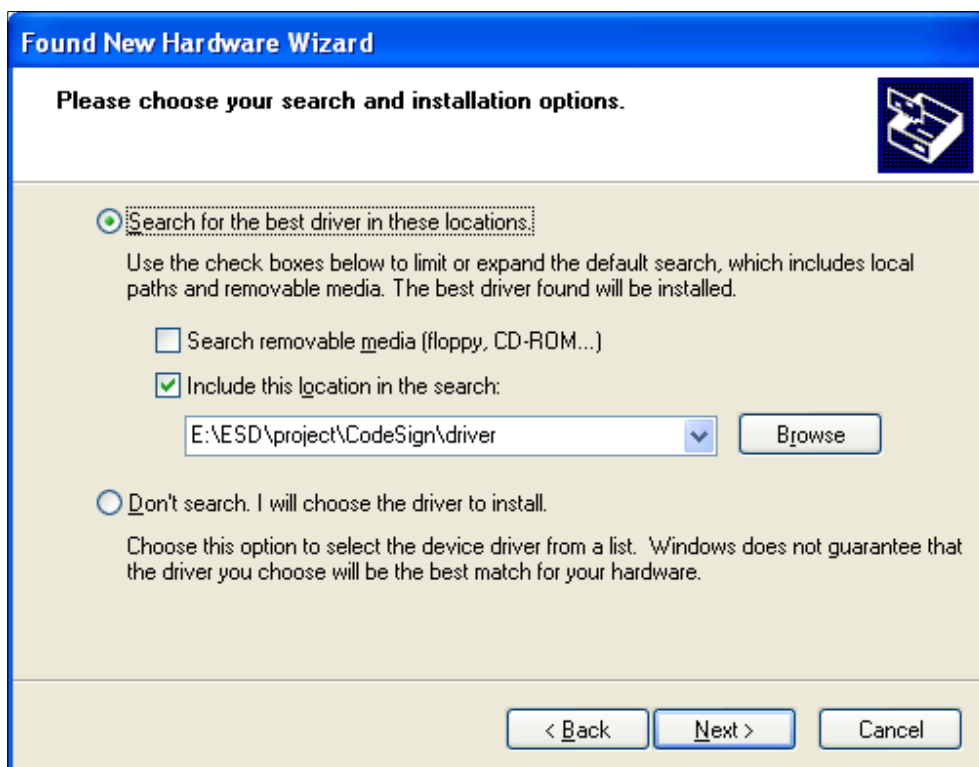
After Windows XP has detected the new hardware, it will start the interactive driver installation process of the *Found New Hardware Wizard* with the dialogue below.



Select *No, not this time* from the available options and proceed by clicking the **Next** button.



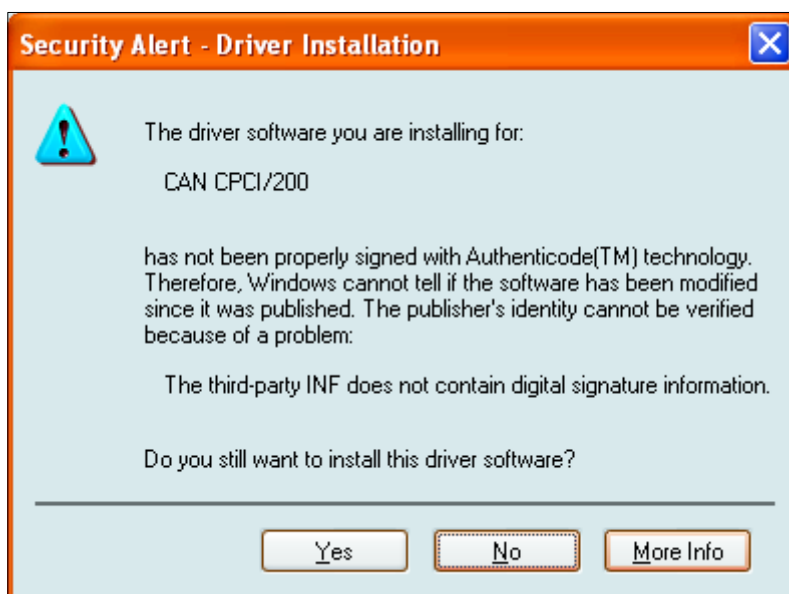
In the dialogue box above choose *Install from a list or specific location (Advanced)* before you click the **Next** button. The reported device type in this and the following dialogues depends on the CAN module.



Select *Search for the best driver in these locations* and choose *Search removable media* if the **esd** CAN CD which accompanied the delivery of your CAN module is inserted into your optical drive or choose *Include this location in the search* and browse for the location of the driver files on your hard disc where you have extracted a driver archive downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu))

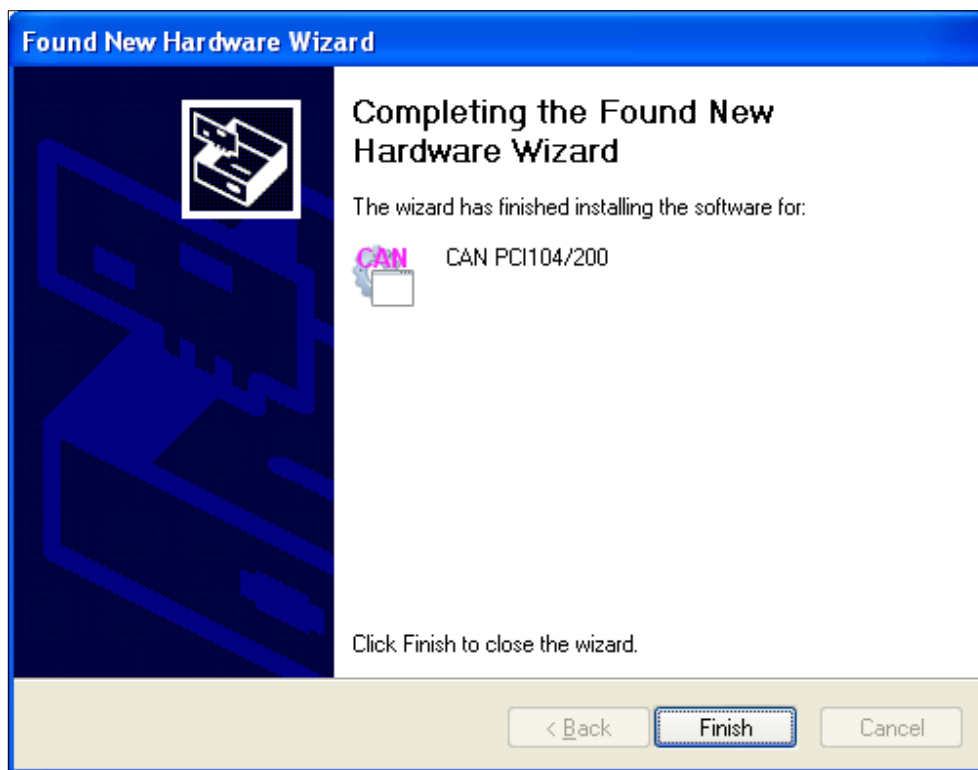
Press the **Next** button to continue device driver installation.

If Windows is configured to ignore file signature warnings the installation process is started and no further dialogue will appear. If Windows is configured to warn when unsigned (non WHQL certified) drivers are about to be installed, a *Security Alert* dialogue will appear. The dialogue text depends on the Windows version (32- or 64-bit).

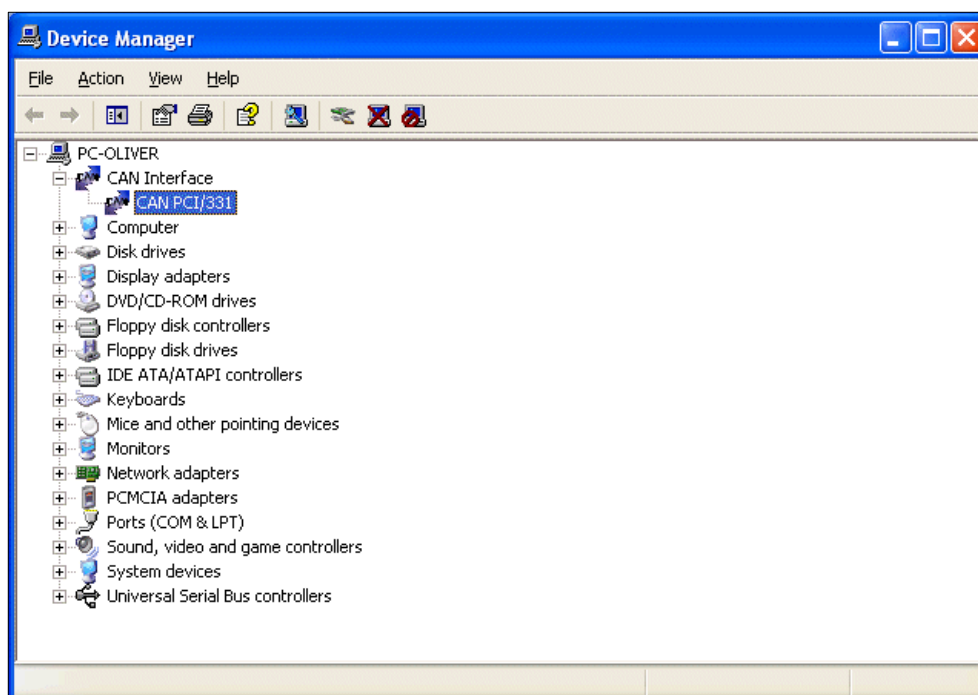


The complete CAN driver package is digitally signed and you will see a dialogue box, which indicates that Windows has verified that the device driver is released by **esd** and hasn't been tampered. As both (32-bit and 64-bit) driver packages are not signed by the WHQL (Windows Hardware Quality Labs) the Security Alert dialogue has to be confirmed with **Yes** to continue the driver installation. For more details about device driver and software signing refer to chapter 2.8.

When the installation is finished a completion dialogue is displayed and the driver is now started automatically with every Windows start-up. The displayed device name depends on the CAN module. Press the **Finish** button to complete the installation.



If you now open the *Device Manager* window you will see that the CAN module is now listed below the new device class **CAN Interface**.



If you have installed several **esd** CAN modules attached to a local bus of your system you will find all of them here. By double clicking the device you will open the *Properties* dialogue where you can configure the device specific options described in chapter 2.2 via the *Settings* tab.

**Note:**

If you just want to run NTCAN based application on the system you are done.

If you intend to develop NTCAN based applications on this system you also have to install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

### 2.9.3.2 Software-First Driver Installation

Windows XP does not contain in-box tools to perform a software-first device driver installation. The process can be performed with the Windows Device Console (devcon). This manual does not contain a description of the software-first installation process for Windows XP but you can refer to this MSDN description how to use the tool to preinstall a device driver package.

### 2.9.3.3 Driver Lifecycle Management

The process of updating, rolling back or uninstalling a device driver package is very similar on all Windows versions and covered in chapter 2.5.

## 2.9.4 Windows 2000

This chapter covers the device driver installation for Windows 2000 and Windows Server 2000 (which is implicitly also referenced if the following text refers to Windows 2000).



**Attention!**

A user which wants to install a device driver must be member of the Administrators group.

### 2.9.4.1 Hardware-First Driver Installation

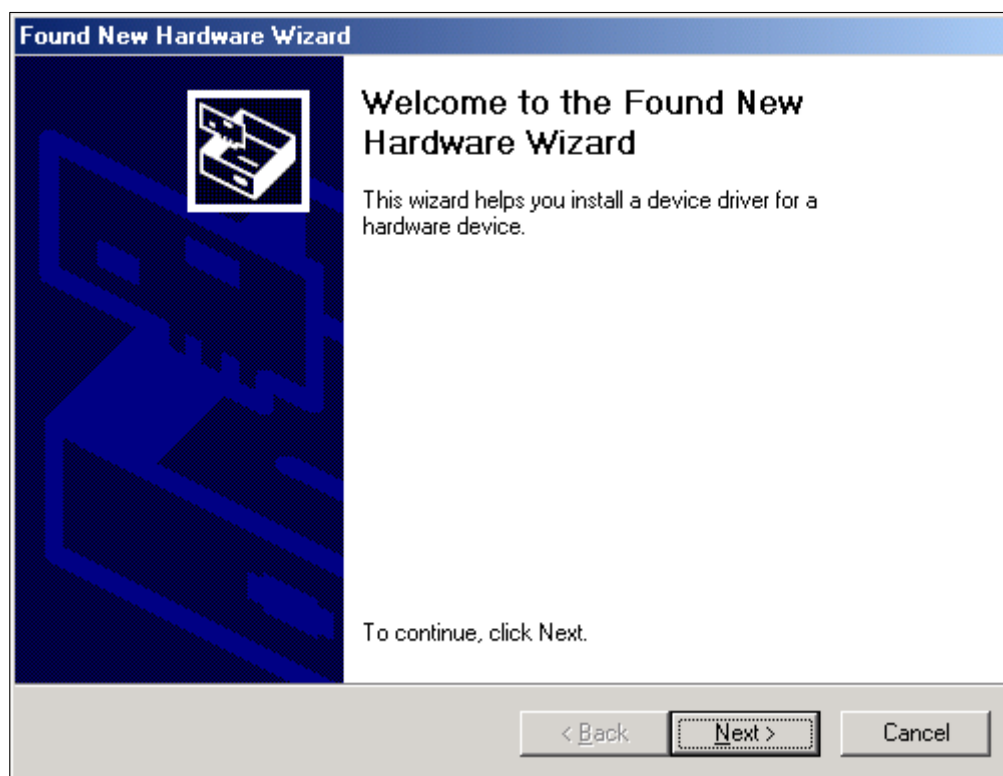
To initiate the device driver installation process, you have to connect the CAN module to your system. Depending on the *Hot Plugging* capability of the hardware you might have to shut down Windows for this. Please refer to the CAN module specific hardware manual for advises.



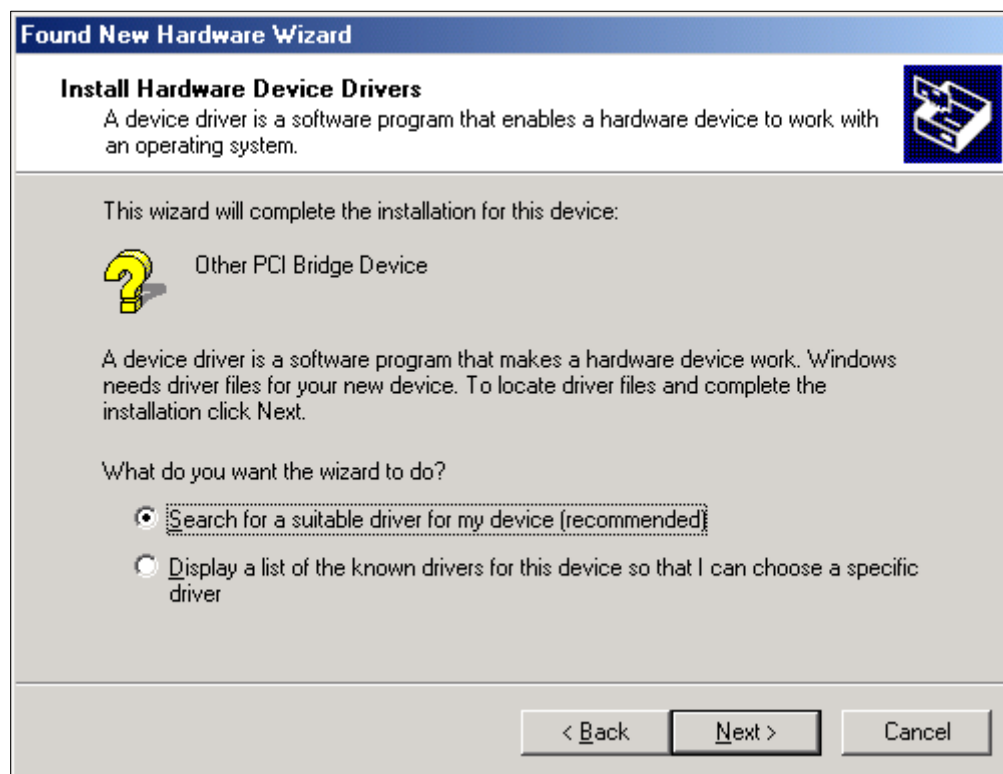
**Note:**

This chapter covers the installation of the device driver for *Plug and Play* capable CAN modules which are connected to a local bus (PCI, USB, ...) of your system. Non PnP **esd** CAN modules for legacy buses (ISA, Parallel Port,...) are supported by installing the legacy Windows NT driver which is described in chapter 2.9.4.4.

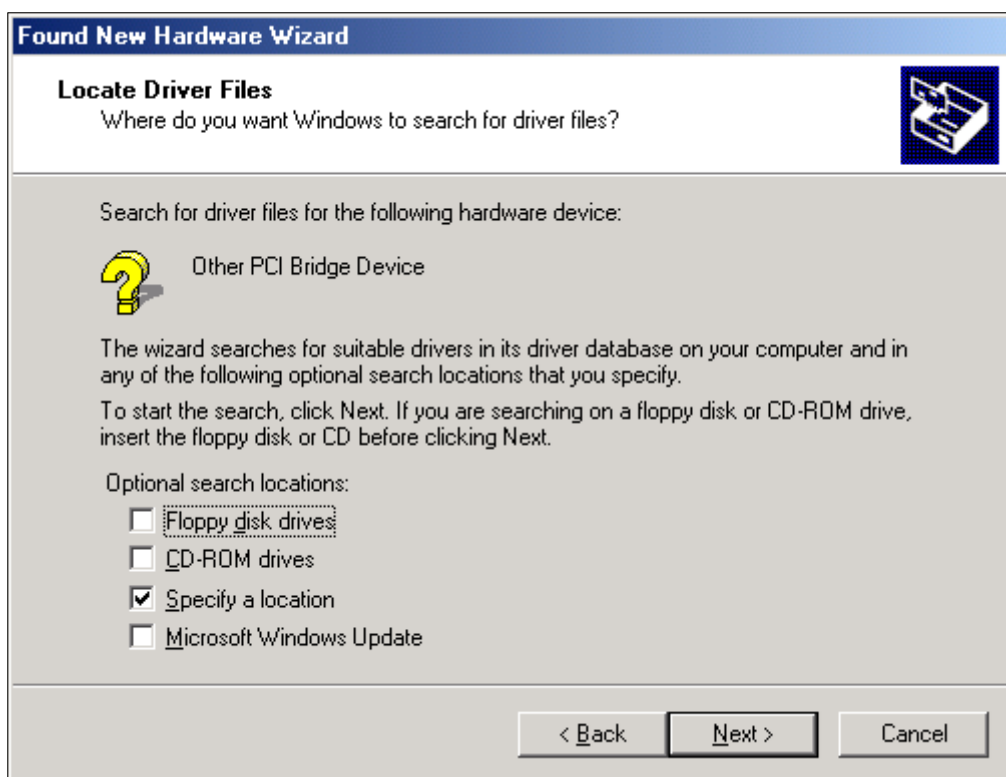
After Windows 2000 has detected the new hardware it will start the interactive driver installation process of the *Found New Hardware Wizard* with the dialogue below.



Choose *Search for a suitable driver for my device (recommended)* and click the Next button. The reported device type ("Other PCI Bridge Device") in this and the following dialogues depends on the CAN module.



Check the box *CD-ROM drives* if the **esd** CAN CD which accompanied the delivery of your CAN module is inserted into your optical drive or choose *Specify a location* if you install a driver downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu)). In the latter case browse for the location of the driver files on your hard disc where you have extracted the driver archive.



The *Found New Hardware Wizard* automatically selects the correct INF file, which again depends on the CAN module. Click *Next* to start the installation process.





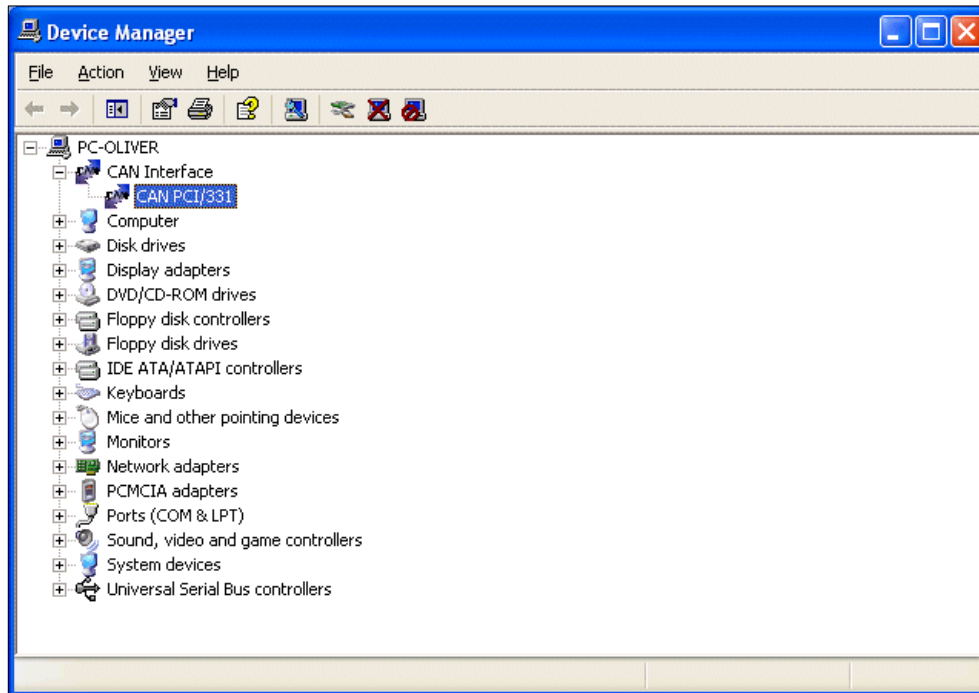
If your system is configured to warn when unsigned (non WHQL certified) drivers are about to be installed, a security alert dialogue will appear.



The complete CAN driver package is digitally signed and you will see a dialogue box, which indicates that Windows has verified that the device driver is released by **esd** and hasn't been tampered. As the driver packages are not signed by the WHQL (Windows Hardware Quality Labs) the Security Alert dialogue has to be confirmed with Yes to continue the driver installation. For more details about device driver and software signing refer to chapter 2.8.

When the installation is finished a completion dialogue is displayed and the driver is now started automatically with every Windows start-up. The displayed device name depends on the CAN module. Press the **Finish** button to complete the installation.

If you now open the *Device Manager* window you will see that the CAN module is listed below the new device class **CAN Interface**.



If you have installed several **esd** CAN modules attached to a local bus of your system you will find all of them here. By double clicking the device you will open the *Properties* dialogue where you can configure the device specific options described in chapter 2.2 via the *Settings* tab.



**Note:**

If you just want to run NTCAN based application on the system you are done.  
If you intend to develop NTCAN based applications on this system you also have to install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

### 2.9.4.2 Software-First Driver Installation

Windows 2000 does not contain in-box tools to perform a software-first device driver installation. The process can be performed with the Windows Device Console (devcon). This manual does not contain a description of the software-first installation process for Windows 2000 but you can refer to this MSDN description how to use the tool to preinstall a device driver package.

### 2.9.4.3 Driver Lifecycle Management

The process of updating, rolling back or uninstalling a device driver package is very similar on all Windows versions and covered in chapter 2.5.

### 2.9.4.4 Non-PnP hardware

There are no device driver for **esd** CAN modules for non PnP capable legacy buses (ISA, PC104, Parallel Port...) which support Windows 2000 in the same way as the device driver for PnP capable hardware.



**Note:**

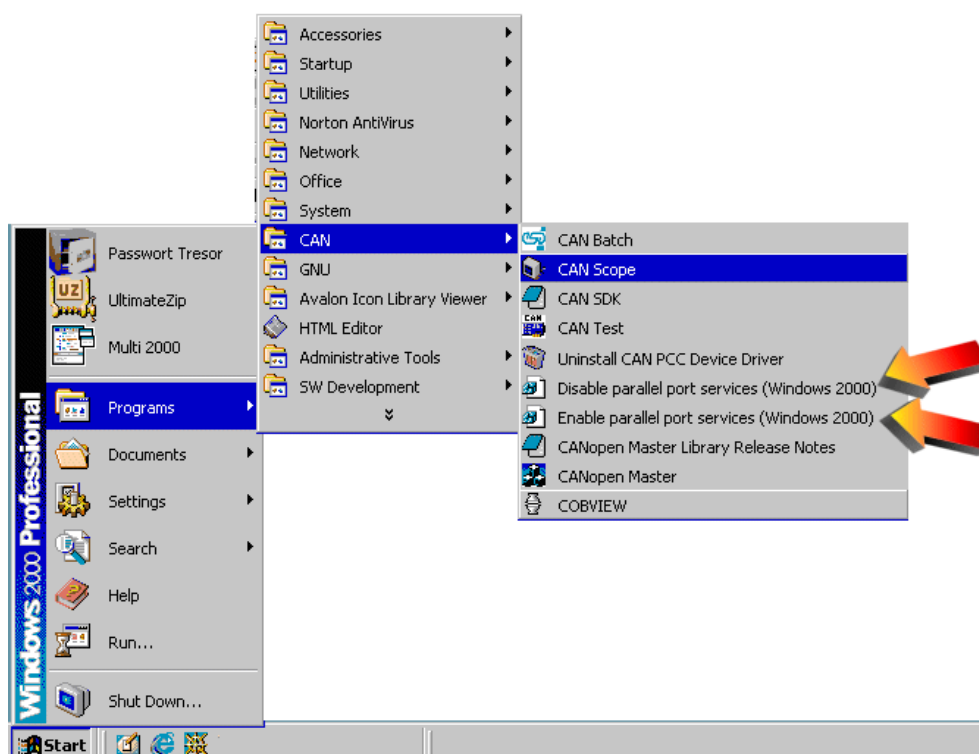
It is possible to use the Windows NT driver for CAN modules attached to legacy non-PnP capable buses. Please refer to chapter 2.9.6.2 for installation instruction.

The remaining chapter describes the necessary system changes to prevent conflicts with the in-box drivers of Windows 2000 which using the parallel port.

#### 2.9.4.4.1 Conflicts with In-Box drivers (CAN-PCC)

The CAN-PCC module is attached to the parallel port of a system. Windows 2000 comes with a driver which tries to enumerate hardware attached to the parallel port. This enumerator can not co-exist with the Windows NT device driver for the CAN-PCC. For this reason the Windows driver has to be stopped **before the CAN driver is started**.

Stopping this enumerator requires a change in the registry which can be accomplished via the *Programs/CAN/Disable parallel port services (Windows 2000)* entry via the start menu which is available after (Windows NT) driver installation.



**Note:**

You have to restart your system before the registry change can take effect.

Re-enabling the enumerator and the original state requires to revert the described changes made in the registry which can be accomplished via the *Programs/CAN/Enable parallel port services*

(Windows 2000) entry via the start menu.



**Note:**

You must restart your system before the registry change can take effect. If starting the CAN-PCC driver has been changed in the Device Manager from *Manual* to *Automatic*, do not forget to revert this change, too.

## 2.9.5 Windows NT 4.0

This chapter covers the steps of device driver install, configuration, start and uninstall for Windows NT 4.0.



**Attention!**

A user which wants to install/uninstall or configure a device driver must be member of the Administrators group.

### 2.9.5.1 Driver Installation

In case of an update, it is necessary to uninstall an older version before you start installing the new one as described in section 2.9.5.4.

You will find the device drivers on the **esd** CAN CD which accompanied the delivery of your CAN module in the folder \WinNT or they can be downloaded from the **esd** website ([www.esd.eu](http://www.esd.eu)). In the latter case extract the content of the driver archive to your disk.

To start the installation, you have to execute the program `setup.exe` for you CAN module and follow the steps of the installation program.

In the course of installation, you can choose between one of three configurations:

Configuration	Description
Typical	Driver, DLL, programs for firmware update and driver configuration.
Compact	Driver, DLL and configuration programs.
Custom	User-defined configuration.



**Note:**

After the installation and before the start of the device driver, the system has to be rebooted. Otherwise the driver will not work correctly.

After the reboot you can configure the device driver with the control panel applet **CAN Control** described in the following chapter.

**Note:**

If you just want to run NTCAN based application on the system you are done.

If you intend to develop NTCAN based applications on this system you also have to install the CAN SDK as described in chapter 2.7 which contains in addition to many tools the required header files, library files and/or wrapper for your development environment.

### 2.9.5.2 Driver Configuration

For the configuration of the driver the program **CAN Control** is installed together with the device driver. It can be started by double clicking the icon below in the Windows NT System Control.

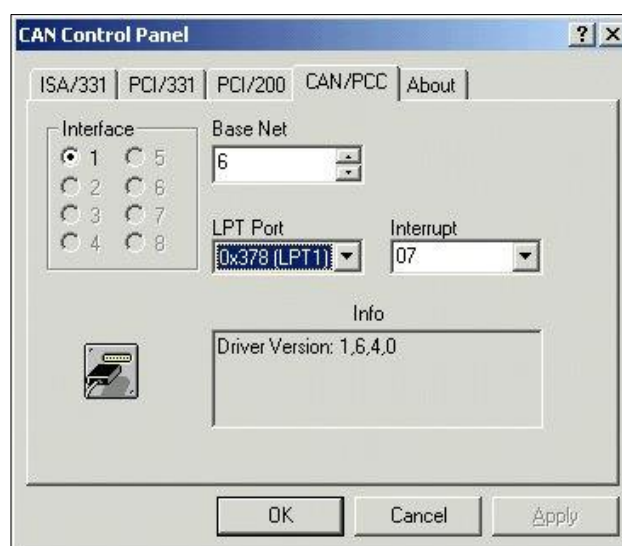
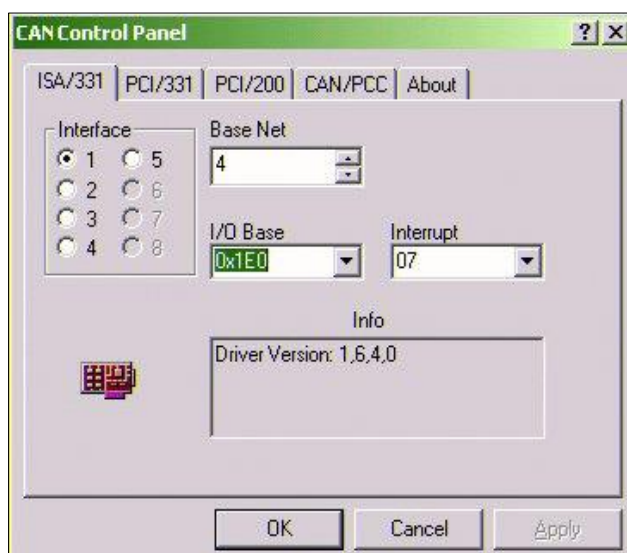


The picture below is an example for a configuration with four **esd** CAN modules (CAN-ISA/331, CAN-PCI/331, CAN-PCI/200 and CAN-PCC). **CAN Control** only shows the **esd** CAN modules which are installed on the system. If you have installed only one CAN module, only one configuration tab will be visible.

**Note:**

For CAN modules attached to local buses which are PnP capable (PCI, CPCI, PMC) you just need to configure the base net for non PnP devices you have to configure a base address and the interrupt used by the CAN module, too.

The pictures below show the configuration options for the non PnP CAN modules:



With the radio button in the *Interface* area you define the device instance for which the configuration has to be applied if you want to use more than one CAN module of the same type on your system.

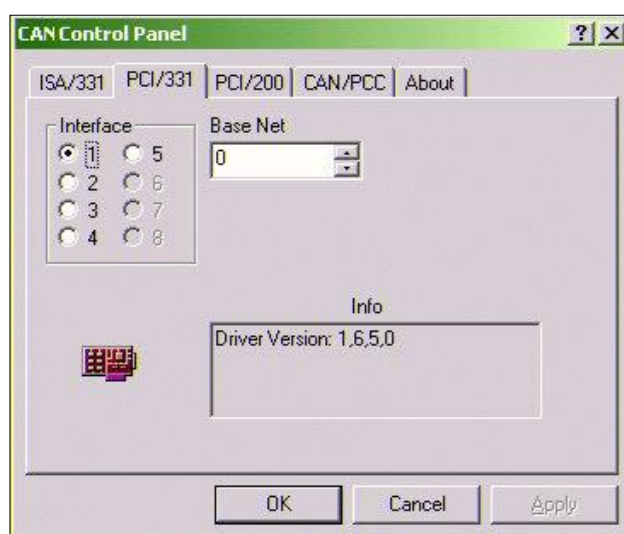
Define the *I/O Base* address and the *Interrupt* level according to your hardware configuration.

With *Base Net* a base network number is assigned to the CAN module which defines the logical net number used by the NTCAN-API. If a module has more than one physical CAN port, the first gets the logical network number specified in Base Net, and the following are counted up from there.

**Attention!**

The user has to make sure that the base net numbers are not assigned twice or overlap as otherwise the device driver will fail to start.

The picture below shows the configuration options for CAN modules attached to a PnP capable local bus. Only the Base Net has to be configured for these devices.



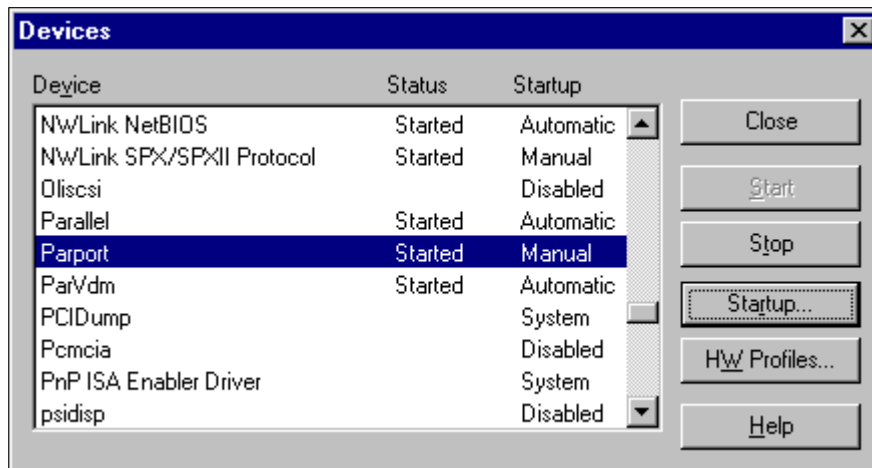
Complete the configuration pressing the **OK** button.

After the configuration you can proceed starting the driver as described in chapter 2.9.5.3.

### 2.9.5.2.1 Conflicts with In-Box drivers (CAN-PCC)

The CAN-PCC module is attached to the parallel port of a system. In the default configuration of a standard Windows NT 4.0 system the port is in use by the system. For an error free operation of the CAN driver, the Windows NT driver has to be stopped **before starting the CAN driver**. For this, you have to proceed as follows:

1. Select *System Control Panel* under *Settings*.
2. Here you have to double click the icon *Devices*. The following window will open:



3. In this window you now have to change the state of device *Parport* to stopped by pressing the **Stop** button. The system now reports that the devices *Parallel* and *ParVdm* are also stopped.
4. To prevent having to repeat this procedure with every system start, press the **Startup** button, and change the start type to *Manual* for the three devices *Parallel*, *Parport* and *ParVdm*.

### 2.9.5.3 Driver Start

To start the driver, the command `net start xxxx` (with xxxx as driver service name) has to be typed in a console windows.

The table below shows the service names for the supported **esd** CAN modules:

<i>CAN Module</i>	<i>Commands for Starting the Drivers</i>
CAN-ISA/200	<code>net start c200i</code>
CAN-PC104/200 (SJA1000 version)	
CAN-ISA/331	<code>net start c331i</code>
CAN-PC104/331	
CAN-PCI/200	<code>net start c200</code>
CAN-PCI/266	
CPCI-CAN/200	
PMC-CAN/266	
CAN-PCI/331	
CPCI-CAN/331	
PMC-CAN/331	
CAN-PCI/360	<code>net start c360</code>
CPCI-CAN/360	
CAN-PCC	<code>net start cpcc</code>

If the driver starts without errors, the start type of the driver can be changed via *System Control/Settings/Devices* from **Manual** to **Automatic**, so that the driver is started automatically with each system boot-up and can also be used by users without administrator rights.

If a problem occurs starting the driver, the error cause can be taken from the *System Event Log File* of Windows NT 4.0.



### 2.9.5.4 Driver Uninstall

The driver can be uninstalled in three steps:



**Attention!**

Stopping the driver and the complete de-installation can only be done with administrator rights on the Windows NT computer!

1. Terminate all applications using the CAN module.
2. The driver has to be stopped in dialogue control panel/devices or by entering `net stop xxxx` (with xxxx as driver service name) in the command line.

CAN Module	Commands for Stopping the Drivers
CAN-ISA/200	<code>net stop c200i</code>
CAN-PC104/200 (SJA1000 version)	
CAN-ISA/331	
CAN-PC104/331	
CAN-PCI/200	<code>net stop c200</code>
CAN-PCI/266	
CPCI-CAN/200	
PMC-CAN/266	
CAN-PCI/331	<code>net stop c331</code>
CPCI-CAN/331	
PMC-CAN/331	
CAN-PCI/360	<code>net stop c360</code>
CPCI-CAN/360	
CAN-PCC	<code>net stop cpcc</code>

3. Select entry for the CAN driver in the folder *control panel* in dialogue box *add/remove program properties* and press icon *add/remove*, in order to delete all files and registry entries of the driver.

### 2.9.6 Windows 9x/ME

The Windows 9x/ME device drivers support a maximum of 5 CAN modules of the same device family within one system. The number of different CAN modules in one system is not limited.



**Note:**

As Windows 95 does not come with USB support the CAN-USB/Mini module is not supported for this platform !

### 2.9.6.1 Installation of PnP CAN modules

This chapter describes the steps to install CAN modules for PnP capable local buses (PCI, USB, ...)

1. Install the board according to the hardware installation description.
2. After bootup Windows 9x/ME detects the board and opens a dialogue. If Windows has not found the driver (e.g. if it is the first installation), you are asked to insert the data carrier (disk or CD-ROM) with the driver.
3. At the end of the software installation, you have to shut down the computer. Terminate all active applications before you shut down the computer. After restart, the driver is automatically started and in the device manager the new device type *CAN Controller* now exists, which can be used to display and edit all **esd** CAN boards.
4. Now you can proceed with installing the SDK as described in chapter 2.7.

**Notes:**

CAN-PCI/360 modules which were manufactured before 01.01.2000 report their maximum memory upgrade of 128 Mbyte to the system during installation, even if they have got less capacity. These boards might get into conflict with graphics boards which act similarly. For these boards the actual memory space requirement can be configured via the Device Manager. More about the Device Manager can be found in chapter 'Changing the Resources Settings via the Device Manager' on page 99.

CAN-PCI/360-modules which were manufactured after 01.01.2000 do not have this problem anymore.

### 2.9.6.2 Installation of non-PnP CAN modules

This chapter describes the steps to install CAN modules for non-PnP capable local buses (ISA, Parallel Port, ...). After a successful driver installation, you can proceed with installing the SDK as described in chapter 2.7.

**Notes:**

For the installation of ISA and PC-104 CAN modules you have to find an unused I/O base address with the help of the Windows 9x/ME system control and have to configure this I/O base address on the hardware (please refer to your hardware manual for details).

**Notes:**

Starting with driver revision 1.2.0 you can select between a driver for 11-bit identifiers (CAN 2.0A) and a driver which also supports 29-bit identifiers (CAN 2.0B) during the installation of **CAN-ISA/200** and **CAN-PC104/200** modules. The latter driver is slightly slower than the CAN 2.0A version, even if you only use it to transmit and receive 11-bit-identifiers, because more I/O requests to the CAN controller are necessary.

Prerequisite for the CAN 2.0B mode is that you do not have a board with the CAN controller Philips 82C200 (delivered until 12/1999), but a board with a Philips SJA1000 instead.

### 2.9.6.2.1 Installation of the Device Drivers

#### 1. Call

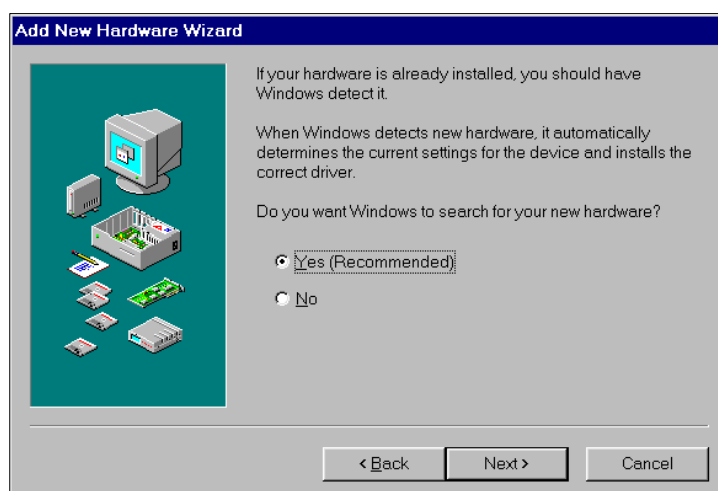
The Windows-9x/ME driver of the CAN-ISA boards is installed by means of the Hardware Wizard. It is started in the System Control Panel window by selecting the Hardware icon:



After that the start window of the hardware wizard has to open:



- The following window asks if the hardware is to be searched for. Since the board is unknown to the system (no plug-and-play), **No** must be selected. If Yes is selected, the computer will search for the board for a relatively long time and then report the discovery of an unknown board.



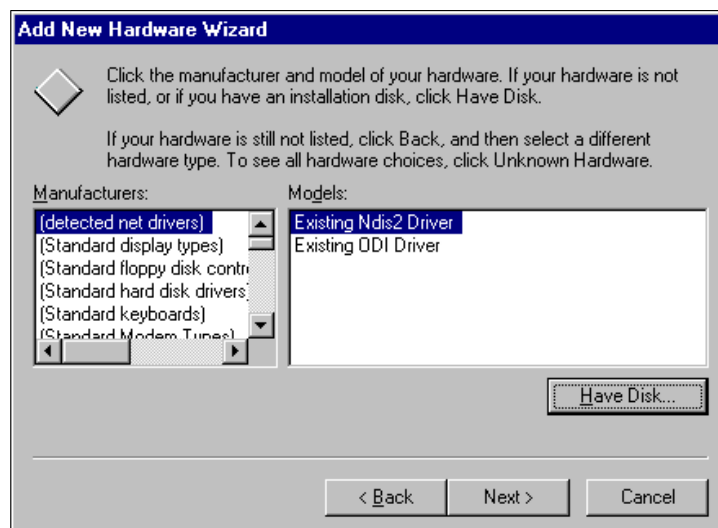
- Then the window for the selection of the type of hardware to be installed appears.



Here *Other devices* has to be selected and then *Next* has to be clicked. (Only if already an **esd** CAN driver had been installed, the selection *CAN Interface* would appear. It is not shown before or during the installation).

**If you haven't already done so, you should now put the data carrier (disk or CD-ROM), contained in the product package, into your drive.**

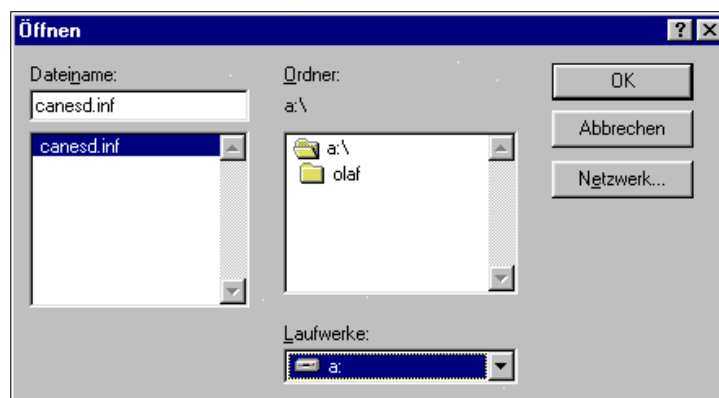
4. In the window which opens after the hardware-type selection first *Have Disk* and then *Next* have to be clicked.



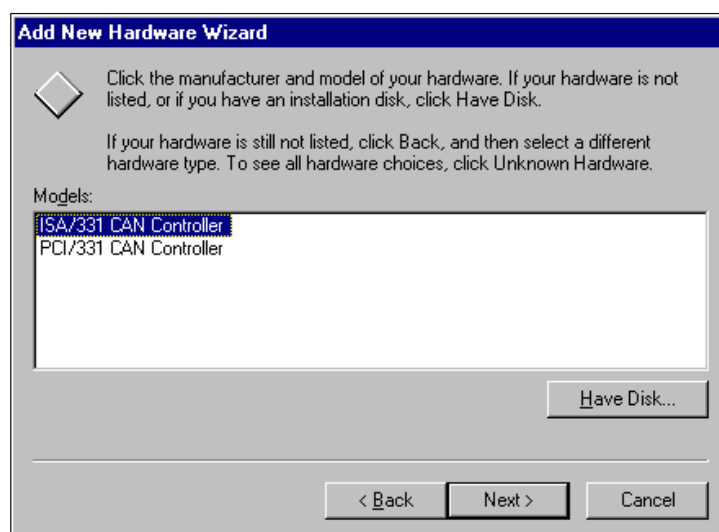
5. The following window appears. Click the button *Find*.



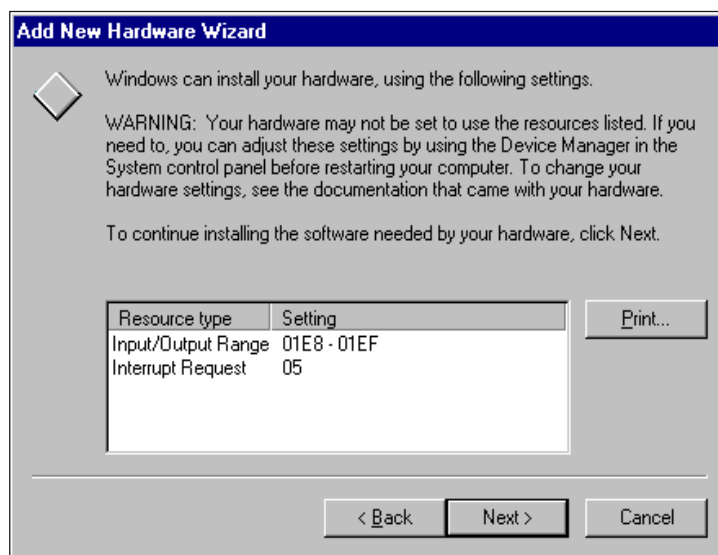
6. Now select the file `canesd.inf` and click *OK*.



7. The following window, which displays all **esd** CAN drivers, has to open (this example only shows the ISA/331 driver). Select the driver of your board and click *Next*.



8. (Only for CAN-ISA boards, otherwise continue with step 9) Windows 9x/ME installs the driver, checks the system resources, compares them to the configurations possible for the CAN-ISA board and offers a possible configuration for the board in the following window:



The *Input/Output* range proposed by the system control does not have to correspond to the default-I/O range which is set on board by means of jumpers or coding switches.

**Attention!**

Therefore it is absolutely necessary to compare the jumper (or coding switch) position on the CAN-ISA board to the I/O-address space selected by the system, and change the jumpers/coding switches, if required.

Click *Next*

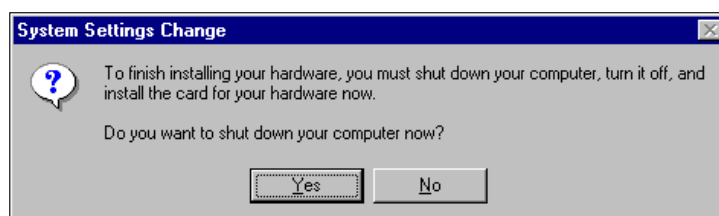
**If the system does not propose settings:**

If the system says that it does not have any resources for the CAN-ISA board, the installation has to be completed, nevertheless. In this case the resources in the system have to be distributed again manually by means of the device manager after the installation.

9. The successful installation of the software driver is shown by the following window:



10. In order to complete the software installation, you have to shut down your computer after terminating all applications which are still open!



Then switch off the computer and **install the hardware now** as described in the hardware manual! Compare the default setting to the setting selected under point 8 and change the jumpers (or coding switches, depending on board type), if necessary.

11. Switch on the computer again after the installation and restart Windows 95. The driver is automatically loaded by Windows 95. The device manager now has the new device class *CAN Controller* under which all **esd** CAN boards can be shown and configured.

### 2.9.6.2.2 Starting the Device Drivers

#### ISA-Bus Hardware

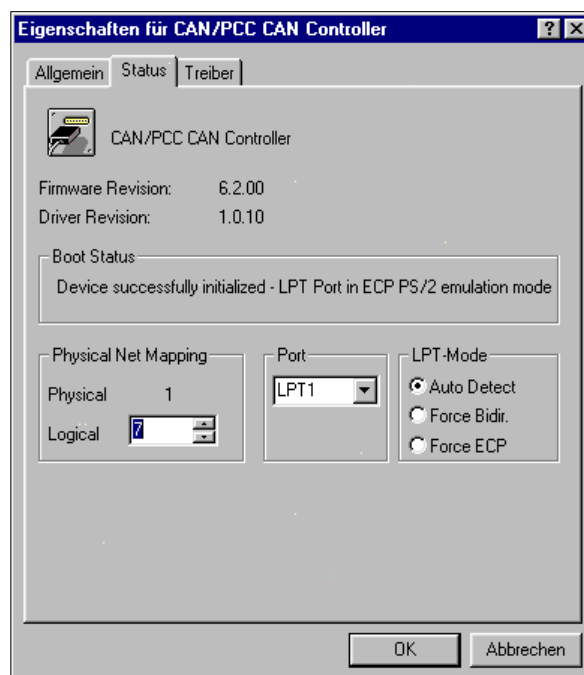
Switch on the computer again after installation and restart Windows 9x/ME. The driver is automatically loaded and started by Windows 9x/ME. In the device manager there is now the new device class *CAN Controller*, under which all **esd** CAN boards can be shown and configured.

#### CAN-PCC

Switch on the computer again after installation and restart Windows 9x/ME. The driver is automatically loaded by Windows 9x/ME but **not** started. In the device manager, however, the new device class *CAN Controller* exists, under which all **esd** CAN boards can be shown and configured.

For starting the driver, the command `cpcc start` has to be entered explicitly in the command line. This is necessary to be able to share the parallel interface with other device drivers. After successfully starting the driver, it has an exclusive access to the parallel interface until it will be stopped again by means of `cpcc stop`.

Another reason for executing an explicit start command is that the driver is not being assigned with its own resources within Windows 9x/ME, but that it uses the available resource 'Parallel Interface'. Therefore, no smart icon *Resources* (see the following chapter) exists for this driver within the device manager. In smart icon *Status* the parallel interface to be used (LPT1, LPT2, LPT3) has to be entered below port instead and the driver automatically uses the current Windows-9x/ME configuration (I/O-address, interrupt) for this interface.

**Attention!**

For the operation of the CAN-PCC under Windows 9x/ME the parallel port must be set to operating mode Force ECP in the properties window. If the hardware does not support this setting, the bidirectional mode has to be selected (Force Bidir.).



### 2.9.6.2.3 Changing the Resource Settings

#### Overview

**Attention!**

Changing the default settings by the device manager or the registry editor can cause conflicts so that one or more devices are not recognized by the system anymore! The device manager and the registry editor are configuration tools for advanced users who are familiar with the parameter configuration and know that changes can have various effects.

The CAN-ISA boards have fixed resource settings which are either distributed by the hardware assistant during the Windows setup or which can be configured later by the device manager.

It might occur that Windows 9x/ME is unable to configure the CAN-ISA board, because it gets into conflict with other devices. Should this be the case the board has to be configured again.

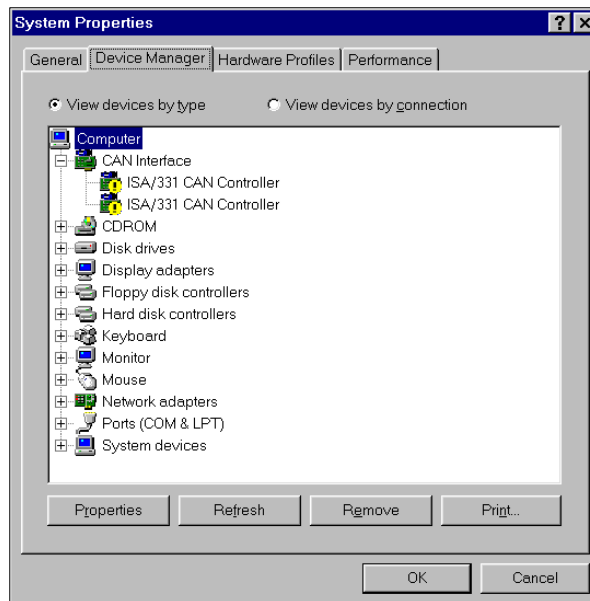
To change the device setting manually, the *Device Manager*, which is called *via Settings* under *System Control Panel*, can be used. By using the device manager errors can be prevented which are likely to occur when editing the registry entries directly.

If you want to solve the device conflicts manually by means of the device manager, you can use following strategies, for instance:

- If the conflicting device is a plug-and-play device, deactivate it to release its resources.
- If the conflicting device is a conventional device, deactivate it by removing the board from the system and unloading the drivers.
- Distribute the resources which are used by other devices again in order to release resources for the conflict device.
- Change the jumpers (or coding switches, depending on the board type) on the CAN-ISA board to adjust the board to the new settings.

## Activating the Device Manager

1. Click icon of *Device Manager* in *Settings* under *System Control Panel* or click *MyComputer* with right mouse button, click *Properties* in *Context* menu and then click the *Device Manager*.



2. Double clicking the desired device type in the list with the left mouse button lists all devices of this type in the computer.
3. Select the device to be configured by double clicking. Or else mark the device and click the *Properties* icon.

## Changing the Resource Settings by Means of the Device Manager

1. Select the device class *CAN Controller* in the device manager by double clicking. The tree branches and shows all devices of this type available in your computer.
2. Double clicking a device opens its property window. Click the Resources icon under the device properties.

The *Conflicting Device List* shows the settings of other devices which are conflicting the current setting of the CAN-ISA board.

3. Select the setting, which is to be changed, for instance the interrupt level, under *resource type*. Click the *Change setting* icon to keep the changed values. Changes can only be made if the option *Use automatic settings* has been deactivated. The Interrupt and *Input/Output-range* (address space) settings can be changed independently.

The dialogue box *Input/Output range* shows the various settings which are supported. An interrupt which is marked by an asterisk (\*) signifies that this interrupt is already used by another device.

After clicking the *Change setting* icon an error message might appear which states that the resource setting cannot be changed. In this case you must select other settings until the system accepts one of the chosen settings.

4. Select settings which are not conflicting other devices and click 'OK'.

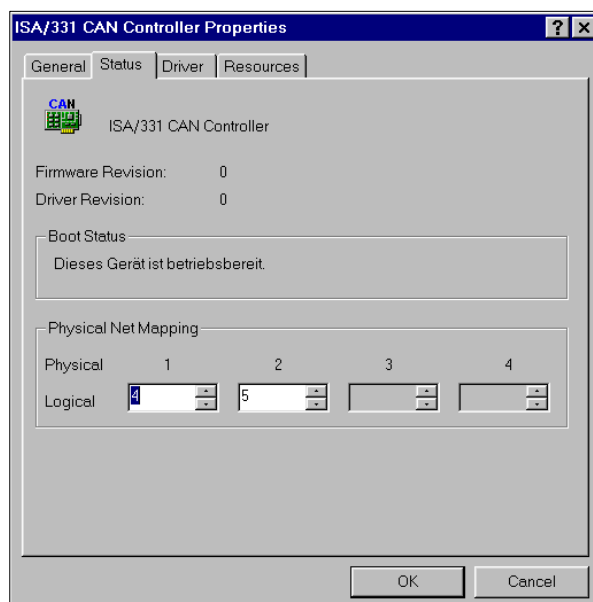
**Note:**

Remember that the I/O-range setting by jumper (or coding switches, depending on the board type) has to comply with the selected setting!

5. Finish Windows 9x/ME, then change the hardware settings of devices which have been configured again and restart Windows 9x/ME.

## Changing the Logical Network Number

1. Double click the device class *CAN Controller* in the device manager. The tree branches and all devices of this type available in your computer are shown.
2. Double clicking a device opens its property window. Click the *Status* icon in the property window.
3. In *Physical Net Mapping* logical network numbers can be assigned to the physical interfaces of the CAN board. By means of the network numbers the CAN board can be addressed by the software. When starting the driver for the first time, logical network numbers starting from 0 are assigned to boards supported by the driver. If more than one **esd** CAN board of various types (e.g. CAN-PCI/331 and CAN-ISA/331) are used in a computer, overlapping network numbers cannot be avoided and therefore must be set manually.
4. A change in logical network numbers remains invalid until the computer is restarted.



# 3 Unix® Operating Systems

This chapter covers the necessary steps to install, configure and start the device drivers for **esd** CAN interfaces available for UNIX® operating systems.

## 3.1 Linux®



### Note:

For current information on the installation please check the readme in the according installation directory. For the changes introduced with the last service pack, please check the release notes.

Depending on the CAN hardware NTCAN support is either realized by device driver packages provided by **esd** (see chapter 3.1.2) or by the standard framework for CAN driver support in Linux (aka *SocketCAN*) together with a wrapper library (see chapter 3.1.3). The EtherCAN interface family requires a user mode driver which installation and configuration is described in chapter 3.1.4.

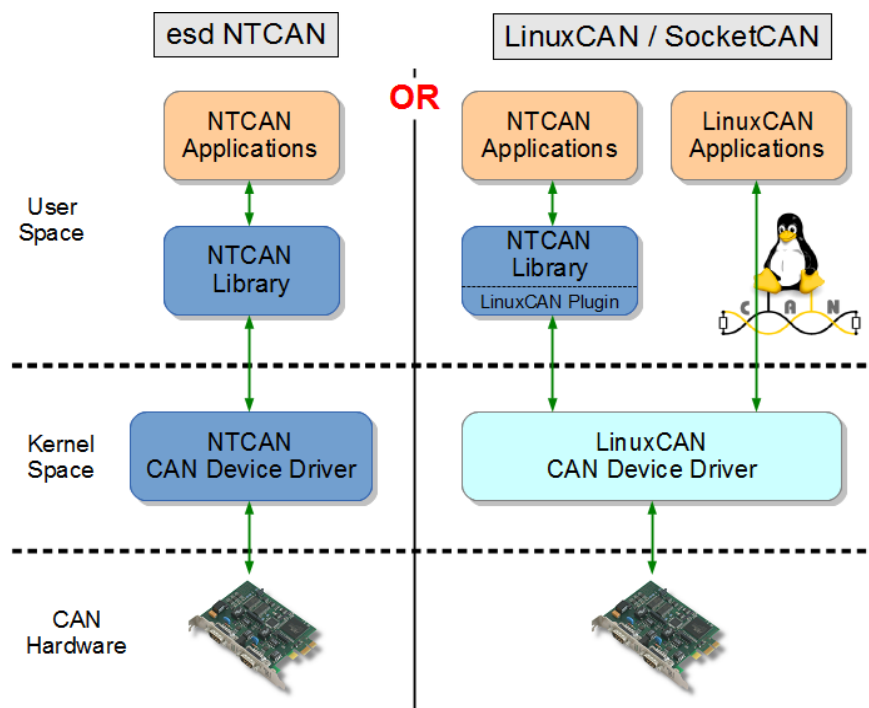


Linux CAN support (aka *SocketCAN*) is a set of *Open Source* drivers and a network stack which extends the BSD socket API in Linux by introducing the new protocol family **PF\_CAN**. Since version 2.6.25 this framework is part of the vanilla (mainline) Linux kernel and can be included by compiling the kernel with **CONFIG\_CAN**.

An introduction and overview about the Linux CAN implementation can be found in a presentation during the International CAN Conference (iCC) 2012 with the title “The CAN networking subsystem of the Linux kernel”.

Refer to Table 13 which esd CAN interfaces come with a native SocketCAN support.

The picture below gives an overview about using the **esd** NTCAN architecture compared to using the SocketCAN architecture together with the NTCAN wrapper.



**Figure 1:** Native NTCAN Architecture vs. Linux CAN



Using the SocketCAN NTCAN wrapper requires that the SocketCAN driver for the **esd** CAN interface is working.

As this driver is maintained by the Linux (CAN) community the appropriate community support mechanisms like mailing lists, etc. should be consulted in case of installation problems for the SocketCAN part.

Problems with the NTCAN driver or the SocketCAN wrapper are handled by the **esd** support.

The CAN drivers provided by **esd** are intended for the standard Linux (vanilla) kernel. Linux real-time extensions (apart from the real-time preempt patches aka **PREEMPT\_RT**) like RTAI, Xenomai, etc. are not officially supported.



**Xenomai** is a real-time development framework cooperating with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time support to user-space applications.

Part of the framework is the RT-Socket-CAN environment which supports hard real-time for CAN communication with **esd** CAN interfaces of the C200 CAN interface family. Please refer to the project's homepage for details about installation and API.

**!! There is no NTCAN API and installation support for Xenomai by esd !!**

Please contact **esd** for options to support further CAN interface families.

### 3.1.1 CAN Board Support Overview

Device drivers for **esd** CAN interfaces are available for x86 (32-Bit) and x64 (64-Bit) target architectures. The CAN interfaces are supported either by **esd** NTCAN driver or by LinuxCAN / SocketCAN and in some cases even by both architectures. The table below gives an overview which CAN interface family is supported by which driver architecture.

CAN Family	Driver Architecture	
	<b>esd NTCAN</b>	<b>SocketCAN</b>
C200I	✓	-
C331I	✓	-
C200	✓	✓
C331	✓	-
C360	✓	-
C400	✓	-
C402	✓	-
C405	✓	-
USB1*	✓	-
USB2***	-	✓
USB3†	-	✓
U400	-	-
EtherCAN	✓	-
AMC4	✓	-

**Table 13:** SocketCAN support for esd interfaces



#### Attention!

If a CAN interface is supported by LinuxCAN / SocketCAN and you want to use the **esd** NTCAN driver you must make sure that the device is not already used by the LinuxCAN / SocketCAN driver as otherwise starting the **esd** CAN driver will fail.

\*Support ends with kernel version 2.6.24 for legal reasons.

\*\*No support for CAN-USB/AIR2.

† Support is integrated in Linux kernel since kernel version 6.6.

In addition to the CAN interfaces **esd** also provides Linux BSPs for several embedded boards with on-board CAN interfaces. These boards are supported by NTCAN driver architecture which is part of the BSP. The table below gives an overview about the availability for different Linux versions and the included CAN driver version.

<i><b>Board</b></i>	<i><b>Linux</b></i>	<i><b>CAN Driver Version</b></i>
PMC-CPU/405(-DE)/440	>=3.2.0	3.x
CPCI-CPU/750	>=2.6.36	3.x
CPCI-405/EPPC-405	>=3.2.0	3.x
EPPC-405-UC	>=3.2.0	3.x
CAN-CBX-CPU52xx	>=3.2.0	3.x
CPCI-CPU/5201	>=3.2.0	3.x



### 3.1.2 NTCAN Driver

**Note:**

On most Linux installations the driver installation is only possible with superuser rights.

Please read the current `README` file that comes with the software!


Please note the drivers delivered on CD are most likely outdated. Increasing speed in Linux kernel development makes it almost impossible for us to provide you with drivers on CD, which work with all Linux versions and distributions. We rather recommend checking our website for the latest Linux drivers. To circumvent any problems before they occur, we advise you to visit this site before installing a driver from this CD.

The latest driver archives can be downloaded from <https://www.esd.eu>

Please note: Drivers released before July 2012 still come as encrypted ZIP archives. Newer drivers are released in the more commonly used TGZ format.

### 3.1.2.1 Files of the Linux Package

The software drivers for Linux are distributed on CD-ROM or delivered as archive via e-mail. The following files are contained:

File	Description
README	current notes and information
Makefile	driver compilation rules
config.mk	configuration file for the compile process It may be necessary to edit this file, in order to suit any peculiarities of the current system (mainly the <i>KERNELPATH</i> , see <b>page 111</b> )
libntcan.a	static CAN-API library (located in directory <i>./lib</i> )
libntcan.so	dynamic CAN-API library (located in directory <i>./lib</i> )
ntcan.h	header of the NTCAN-API/Library (located in directory <i>./lib</i> ) This is the only header that has to be include in your application. Please do not use any defines located in any of the other headers, in order to keep your applications working with future version of the driver!
cantest.c	source code of the example-application 'cantest' (located in subdirectory <i>./example</i> ) (see <i>/1/</i> )
cantest	binary of example-program 'cantest' (located in subdirectory <i>./bin</i> )
xxxx.o xxxx.c xxxx.h	source- and object-files (located in subdirectory <i>./src</i> ) This driver is released as a combination of binary-objects (*.o) and source-files (*.c and *.h). This way <b>esd</b> can provide a CAN-driver working with many different Linux-kernels. The source files are NOT under the GPL (GNU Public Licence)! You are not allowed to modify, redistribute or sell the files! They are intellectual property of esd electronics gmbh. <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"><b>Attention!</b> Do not try to use any defines or data-structures located in these files in your own sources. This will lead to non-working applications in the future.</div>
updcrd	This tool is only delivered with CAN modules that are equipped with a local processor (e.g. CAN-PCI/331). It is located in subdirectory <i>./bin</i> . This tool can be used to switch the firmware of such a card between CAN-2.0A-firmware (used for reception of CAN-messages with 11-bit-identifier) and CAN-2.0B-firmware (used for additional reception of CAN-messages with 29-bit identifier). Syntax: <code>updcrd -tx net</code> Parameter: <code>crd:</code> CAN module ID, e.g. pci331, usb331 (see table on <b>page108</b> ) <code>x:</code> 'a', if CAN 2.0A firmware 'b', if CAN 2.0B firmware <code>net:</code> Net number of the CAN interface in the system (0, 1, 2, ...)



**Note:**

In driver archives for x86\_64-Linux the path for libraries and binaries exists twice: Once for 32-bit (*./lib32* and *./bin32*) and once for 64-bit (*./lib64* and *./bin64*).

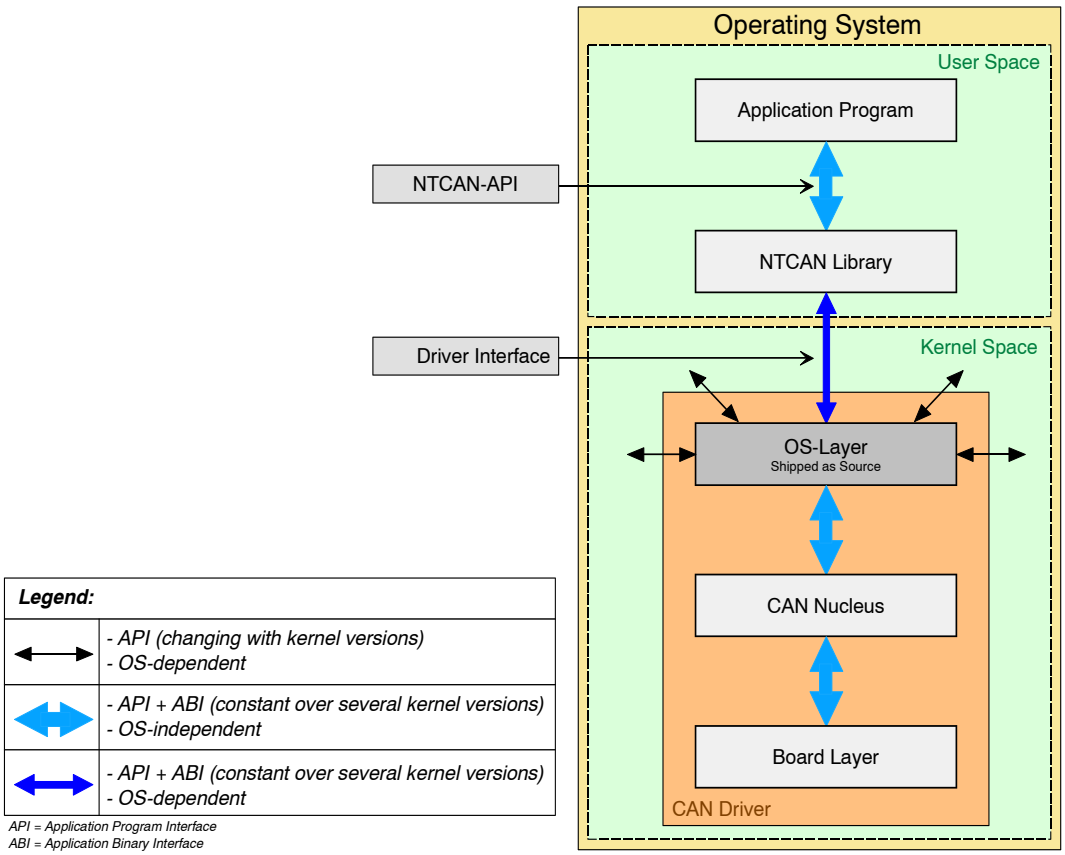


Figure 2: Linux driver architecture

### 3.1.2.2 CAN-Module-ID and Default Parameters of the Driver

CAN Module	Module ID <i>crd</i>	Default Values*		
		<i>major</i>	Address <i>io</i>	Interrupt <i>irq</i>
AMC-CAN4	amc4	54	–	–
CAN-ISA/200	isa200	53	0x1E8	7
CAN-PC104/200 (SJA1000 version)				
CAN-ISA/331 CAN-PC104/331	isa331	52	0x1E0	5
CAN-PCI104/200 CAN-PCI/200 CAN-PCle/200 CPCI-CAN/200 CAN-PCI/266 PMC-CAN/266	pci200**	54	–	–
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	pci331	50	–	–
CAN-PCI/360 CPCI-CAN/360	pci360	51	–	–
CAN-PCI/400 CAN-PCle/400 CPCI-CAN/400 PMC-CAN/400	esdaccbm	55	–	–
CAN-PCI/405	pci405	53	–	–
CPCI-405 (local driver)	cpci405	53	–	–
CPCI-CPU/750 (local driver)	cpci750	53	–	–
CAN-USB/Mini	usb331	50	–	–

\*The default values can be overwritten by the command `insmod` (see following chapter).

\*\*Before installing the NTCAN driver for `pci200` please check, that the CAN interface is not yet used by the SocketCAN driver.

### 3.1.2.3 Installation



**Note:**

It is necessary to install the kernel sources and configure them to comply with the running kernel, before installing the CAN-driver!

#### Unpacking the Archive

##### Unpacking the TGZ archive (for drivers released after July 2012) with

```
tar -xzf esdcan-crd-os-arch-ver-ext.tgz
```

*crd* = card-id (e.g.: *pci200* or *cpci405*, see table in chapter 3.1.2.2)

*os* = host-operating-system (e.g.: *linux\_2.4.x*)

*arch* = host-architecture (e.g.: *x86* or *x86\_64*)

*ver* = driver version (e.g.: *3.7.2*)

*ext* = extension (applicable to certain cards only, e.g.: *gcc2*)

You'll end up with a directory named as the archive.

##### Unpacking the ZIP archive (for drivers released prior July 2012) with

```
unzip esdcan-crd-os-arch-ver-ext.zip
```

*crd* = card-id (e.g.: *pci200* or *cpci405*, see table chapter 3.1.2.2)

*os* = host-operating-system (e.g.: *linux\_2\_4\_x*)

*arch* = host-architecture (e.g.: *x86* or *x86\_64*)

*ver* = driver version (e.g.: *3.7.2*)

*ext* = extension (applicable to certain cards only, e.g.: *gcc2*)



**Note:**

You will be prompted for a password. The password can be found in the accompanying file `README_FIRST` (delivered on the CAN-CD or per e-mail).

**Resulting file:** `esdcan-crd-os-arch-ver-ext.tar`

### Untar the Driver Directory

```
tar -xvf esdcan-crd-os-arch-ver-ext.tar
```

*crd* = card-id (e.g.: *pci200* or *cpci405*, see table chapter 3.1.2.2)  
*os* = host-operating-system (e.g.: *linux\_2\_4\_x*)  
*arch* = host-architecture (e.g.: *x86* or *x86\_64*)  
*ver* = driver version (e.g.: *3.7.2*)  
*ext* = extension (applicable to certain cards only, e.g.: *gcc2*)

You will end up with a directory named as the archive.

### Unzip and Untar in a Single Step

Alternatively, the unzipping and untaring of the driver directory can be accomplished in a single step with:

```
unzip -p esdcan-crd-os-arch-ver-ext.zip | tar -xv
```

The unpacked files will be stored in a directory that carries the same name as the archive file.

## Compiling the Driver

```
cd ./esdcan-crd-os-arch-ver-ext
```

In some cases, you need to edit a configuration file for the compilation: In `config.mk` you need to set the variable `KERNELPATH` correctly. Normally the default path should be correct. If your Linux configuration differs from standard, correct the following line accordingly:

```
KERNELPATH = <your-path-to-the-kernel-source>
```



**Note:**

For Linux kernel > 2.6.0:

On some systems you might need to be "root" to compile the driver.

Compilation of the driver is simply started by typing:

```
make
```

For some cards there are warnings like `COMPILING FOR xxx`. These can be ignored and will be removed in future versions.

Now, you have a file called as described below, which is the actual driver-module in the same directory:

```
esdcan-crd-os-arch-kver
```

Dynamically loadable driver-file with:

`crd` = card-id (e.g.: `pci200` or `cpci405`, see table chapter 3.1.2.2)

`os` = host operating system (e.g.: `linux...`)

`arch` = host architecture (e.g.: `x86`)

`kver` = target-version information (e.g.: `2.4.18`)

For Linux the kernel version is coded here, because the compiled version is kernel specific!

**Example:**

For a CAN-PCI/331 for 32-Bit-Linux on x86 with 2.4.21-99-smp kernel the driver with version 3.6.1 is called as following:

```
esdcan-pci331-linux-x86-3.6.1-2.4.21-99-smp
```



**Note:**

For Linux kernel > 2.6.0:

The driver file is called `esdcan-crd.ko` and is generated inside of the `src-` subdirectory.

**Example:** (for Linux kernel > 2.6.0)

For the above mentioned CAN-PCI/331 the driver file is called:

`esdcan-pci331.ko` (the file is located in `./src` subdirectory)

### 1. File Locations

It is recommended (though not necessary) to store the driver module in the following directory:

```
/lib/modules/kernelversion/
```

The variable `kernelversion` has to be replaced by the according string of the system. The string (on kernels 2.4.x it should be equivalent to the `os`-string in the driver's name (see above)) is returned, if the following command is called:

```
uname -r
```

The dynamic shared library `libntcan.so` should be placed in the directory `/usr/local/lib/` or an equivalent path, which is contained in the `LD_LIBRARY_PATH` environment variable.



**Note:**

On 64-bit systems, there are two versions of `libntcan.so`. One in `./lib32` and one in `./lib64`. The first belongs into `/usr/local/lib` on most Linux distributions. The later should be kept together with other 64-bit libraries, e.g. in `/usr/local/lib64`.

The static version of the library `libntcan.a` can be kept wherever you want. Here at **esd** we prefer to keep it with the sources of a project, on the other hand, one might like to install it with the shared-lib at `/usr/local/lib/`.



**Note:**

**esd** discourages the use of statically linked libraries. We rather recommend to make use of the dynamically linked libraries. Your advantages will be much easier backtracking of involved versions and much simpler update procedures.



**Installation Note:**

The shared library should belong to user and group "root" with the following file access permissions: `u=rwx, g=rx, o=rx`

After installation of the library, the root-user should call:

```
ldconfig -n /usr/local/lib (if installed to this directory)
```

Afterwards there is a link `libntcan.so.v --> libntcan.so.v.mv.r`.

For your own convenience it is advised to generate another link in your library-directory:

```
libntcan.so --> libntcan.so.v
```

The static-library, if installed in `/usr/local/lib/`, should also belong to user/group root, but it does not need (and should not have) the executable-flag. Leading to the following file access permissions: `u=rw, g=r, o=r`



## 2. Load the Driver File (as Superuser)

Syntax:

```
insmod ./esdcan-crd-os-arch-kver [major=m]
```

with the following optional parameter: *m* = non-default major

The naming of the kernel module is equivalent to the naming of the driver archive as printed on page 108 (exception on kernels > 2.6.x).



**Note:**

With the module CAN-PCI/405 this call returns after approx. 5 seconds!

## 3. Make the 'inodes' (as Superuser)



**Note for Systems with Kernel 2.6.x:**

On 2.4.x systems this step has to be executed just once. On 2.6.x systems the inodes might vanish after reboot. If this is the case on your system, please do the following:

Instead of `/dev` rather change into `/lib/udev/devices` directory and create the inodes there. In this way they will be automatically recreated on every reboot.

```
cd /dev
```

```
mknod --mode=a+rw can0 c xx 0
```

```
mknod --mode=a+rw can1 c xx 1    ==>  as many as physical CAN nets provided
                                     by the modules
```

with

*xx* = major number of the driver (see table on **page 108**)



**Note:**

For your own convenience as soon as the driver has been loaded (Step 4 complete) there is a script in proc-filesystem (`/proc/bus/can/XXX/inodes`, where *XXX* is a subdirectory depending on your CAN device), which relieves you of this step. The script also has a parameter to specify the starting net number. It will handle this step for multiple CAN devices as well.

### 6. Checking the Installation

Whether the installation has been successful or not, can be checked in the following file:

`/var/log/messages`

or by calling `dmesg`

Here is an example for a successfully loaded driver for CAN-PCI/405:

```
kernel: esd CAN driver: pci405
kernel: esd CAN driver: baudrate not set
kernel: esd CAN driver: mode=0x00000000, major=53, 4 nodes on 1 cards
kernel: esd CAN driver: version 0.3.1 14:36:35 Feb 13 2003: successfully loaded
```

After a successful installation, the CAN bus can be accessed by means of the NTCAN-API. The application has to be linked to the library `libntcan.a` (static) or `libntcan.so` (shared).



**Note:**

esd strongly recommends usage of dynamically linked libraries (aka shared objects)!

If the example application `cantest` is called without parameters, the available CAN nets are displayed.

### 7. Unload the Driver File (as Superuser)

Kernel 2.4.x:

```
rmmod esdcan-crd-os-arch-kver
```

Kernel 2.6.x:

```
rmmod esdcan-crd.ko
```

### 3.1.3 Linux CAN Driver (aka SocketCAN)

#### 3.1.3.1 Integration

The NTCAN library plugin capability is used to provide NTCAN support for **esd** CAN interfaces that are supported natively with the CAN driver implementation which is part of the Linux kernel usually referred to as *SocketCAN*. The plugin approach provides support for most basic features of the NTCAN API (see chapter 3.1.3.4 for limitations and differences) so NTCAN based applications can run in parallel with SocketCAN based applications and the hardware can be used in parallel with other **esd** CAN interfaces which are supported with **esd** NTCAN drivers. Refer to Table 13 which esd CAN interfaces come with a native SocketCAN support.

The following additional requirements have to be met:

1. Since version 2.6.25 the CAN support is part of the Linux kernel and can be enabled if compiled with `CONFIG_CAN`. Support for previous kernel versions might be available via the no longer updated web site of the SocketCAN project.
2. "ip"-tool supporting can  
 The install script tries to detect/build this automatically. If that help fails please go to `iproute2` directory and check **READMEs** there to build it manually.  
 If `make` is successful the file `.../ip/ip` should be built. Copy it to a binary path, e.g. `/usr/local/bin` then. All usages of just `ip` here might need that path then, too. (So `/usr/local/bin/ip` instead if just `ip` has to be typed)

#### 3.1.3.2 Installation

1. Make sure SocketCAN is available and a device driver is loaded, etc., e.g. by  
`ls /sys/class/net/`. When there is an entry `can0` everything should be fine, else don't try to install this NTCAN plugin before it exists.
2. Extract the archive, e.g. by  
`tar xvfz ntcansckPlugin32-2.0.7-ntcan-3.3.6.tgz`
3. Change into the newly created directory and run the install script, e.g. by  
`sudo ./install.sh`

### 3.1.3.3 Configuration

The configuration is done with the file `/etc/esd-plugin`

By default 3 nets are configured: SocketCAN device `can0` is configured as NTCAN net 60, `can1` is configured as NTCAN net 61 and so on.

To change the logical net numbers just edit the corresponding lines in the file:

- `libntcanSckPlugin.0.Net=60`
- `libntcanSckPlugin.1.Net=61`
- `libntcanSckPlugin.2.Net=62`

Other settings in that file:

- Verbosity level (Text output in the console):
  - `libntcanSckPlugin.verbose=0` Prints nothing
  - `libntcanSckPlugin.verbose=1` Only errors
  - `libntcanSckPlugin.verbose=2` Errors and warnings
  - `libntcanSckPlugin.verbose=3` Errors, warnings and Infos
- Automatically adapting Socket-CAN settings:
  - `libntcanSckPlugin.noscc=1` To disable that

Use the *cantest* application described in */1/*.which binary is extracted to the sub folder `cantest` to verify the correct installation. Run it without parameters to see a short help and a list of available CAN nets.

### 3.1.3.4 Restrictions

Due to the different driver architecture the SocketCAN wrapper can not map all capabilities of the NTCAN API because this feature is either not supported (e.g. error injection) or is implemented in a way that it can not be easily mapped (e.g. count of lost frames). This chapter contains (a maybe incomplete) list of differences between the SocketCAN wrapper and native NTCAN drivers. The NTCAN API is described in */1/*.

- Lost counter in `CMSG/CMSG_T` structs is not used (always zero).  
Use e.g. `sys/class/net/can0/statistics` for information about lost frames.
- Events: `NTCAN_EV_CAN_ERROR` is the only event supported. And within that event's data byte one (*error*) is the only supported byte.
- Only these I/O controls are supported (See NTCAN docs **canIoctl**):
  - `NTCAN_IOCTL_GET_BAUDRATE`
  - `NTCAN_IOCTL_SET_BAUDRATE`
  - `NTCAN_IOCTL_FLUSH_RX_FIFO`
  - `NTCAN_IOCTL_SET_20B_HND_FILTER`
  - `NTCAN_IOCTL_GET_TIMESTAMP`
  - `NTCAN_IOCTL_GET_TIMESTAMP_FREQ`
  - `NTCAN_IOCTL_GET_RX_TIMEOUT`
  - `NTCAN_IOCTL_GET_TX_TIMEOUT`
  - `NTCAN_IOCTL_SET_TX_TIMEOUT`
  - `NTCAN_IOCTL_SET_RX_TIMEOUT` (While `TX_TIMEOUT` is ignored, see **canWrite()**)

- Bus-OFF handling  
Current CAN's default is to stay off bus then. The application is usually helpless in that case. To avoid this, set Socket-CAN to automatically restart the device then. Currently this can be done with the `ip` command:  
e.g.:  

```
ip link set can0 type can restart-ms 1000
```

(Manual restart possible by `ip link set can0 type can restart`)

This is also done automatically. See chapter 3.1.3.3 for infos about how to avoid that.

- **canStatus()**  
Resulting `CAN_IF_STATUS` struct members:
  - *hardware* and *firmware* as described or 0 when they could not be determined.
  - *driver* is the wrapper plugin version number.
  - *dll* is set by NTCAN library.
  - *boardstatus* is one of enum `can_state` defined in `netlink.h` (or `0xffffffff` if it could not be determined)
  - *boardid* is "SocketCAN".
  - *features* is `NTCAN_FEATURE_BASIC_20B | NTCAN_FEATURE_TIMESTAMP`  
Where it is not guaranteed that the underlying hardware really supports CAN 2.0B or hardware timestamps. (Else timestamps will be set by software.)

- **canOpen()**

The flags, *txqueuesize* and *txtout* parameters are ignored. Socket-CAN's TX queue size can be set with the `ip` tool, e.g.:

```
ip link set can0 txqueuelen 1000
```

The receive buffer size depends on the given *rxqueuesize* value. But does not match exactly that number of frames. Also it is limited by *rmem\_max* value, which can be increased with e.g.

```
echo 1048576 >/proc/sys/net/core/rmem_max
```

These example values are also written automatically. See chapter “3.1.3.3 Configuration” for infos about how to avoid that.

- **canSetBaudrate()**

Only *Pre-defined bit rate table* and *User Bit Rate Numerical* are allowed. Usually only possible as "superuser". (e.g. when application is started with `sudo`) Also (re)starts the SocketCAN net interface if it was down or controller was Bus-OFF.

Stops the Socket-CAN net interface with `baud param NTCAN_NO_BAUDRATE`.

- **canGetBaudrate()**

Returns `NTCAN_NO_BAUDRATE` when controller is Bus-OFF or Socket-CAN net interface is down.

- **canRead()/canReadT()**

When function is waiting for a message, it is not interrupted when another thread closes the same NTCAN handle.

- **canWrite()/canWriteT()**

The TX timeout is ignored. It is blocking until message is written to socket. While this is fine for safely sending more messages than buffer could hold, it is still no guarantee they really went on the bus. (As we're only waiting till they're in the buffer/queue)

Does not return `NTCAN_CONTR_OFF_BUS/NTCAN_CONTR_WARN`. It will return `NTCAN_SUCCESS` even when controller is Bus-OFF! To avoid this use the error events and stop sending when controller goes off bus.

- CAN frame data length counter > 8

SocketCAN does not allow this, so when `CMSG/CMSG_T` member *len* is set to a value between 9 and 15 it is treated as 8.

### 3.1.4 EtherCAN and EtherCAN/2

In comparison to CAN interfaces connected to a local PC bus (PCI, USB, ...) supported with a Linux kernel mode device driver the EtherCAN and EtherCAN/2 interfaces are supported with a user mode device driver which integrates this remote CAN hardware into the NTCAN architecture in the same way as a local interface. This user mode device driver supports the EtherCAN/2 as well as the legacy EtherCAN hardware but for reasons of simplicity this chapter only refers to the EtherCAN/2.

The EtherCAN/2 software package is available for Linux (x86/x64) and contains the following files:

<i>File</i>	<i>Description</i>
README.x.x	Current notes and information
cantest.c	Source code of the example-application 'cantest' (located in subdirectory ./example) (see /1/.)
cantest	Binary of program 'cantest' (located in subdirectory ./bin)
makefile.ethercan	Example makefile for compiling the file cantest.c
installEthercanLibs	Bash script for installation of libraries and include files
etc/esd-plugin	Example config file for ntcaneThPlugin (to be stored as /etc/esd-plugin)
pdf/*.pdf	NTCAN API documentation (Part 1) and Installation (Part 2) and documentation of <b>esd</b> system abstraction layer API
include/ntcan.h	Header of the NTCAN API (to be stored e.g. at /usr/local/include). This is the only header that has to be include in the application. Please do not use any defines located in any of the other headers, in order to keep your applications working with future version of the driver!
lib/libntcan.so.x.x.x	Shared library containing the NTCAN-API (to be stored e.g. at /usr/local/lib)
lib/ntcaneThPlugin.so.x.x.x	Dynamically loadable plugin for libntcan.a or libntcan.so (to be stored at e.g. /usr/local/lib)
psys_linux/include/psys.h psys_linux/include/ psyslinux.h	Psys header files, <b>esd</b> System Abstraction Layer API (to be stored e.g. at /usr/local/include)
psys_linux/lib32/ libpsys.so.x.y.z	32-bit shared library, containing <b>esd</b> System Abstraction Layer (to be stored at e.g. under /usr/local/lib)
psys_linux/lib64/ libpsys.so.x.y.z	64-bit shared library, containing <b>esd</b> System Abstraction Layer (to be stored e.g. at /usr/local/lib64)
psys_linux/src/psysdrv.c	Source of the psys-driver
psys_linux/src/psysdrv.h	Psys-driver header
psys_linux/src/Makefile	KBuild-Makefile needed for the module generation with kernel 2.6.x
psys_linux/Makefie	makefile for PSYS-driver for kernel 2.4.x or 2.6.x
psys_linux/README	Release notes and installation hints for psys / psys-driver
psys_linux/LICENSE	License covering PSYS driver and library

### 3.1.4.1 Installation

**Note:**

The installation is only possible with superuser rights (user: root). Please read the current `README` file that comes with the software!

#### 1. Unpacking the Archive

In order to unpack the zipped tar-archive file the following command has to be called:

```
tar xvfz ethercan-lx-2.0.10-ntcan-3.0.6- psys-1.3.0-gcc-3.3.1-glibc-2.3.2.tgz
```

The unpacked files will be stored in a directory that carries the same name as the archive file.

#### 2. Compile and Load Psys-Driver

Therefore follow the installation instructions in *psys\_linux/README*.

#### 3. Installing the Libraries and Header Files

In order to install the libraries and header files the following command has to be called (with supervisor rights):

```
./installEthercanLibs
```

All library files will be stored in `/usr/local/lib`  
and all include files will be stored in `/usr/local/include`.

#### 4. Adapt `/etc/esd-plugin`

See chapter “3.1.4.2 Configuration”.

#### 5. Compiling the Example Program ‘cantest’

With the call

```
make -f makefile.ethercan
```

the test- and example program `cantest` will be compiled.

After successful installation you can access the CAN-Bus via the esd NTCAN API (link `libntcan` with your application). For a complete NTCAN API documentation refer to `/1/`.



### 3.1.4.2 Configuration

All user configurable stuff concerning EtherCAN can be found in the file `/etc/esd-plugin`.

List of available keywords in `/etc/esd-plugin` (with  $0 < x < 4$ ):

Keyword	Description	Default Value
PeerName[x]	host name or IP-address of EtherCAN server	-
Net[x]	CAN net number assigned to EtherCAN server with above PeerName[x]	50 + x
ConnTimeout[x]	Time to wait until connection to the EtherCAN server is established. If timeout exceeds, <code>canOpen()</code> returns <code>NTCAN_SOCK_CONN_TIMEOUT</code> .	2500 ms
CmdTimeout[x]	Timeout for special commands send from client to EtherCAN-server.	2500 ms
KeepAliveTime[x]	If there is no CAN traffic, client sends a keep-alive message to server. If sending the keep-alive message fails, the connection to the server is disconnected and the EtherCAN client will try to reconnect the server.	2500 ms
TCPNoDelay[x]	0: Nagle algorithm active (i.e.: Coalesce a number of TCP messages and send them all at once) 1: Nagle algorithm off (i.e.: Immediately send TCP messages without any inhibit)	1

**Example 1:** EtherCAN configured as CAN net 30 and all other parameters in default setting.

```
PeerName[1]=      "10.0.16.58"
Net[1]=           30
```

**Example 2:** EtherCAN configured as CAN net 20 and, with increased timeouts, because it is located outside the company network.

```
PeerName[0]=      "134.66.177.1"
Net[0]=           20
KeepAliveTime[0]= 10000 # increase keep-alive timeout
ConnTimeout[0]=   25000 # increase connection timeout
CmdTimeout[0]=    5000  # increase command timeout
```

### 3.1.4.3 Miscellaneous

This chapter covers several topics which should preclude problems using the EtherCAN/2 on Linux in your applications.

#### 3.1.4.3.1 Linking Against `libntcan` (gcc-Option `-rdynamic`)

**Attention!**

It is mandatory to use the option `-rdynamic` when linking against `libntcan`, because the dynamically loaded library `ntcanEthPlugin.so` (beside delivering some new functionality) itself needs some symbols from within `libntcan`. Without `-rdynamic` this does not work!

If your application (on runtime) complains about `'ntcanEthPlugin.so: undefined symbol: openRegistryCanIf Ether'` the option `-rdynamic` is still missing in your makefile.

# 3.2 Legacy UNIX Versions



**Attention!**  
As the release date of all UNIX versions covered in this chapter was more than 20 years ago, technical support and maintenance by **esd** for these drivers is terminated.

## 3.2.1 PowerMAX OS Installation



**Note:**  
The 'object mode' of the CAN-API is not supported by PowerMAX OS.

### 3.2.1.1 Files of the PowerMAX OS Package

The software drivers for PowerMAX OS are on a CD-ROM, which contains the following files:

File	Description							
README.ican4  e.g. ican4	current notes and information  dynamically loadable driver	<div>The letter combination ‘ican4’ signifies the files of the module VME-CAN4. For other modules these letters are changed as follows:</div> <table><tr><th>CAN Module</th><th>File Name</th></tr><tr><td>VME-CAN2</td><td>ican2</td></tr><tr><td>VME-CAN4</td><td>ican4</td></tr></table>	CAN Module	File Name	VME-CAN2	ican2	VME-CAN4	ican4
CAN Module	File Name							
VME-CAN2	ican2							
VME-CAN4	ican4							
libntcan.o	ntcan-API							
ntcan.h	header for ntcan-API							
cantest.c	source code of example program ‘cantest’ (see /1/)							
cantest	binary file of example program ‘cantest’							

### 3.2.1.2 Sequence of Installation Under PowerMAX OS

#### 1. Copy the tar-Archive `vmecan4_v1.2.tar` in the Home Directory

#### 2. Unpacking the Archive

In order to unpack the tar-archive the following command has to be called:

```
tar xvf vmecan4_v1.2/ican4
```

'ican4' has to be entered for the VME-CAN4 module. For other modules the character combination shown in the following table has to be entered. The same applies to the following commands.

Input syntax for unpack the archive:

CAN Module	Entry Syntax
VME-CAN2	ican2
VME-CAN4	ican4

#### 3. Generating the 'inodes' (as Superuser)

```
cd 4_2/vmecan4_v1.2/ican4/
or
cd 4_3/vmecan4_v1.2/ican4/
```

#### 4. Only at the First Installation: Add Adapter Definition

At the first installation of the CAN modules the following lines hat to be added to the file `/usr/include/sys/adapt3er_vme.h` :

```
#define ADAPTER_ICAN2    (AVB+0x12)    /* 0x212 - esd ican2 */
#define ADAPTER_ICAN4    (AVB+0x13)    /* 0x213 - esd ican4 */
```

#### 5. Only if an Existing Driver is Updated:

If an existing driver should be updated, the following line is necessary:

```
modadmin -U ican4
```

#### 6. Installation

- in case of first installation:

```
./install -f
```

with following reboot

## 7. Change Directory:

change to

`~/4_2/vmecan4_v1.2` for PowerMAX OS 4.2

or `~/4_3/vmecan4_v1.2` for PowerMAX OS 4.3

## 8. Start Driver:

After calling

```
modadmin -l ican4
```

the driver displays his start message.

## 9. Starting `cantest`:

After starting `cantest` with

```
./cantest
```

the program shows four accessible CAN nets (net 0...3)

After a successful installation, the CAN bus can be accessed by means of the NTCAN-API (integration of '*libntcan.o*' into the application).

## 3.2.2 Solaris™ Installation



**Note:**

The 'object mode' of the CAN-API is not supported by Solaris.

### 3.2.2.1 Files of the Solaris Package

The software drivers for Solaris are contained on a CD-ROM. This CD contains the following files:

File	Description									
install	script for installation of driver	<div>The character combination 'c331' signifies the files of the CAN-PCI/331 module. For other modules the characters change as follows:</div> <table><tr><th>CAN Module</th><th>File Name</th></tr><tr><td>CAN-ISA/331 CAN-PC104/331</td><td>c331i</td></tr><tr><td>CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331</td><td>c331</td></tr><tr><td>VME-CAN4</td><td>ican4</td></tr></table>	CAN Module	File Name	CAN-ISA/331 CAN-PC104/331	c331i	CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331	VME-CAN4	ican4
CAN Module	File Name									
CAN-ISA/331 CAN-PC104/331	c331i									
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331									
VME-CAN4	ican4									
README.c331	current notes and information									
c331	dynamically loadable driver									
c331.conf	parameter file for driver (is read at installation and deinstallation)									
ntcan.o	ntcan-API									
ntcan.h	header for ntcan-API									
canupd	program for firmware update									
cantest.c	source code of example program 'cantest' (see /1/.)									
cantest	binary file of example program 'cantest'									

### 3.2.2.2 Sequence of Installation Under Solaris

#### 1. Unpacking the Archive

In order to unpack the tar-archive you have to call the following command:

```
tar -xvf c331-solaris-vx.x.x.tar
```

(x.x.x = driver version number)

The entry 'c331' has to be made for the CAN-PCI/331 module. Please specify the corresponding letter combination shown in the table below for other modules:

Entry syntax when unpacking the archive:

<i>CAN Module</i>	<i>Entry Syntax</i>
CAN-ISA/331 CAN-PC104/331	c331i
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331
VME-CAN4	ican4

#### 2. Preparing the System to Dynamically Create 'inodes':

To make sure that during the driver installation 'inodes' are automatically created under the /dev directory, it is necessary to conform the file /etc/devlink.tab. This has to be made with superuser rights. It is recommendable to make a backup of this file before starting. The following line has to be added to the file:

```
type=can; \M0
```

When installing the driver for the first time, the files /dev/canx should be automatically created now, x indicating the (hexadecimal) network number.

## 3. Conforming the Configuration File

### 1. Conforming the Configuration File for CAN-ISA Modules:

Before the driver is installed for the first time, its driver configuration file has to be conformed to the hardware configuration. The following properties have to be available in `driver.conf`, where `driver` has to be substituted by the according driver name. Only the properties printed in *italics* have to be conformed. For a summarizing overview of the (bus-specific) properties) please consult the according manual pages (`driver.conf`, `sysbus`, `pci`).

<i>Name</i>	<i>Parameter</i>	<i>Meaning</i>
<code>name</code>	String	Driver name
<code>class</code>	String	Bus type
<code>interrupts</code>	Numeric	Interrupt vector
<code>interrupt-priorities</code>	Numeric	Interrupt priority level (IPL)
<code>reg</code>	Numeric	Three values, separated by commas. The second corresponds to the CAN interface basis address and the third describes the size of the I/O area in bytes.

The value given in `reg` has to correspond to the basis address configured in the hardware by means of jumpers or coding switches. The IPL level has to be assigned to a high-level interrupt and the interrupt vector must not have been assigned by another hardware component.

Below an example configuration file for a CAN-ISA/331 is shown. Its basis address has been configured at 0x1E0 and it is to use the interrupt vector 7 with an IPL of 11:

```
# Copyright (c) 1998, by esd gmbh.
#
name="isa331" class="sysbus" interrupts=7 interrupt-priorities=11 reg=1,0x1e0,8;
```

### 2. Conforming the Configuration File for CAN-PCI Modules:




Since PCI devices report themselves, the configuration and assignment of resource is automatically when the system is started (Plug & Play). The device driver automatically adopts the assigned resource so that it does not have to be manually assigned in the driver configuration file. The only parameter which can be set is the interrupt priority to be used by the IPL driver (see also example `c331.conf`).

```
# Copyright (c) 1999-2000 electronic system design gmbh
#
# do not remove the next line
interrupt-priorities=9;
```



### 3. Conforming the Configuration File for the VME-CAN4 Module:

Before the driver is installed for the first time, its driver configuration file has to be conformed to the hardware configuration. The following properties have to be available in `driver.conf`, where driver has to be substituted by the according driver name. Only the properties printed in *italics* have to be conformed. For a summarizing overview of the (bus-specific) properties) please consult the according manual pages (`driver.conf`, `sysbus`, `pci`).

Name	Parameter	Meaning
name	String	Driver name, here always: <code>ican4</code>
class	String	Bus type here always: <code>vme</code>
interrupts	Numeric	<p>Interrupt level and vector</p> <p>Four interrupts at the same interrupt level with interrupt vectors with the offset of 1, 4 and 5 have to be defined (see following example).</p> <div data-bbox="612 757 708 851"></div> <p><b>Note:</b> The hardware of the VME-CAN4 supports up to 8 interrupt vectors. Because of the compatibility to the VME-CAN2 the vectors 1, 2, 5 and 6 are used (according entries: 0x80, 0x81, 0x84, 0x85).</p> <div data-bbox="604 965 719 1068"></div> <p><b>Attention:</b> The interrupt level has to carefully selected to obviate conflicts with existing interrupts of the system!</p>
reg	Numeric	<p>Contains six values separated by commas, that define the address range of the VME-CAN4:</p> <ul style="list-style-type: none"> <li>• The first value defines the 1. address range: 0xad =&gt; A16</li> <li>• The second value defines the board address within the 1. address range that is used to initialize the address registers of the VME-CAN4:  <code>0xe000 + (coding_switch_setting * 0x100)</code>  (only, if the geographical addressing is inactive, please refer hardware manual of VME-CAN4)</li> <li>• The third value defines the size of the 1. address range in bytes and thus the offset to the next board within the A16-address range of the system: always 0x100</li> <li>• The fourth value defines the 2. address range: 0x4d =&gt; A32</li> <li>• The fifth value defines the A32-address of the VME-CAN4 board: e.g. 0x10000000</li> <li>• The sixth value defines the size of the 2. address range in bytes: always 0x00100000</li> </ul> <div data-bbox="612 1778 708 1872"></div> <p><b>Note:</b> Due to reasons of the address coding the offset to the next board within the A32-address range in the system is 0x00200000!</p>

Below an example configuration file for the VME-CAN4:

```
name="ican4"
class="vme"
interrupts=6,0x80,6,0x81,6,0x84,6,0x85
reg=0xad,0xe100,0x100,0x4d,0x10000000,0x00100000;
```

#### 4. Installing the CAN Driver

In order to install the driver the script `install` has to be executed as superuser. It copies the driver files into the target directory, installs and starts the driver. From now on the driver will be automatically loaded and started with every system start.

After it has been successfully installed, you can access the CAN bus via the NTCAN-API (including `'ntcan.o'` into the application).

#### 5. Checking the Installation

##### 1. For CAN-PCI Modules Only:

If the installation script has been executed correctly, the driver is started and automatically loaded with every system start.

By entering `"modinfo | grep d3x"` into the root you can check whether the driver is loaded or not. An output similar to the following has to appear:

```
205 f5d11000    34d9 132    1  c331 (CAN-PCI/331 driver v1.3.3)
```

##### 2. For All Modules:

You can check whether the installation was successful by reading the driver boot message in the following file:

```
/var/adm/messages
```

#### 6. Unloading the CAN driver

Unloading the CAN driver is. For example, necessary after an update of the local firmware for resetting the processor. For the CAN-ISA/331 module this can be achieved by the following command, which has to be executed as a superuser:

```
rem_drv c331i
```

### 3.2.3 SGI-IRIX6.5 Installation



**Note:**

The 'object mode' of the CAN-API is not supported by SGI-IRIX6.5 .

#### 3.2.3.1 Files of the SGI-IRIX6.5-Package

The software drivers for SGI-IRIX6.5 are shipped on a CD-ROM. The CD contains the following files:

File	Description		
c331	CAN driver (object code) driver configuration file driver configuration file	The letter combination 'c331' indicates the files of module CAN-PCI/331. For other modules these letters will be substituted as follows:	
c331.master			
c331.sm			
		CAN Module	File Name
		CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331
		CAN-PCI/405	pci405
makefile	makefile for installing and loading the driver		
ntcan.o	ntcan-API		
ntcan64.o	ntcan-API / 64-bit version		
ntcan.h	Header for the ntcan-API		
cantest.c	source code of the example program 'cantest' (see /1/)		
cantest	binary file of the example program 'cantest'		

**3.2.3.2 Sequence of Installation Under SGI-IRIX6.5****1. Login as Root****2. Unpacking the Archive**

In order to unpack the tar-archive of the CAN modules CAN-PCI/331, CPCI-CAN/331 and PMC-CAN/331 you have to call the following command:

```
tar -xvf c331-IPyy-vx.x.x.tar
```

with

`x.x.x` = software driver version (e.g. 2.1.0)

`yy` = processor identification (e.g. '32' for SGI-O2)

In order to unpack the tar-archive of the CAN module CAN-PCI/405 you have to call the following command:

```
tar -xvf esdcan-pci405-irix-mips-x.x.x-IPyy-z.z
```

with

`x.x.x` = software driver version (e.g. 0.3.2)

`yy` = processor identification (e.g. '27')

`z.z` = operating system version (e.g. '6.5')

**3. Change Directory (e.g. CAN-PCI/331, CPCI-CAN/331 and PMC-CAN/331)**

```
cd c331-IPXX-v2.1.0
```

**4. Install Driver Data**

```
smake install
```

**5. Load Driver**

```
smake load
```

If the driver has been installed and loaded correctly, a message of the driver has to appear on the screen now.

## 3.2.4 AIX Installation

### 3.2.4.1 Special Features of the AIX Implementation

- The CAN driver has been designed for operating system version AIX 4.2.1
- Hardware platform: PowerPC
- Implemented **esd** CAN modules: CAN-PCI/331, CPCI-CAN/331, PMC-CAN/331
- 29-bit identifiers are not yet being supported
- The 'object mode' of the CAN-API is not supported by AIX

### 3.2.4.2 Files of the AIX Package

The software drivers for AIX are contained on a CD-ROM. The CD contains a tar-archive.

File	Description					
README.c331	current notes and information	The character combination ‘c331’ signifies the files of the following modules: <table><tr><th>CAN Module</th><th>File Name</th></tr><tr><td>CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331</td><td>c331</td></tr></table>	CAN Module	File Name	CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331
CAN Module	File Name					
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331					
c331	dynamically loadable driver					
ntcan.o	ntcan-API					
ntcan.h	header for ntcan-API					
nttest.c	source code of example program ‘cantest’					

### 3.2.4.3 Installation Sequence under AIX

#### 1. Login as Root

#### 2. Unpacking the Archive

In order to unpack the tar-archive the following command has to be called:

```
tar -xvf c331-ppc-vx.y.z.tar
```

(x.y.z = driver version number)

#### 3. Copying the Files

At unpacking the directory `/c331-ppc-vx.y.z` is created.

Copy the files `cfg_pci331` and `ucfg_pci331` from this directory into the directory `/usr/lib/methods/`.

Copy from this directory the file `c331` into the directory `/usr/lib/drivers/pci/`.

#### 4. Create Entries in the Data Base

```
run odmadd pci331
```

#### 5. Call the Configuration Manager

```
cfgmgr -v
```

#### 6. Check the Files and Create Symbolical Links

Check whether the devices `c33100` and `c33101` are entered in directory `/dev/`. Create a symbolical link:

```
ln -s /dev/c33100 /dev/can0
ln -s /dev/c33101 /dev/can1
```

The driver is now installed.

## 7. Check Whether the Driver Runs With CAN tool 'CANreal'

Make sure that the wiring and terminations are correct and make sure that at least one other working CAN participant has been connected!

Start the monitor program CANreal:

```
./canreal &
```

Set the baud rate of your CAN network for network 0 in CANreal and click *Init*.

'Init done' appears in the window

Click *Add Id* to select the identifier area of 0 to 2047.

Click *Start* to display the messages of the CAN bus.

## 8. Test Program `cantest`

In addition to CANreal, which has got a UNIX-typical user interface, you can also use the test program `cantest` under AIX. It is operated by means of command line specification (for details see chapter 'Test Program `cantest`' in /1/).

compile `cantest`:

```
gcc -o cantest nttest.c ntcana.o
```

call `cantest`:

```
./cantest
```

## 4 Real-Time Operating Systems

This chapter covers the necessary steps to install, configure and start the device drivers for **esd** CAN interfaces available for the real time operating systems described in this chapter. In comparison to Windows or Unix operating systems, described in the previous two chapters, the application is usually developed on a host system which is different from the (embedded) target system the device driver and the application runs on.

In comparison to the previous chapters which described the driver installation and configuration for **esd CAN Interfaces** this chapter covers **esd CAN Boards** (**esd** CAN Interfaces and embedded systems) as well as CAN driver developed for customer hardware.

### 4.1 VxWorks®

**Note:**

For the changes introduced with the last service release, please check the readme of the current release.

This chapter covers the necessary steps to install, configure and start the device drivers for **esd** CAN boards supporting the real time operating system Wind River VxWorks.

CAN device drivers are available for VxWorks 5.x, 6.x and 7.x for different CPU target architectures.

The architectural differences between VxWorks 5.x and VxWorks 6.x with and without VxBus support are covered in separate chapters.

**Note:**

Wind River offers the middleware component '*Wind River CAN for VxWorks 6.x*' as part of the VxWorks 6.x platforms '*Wind River Platform for Automotive Devices*' and '*Wind River Platform for Industrial Devices*'. Only the **esd** CAN interfaces CAN-PCI/200 and CAN-PC104/200 are supported directly by this Wind River implementation.

**The esd NTCAN drivers do not rely on '*Wind River CAN*' and can be used with any VxWorks platform.** These device drivers are required if you want to use the optionally available higher layer CAN protocol stacks (CANopen, ARINC825, J1939, etc.) by **esd**.

A VxWorks CAN driver package supports one or more families of **esd** CAN Interfaces for a certain target architecture. The package contains a file `relnotes.htm` in HTML format which contains the revision history of the drivers and late-breaking information which did not make into one of the manuals. Please also read this file before installing the driver.

**Note:**

The CAN device driver for **esd** embedded boards with support for VxWorks are part of the BSP and are not deployed as a separate CAN driver package.



### 4.1.1 CAN Board Support Overview

Device drivers for **esd** CAN interfaces are available for different VxWorks versions and CPU architectures. A single device driver often supports more than one CAN interface type (refer to chapter 1.4 for the CAN interfaces which belong to the same interface family). For VxWorks 6.x device drivers with and without VxBus support are available.


**Note:**

If a combination of CAN interface, VxWorks version and/or CPU architecture is currently marked as not supported in the tables in this chapter please contact the **esd** support for help.

All VxWorks 5.x and 6.x non-VxBus device drivers are based on the 2.x version of the **esd** NTCAN driver architecture (see chapter *Driver History* in *I1I*). For VxWorks 6.x revision 6.6 or later is required.

VxWorks	5.4.x			5.5.x		6.x (UP, no VxBus)		
Architecture	386/486	Pentium	PPC	Pentium	PPC	Pentium	Pentium4	PPC
CAN Family								
C200I	✓	-	-	✓	✓	✓	-	-
C331I	✓	✓	-	✓	-	✓	-	-
C200	✓	✓	-	✓	-	✓	✓	✓
C331	✓	-	✓	✓	✓	✓	✓	✓
C360	✓	-	-	-	-	-	-	-
ICAN4	-	-	✓	-	✓	-	-	✓

The VxWorks VxBus-enabled CAN device driver support uniprocessor (UP) and symmetric multiprocessor (SMP) versions of VxWorks. Minimum requirement is a VxWorks version which supports at least VxBus 4 which was introduced with VxWorks 6.7. All drivers are based on the 3.x version of the **esd** NTCAN driver architecture (see chapter *Driver History* in *I1I*).


**Attention!**

For VxWorks 6.x UP the C200/C331 CAN interface family is supported by a VxBus and a non-VxBus (legacy) driver. Make sure you never include both device driver types in the same VxWorks image.

<i>VxWorks</i>	<b>6.x (UP, VxBus)</b>					<b>6.x (SMP, VxBus)</b>		
<b>Architecture</b>	Pentium	Pentium4	Core	Nehalem	PPC	Pentium4	Core	Nehalem
<b>CAN Family</b>								
C200	✓	✓	✓	✓	✓	✓	✓	✓
C331	✓	✓	✓	✓	✓	✓	✓	✓
C400	✓	✓	✓	✓	✓	✓	✓	✓
C405	✓	✓	✓	✓	✓	✓	✓	✓

<i>VxWorks</i>	<b>7.x (UP, VxBus)</b>					<b>7.x (SMP, VxBus)</b>		
<b>Architecture</b>	Pentium	Pentium4	Core	Nehalem	PPC	Pentium4	Core	Nehalem
<b>CAN Family</b>								
C331	✓	✓	✓	✓	✓	✓	✓	✓
C402	✓	✓	✓	✓	✓	✓	✓	✓

In addition to the CAN interfaces **esd** also provides VxWorks BSPs for several embedded boards with on-board CAN interfaces. These boards are also supported with the NTCAN driver architecture but the CAN device driver itself is part of the BSP. The table below gives an overview about the availability for different VxWorks versions and the included CAN driver version.

<b>Board</b>	<b>VxWorks</b>	<b>BSP version</b>	<b>CAN Driver Version</b>
PMC-CPU/405	5.5	1.2/15	2.x
PMC-CPU/440	>= 6.8	2.0/5	2.x
CPCI-CPU/750	5.5, >= 6.7	1.2/02, 1.2/02	3.x, 3.x
EPPC-405	5.4, 5.5, >=6.5	1.2/8, 1.2/11,2.0/10	2.x, 2.x, 2.x

## 4.1.2 Driver Integration

This chapter describes how the **esd** CAN device drivers can be included into a VxWorks project. The integration and installation of the drivers and libraries is different between VxWorks 5.x and VxWorks 6.x/7.x.

For VxWorks 5.x the driver and libraries are deployed as binaries which have to be linked to your application or BSP together with a configuration file which has to be adapted to the target hardware.

For VxWorks 6.x/7.x the driver installation and configuration is integrated into the Wind River Workbench.

### 4.1.2.1 VxWorks 5.x

The driver software for VxWorks 5.4.x and 5.5.x is deployed as Downloadable Kernel Modules (DKM) which have to be linked to the VxWorks image or can be loaded on application startup. An integration into Tornado is not supported. The driver comes as a CAN Interface specific package with the directory structure `/vw5x/CPU-Architecture/`.

A CAN device driver package for VxWorks 5.x contains the following files where `<drvname>` is the device family specific driver name following driver naming convention I (see chapter 1.4).

<i>File</i>	<i>Description</i>
<code>Ldcan</code>	Script to load driver and NTCAN library.
<code>&lt;drvname&gt;.sys</code>	CAN driver (binary).
<code>&lt;drvname&gt;.ini</code>	CAN driver configuration and start code (binary)
<code>caninit.c</code>	CAN driver configuration and start code (source of <code>&lt;drvname&gt;.ini</code> binary)
<code>ntcan.o</code>	NTCAN library (binary)
<code>ntcan.h</code>	Header of the NTCAN-API for application development
<code>cantest.c</code>	Source code of example program <b><i>canTest</i></b>
<code>cantest</code>	Binary file of example program <b><i>canTest</i></b>

Depending on the released version of the driver there may be some postfixes on the `CPU-Architecture` part of directory names that denote build variants of the same driver for a certain `CPU-Architecture`. Here are some examples:

- `_SHRD`: The driver was built to use ***pcilntConnect()*** instead of ***intConnect()***. An example is `PENTIUM_SHRD`. The other version that uses ***intConnect()*** will be named `PENTIUM`. This is valid only if both variants for one `CPU-Architecture` exist. In any other cases the default value for the architecture is used.
- `_LONG`: The driver was built with the “-mlongcall” option for the C-Compiler that circumvents the 32MB branch distance limitation on the PowerPC architecture. An example is `PPC604_LONG`.

### 4.1.2.2 VxWorks 6.x

The driver software for VxWorks 6.x with and without VxBus support comes as a package for all supported host CPU architectures and CAN hardware with the following directory layout structure:

Directory	Description
doc\ src\ target\ 	Documentation of driver installation and NTCAN-API Example code ( <code>cantest.c</code> ) also included in binary format which can be optionally added to your project. This directory contains all necessary files to integrate and configure the driver. <b>For driver installation and integration in the Wind River Workbench the complete folder has to be copied into the target directory of your VxWorks 6.x installation keeping the directory hierarchy.</b>

The support to integrate the CAN driver in the VxWorks image will be available with the next start of the Workbench in the '*Kernel Configuration*' view below the new category '*esd gmbh Driver, Protocol Stacks and Software*'.



**Note:**

Some versions of the Wind River Workbench do not update their Components with each restart. To solve this issue it might help to remove the CDF cache file

`\target\config\comps\vxWorks\CxrCat.txt`

before restarting the Workbench.

To uninstall the driver package you just have to remove all files copied into the target architecture of your VxWorks installation.

### 4.1.2.3 VxWorks 7.x

The driver software for VxWorks 7.x with VxBus GEN2 support comes as a package for all supported host CPU architectures and CAN hardware. Once the RPM files are copied onto the directory <local\_dir> on the host, the following commands are used to integrate the software to be used with the Workbench (the version numbers of the RPM files depends on the current distribution):

#### Linux:

```
<vw7_install_dir>/maintenance/wrInstaller/x86-linux2/wrInstaller -silent -nosplash -yum localinstall
<local_dir>esd_CAN_PCI331-4.0.0.0-1.noarch.rpm -y
```

```
<vw7_install_dir>/maintenance/wrInstaller/x86-linux2/wrInstaller -silent -nosplash -yum localinstall
<local_dir>esd_CAN_NTCAN-3.7.4.0-1.noarch.rpm -y
```

#### Windows:

```
<installation>\maintenance\wrInstaller\x86-win32\wrInstallerc.exe -silent -nosplash -yum localinstall
<local_dir>\esd_CAN_PCI331-4.0.0.0-1.noarch.rpm -y
```


```
<installation>\maintenance\wrInstaller\x86-win32\wrInstallerc.exe -silent -nosplash -yum localinstall
<local_dir>\esd_CAN_NTCAN-3.7.4.0-1.noarch.rpm -y
```

The following output should be seen (e.g. when installing the PCI331 driver):


```
--> Finished Dependency Resolution
Dependencies Resolved
#####
Package Arch Version Repository Size
#####
Installing:
esd_CAN_PCI331 noarch 4.0.0.0-1 local rpms 1009.9 KB
Transaction Summary
#####
Install 1 Package(s) (+0 Dependent packages)
Total download size: 1009.9 KB
Is this ok [y/N]: y
Downloading packages:
esd_CAN_PCI331-4.0.0.0-1.rpm | 1009.9 KB < 1 min
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing : esd_CAN_PCI331-4.0.0.0-1.noarch 1/1
Configuring installation...
Installed:
esd_CAN_PCI331.noarch 4.0.0.0-1
Complete!
```

When starting the Workbench the CAN driver, the NTCAN library and the cantest utility is available and integrated by default.

To **uninstall** the driver package, just start the wrInstaller, click on "Remove" and select the package to remove: esd\_can.

 **Select products to remove**

Filter products and features

▼  Packages

▶ ☐ Altera\_Cyclone\_Support

▶ ☐ ARM\_Architecture\_Support

▶ ☐ Authentication

▶ ☐ Basic\_Security

▶ ☐ bsp

▶ ☐ CANbus


▶ ☐ Cryptography


▶ ☐ Cryptography\_Security


▶ ☐ Debug\_and\_Analysis

▶ ☐ Device\_Drivers

▶ ☐ EMS\_Agent

▼ ☒  esd\_can

☒  esd\_CAN\_NTCAN 3.7.4.0

☒  esd\_CAN\_PCI331 4.0.0.0

▶ ☐ File\_Systems

▶ ☐ Font

▶ ☐ Frame\_Buffer

▶ ☐ Freescale\_iMX\_Support

▶ ☐ Freescale\_Kinetis\_Support

Select all

Deselect all

### 4.1.3 Driver Configuration

This chapter describes the CAN device driver configuration for the various versions of VxWorks.

#### 4.1.3.1 VxWorks 5.x

Driver for VxWorks 5.x are configured calling the driver start routine `<drvname>_install()` with a pointer to an initialized array of structures of the type `<drvsig>_CAN_INFO` (one array entry for each CAN interface). This can either be performed by modifying the code of the example start-up file `caninit.c` (recommended) or by calling the driver start routine `<drvname>_install()` from within your own application. In either case please take a look into the `caninit.c` to see configuration examples for several target architectures.

The varying name parts `<drvname>` and `<drvsig>` can be derived from Table 2 in chapter 1.4 where `<drvname>` is the device family specific driver name following driver naming convention I and `<drvsig>` is the signature.

The following section shows the members of a configuration structure, which is valid for most of the CAN Interface Families but NOT for all. To be confident about the structure's layout refer to the `caninit.c` source from the driver package.

```
struct <drvsig>_CAN_INFO
{
    unsigned long base;
    unsigned char net[4];
    unsigned char prio;
    unsigned char irq;
    unsigned long timestampFreq;
    unsigned char flags;
    unsigned char reserved;
};
```

The table below describes the members of the configuration structure:

Member	Description
base	<p><b><u>ISA and PC104 CAN Interfaces:</u></b></p> <p>I/O base address of the module. <code>base</code> has to be set to the value configured on the hardware via jumpers or coding switches. If <code>base</code> is set to '0', the driver terminates the search for further CAN interfaces.</p> <p><b><u>PCI, CPCI, PMC and PCIe CAN Interfaces:</u></b></p> <p>For <b>x86</b> architectures the PCI resources of the CAN hardware are configured usually by a kind of boot loader (usually the PC BIOS). In order to tell the device driver to use these addresses you have to set the parameter <code>base</code> to <code>0xFFFFFFFF</code>. You have to make sure that your BSP has got enough available entries in the MMU Memory Descriptor table (dummy entries) to register the PCI addresses of the CAN interface.</p>

Member	Description
	<p>For <b>PPC</b> architectures where the BSP or a boot loader performs the PCI bus enumeration and configuration you have to set the parameter <i>base</i> to 0xFFFFFFFF.</p> <p>For <b>PPC</b> architectures without any kind of boot loader you have to find an unused physical PCI address room (at least 3 MB, page aligned) which is used by the driver if configured in the parameter <i>base</i>.</p> <p>For debug level information about the correct setup of <i>base</i> have a look into the troubleshooting chapter 4.1.6.4.2.</p>
<code>net[0]</code>	<p>The parameter <code>net[0]</code> is the logical base net number that should be assigned to the first physical CAN port on the CAN board. If the hardware has more than one physical CAN port these ports will get consecutive logical net numbers starting with the base number. The values in <code>net[1]</code> to <code>net[3]</code> are ignored by the driver. The user has to make sure that all assigned logical net numbers are unique (especially if more than one <b>esd</b> CAN driver is active) as driver initialization otherwise will fail.</p>
<code>prio</code>	<p>Priority of back end task(s) which handle post processing of CAN messages. This back end task is responsible for the distribution of received CAN messages to all open handles. Therefore its priority must be better than the priority of all tasks that are doing CAN I/O via the driver to make it all work as expected.</p>
<code>irq</code>	<p><b><u>ISA and PC104 CAN Interfaces:</u></b> Interrupt vector which should be configured and used by this CAN interface. You have to make sure that this interrupt is not used by any other hardware.</p> <p><b><u>PCI, CPCI, PMC and PCIe CAN Interfaces:</u></b></p> <p>For <b>x86</b> architectures the PCI resources of the CAN hardware are configured usually by a kind of boot loader (usually the PC BIOS). In order to tell the device driver to use the configured resources you have to set the parameter <code>irq</code> to 0xFF. In this case the IRQ value configured in the configuration space of the PCI bridge is used and the default offset of 0x20 for a PIC system is added.</p> <p>For <b>PPC</b> architectures where the BSP or a boot loader performs the PCI bus enumeration and configuration you have to set the parameter <code>irq</code> to 0xFF.</p> <p>For <b>PPC</b> architectures without any kind of boot loader you have to find out the interrupt used by your target for the PCI slot in your hardware manual and set the parameter <code>irq</code> to this value regarding any BSP specific interrupt vector offsets. This manual setup may also be used to overcome issues that arise from wrong IRQ number translation, see chapter 4.1.6.5.2.</p>
<code>timestampFreq</code>	<p>The parameter <i>timestampFreq</i> defines the frequency of the timestamp in kHz for CAN drivers supporting software timestamps which are derived by a high resolution timer of the target CPU. If this parameter is set to 0 the driver probes this frequency, which causes a delay of driver startup where all interrupts and the scheduler are disabled.</p> <p>For <b>x86</b> architectures the high resolution timestamp runs at processor frequency for most <b>PPC</b> targets it runs at 1/4 processor frequency. If a driver or the hardware does not support software timestamps the parameter should be set to 0.</p> <p><b>Note:</b> This structure member does not exist for the VME-CAN4 driver. Instead the driver exports <i>can4_tickFreq</i> because of parameter structure size restrictions.</p>

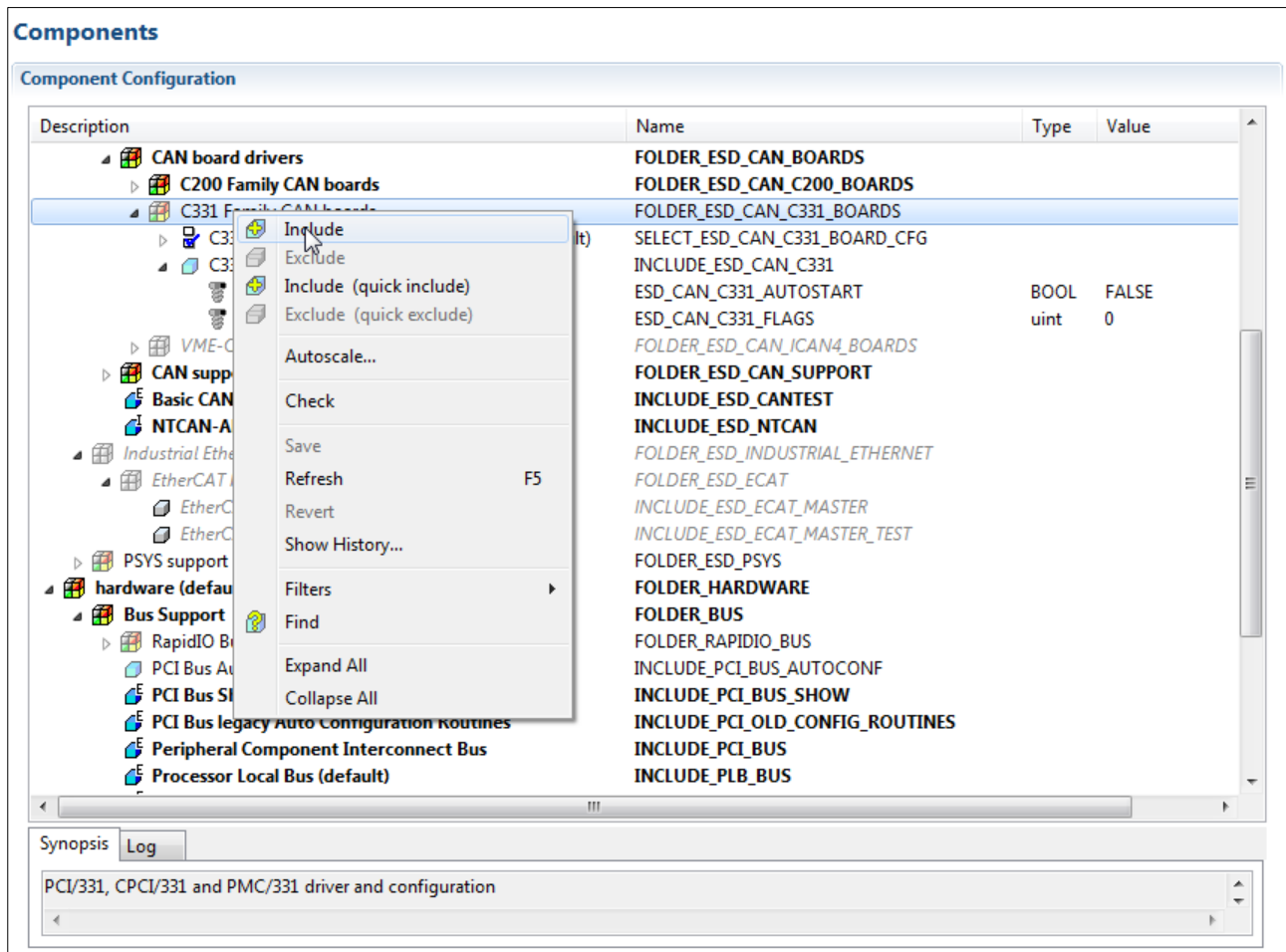


<b>Member</b>	<b>Description</b>						
flags	<p>The parameter <i>flags</i> defines device flags supported by the driver:</p> <table> <tr> <td>0x01</td><td>Suppress driver start-up banner.</td></tr> <tr> <td>0x02</td><td>Devices of the C331 and C200 family (Driver rev. 2.5.8 and later): Enable 'Delayed Read' support as specified in PCI specification v2.1</td></tr> <tr> <td>0x04</td><td>Devices of the C331 and C200 family (Driver rev. 2.7.0 and later): Use alternate connection method to attach the interrupt handler. <b>x86</b> architecture: Use <b><i>intConnect()</i></b> instead of <b><i>pcilntConnect()</i></b>. <b>PPC</b> architecture: Use <b><i>pcilntConnect()</i></b> instead of <b><i>intConnect()</i></b>.</td></tr> </table> <p><b>Note:</b> This structure member does not exist for the VME-CAN4 driver. Instead the driver exports <i>can4_flags</i> because of parameter structure size restrictions.</p>	0x01	Suppress driver start-up banner.	0x02	Devices of the C331 and C200 family (Driver rev. 2.5.8 and later): Enable 'Delayed Read' support as specified in PCI specification v2.1	0x04	Devices of the C331 and C200 family (Driver rev. 2.7.0 and later): Use alternate connection method to attach the interrupt handler. <b>x86</b> architecture: Use <b><i>intConnect()</i></b> instead of <b><i>pcilntConnect()</i></b> . <b>PPC</b> architecture: Use <b><i>pcilntConnect()</i></b> instead of <b><i>intConnect()</i></b> .
0x01	Suppress driver start-up banner.						
0x02	Devices of the C331 and C200 family (Driver rev. 2.5.8 and later): Enable 'Delayed Read' support as specified in PCI specification v2.1						
0x04	Devices of the C331 and C200 family (Driver rev. 2.7.0 and later): Use alternate connection method to attach the interrupt handler. <b>x86</b> architecture: Use <b><i>intConnect()</i></b> instead of <b><i>pcilntConnect()</i></b> . <b>PPC</b> architecture: Use <b><i>pcilntConnect()</i></b> instead of <b><i>intConnect()</i></b> .						
reserved	Reserved for future use. Set to 0.						

#### 4.1.3.2 VxWorks 6.x (Non-VxBus)

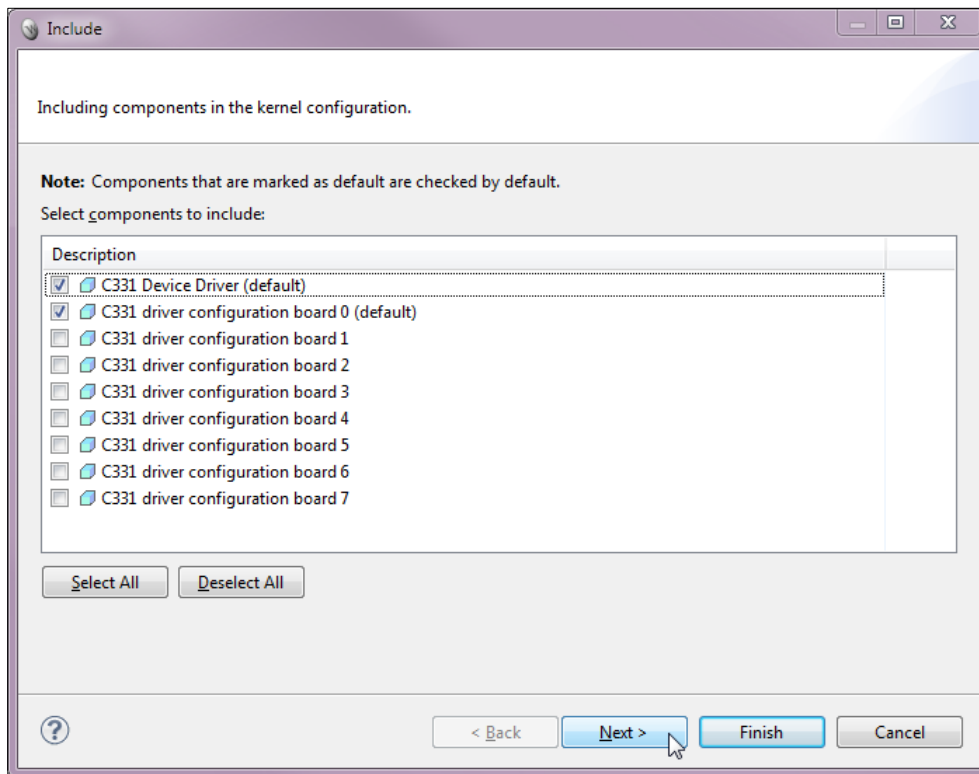
The integration of the VxBus CAN device driver into your VxWorks image and the configuration of the driver parameter is done in the Wind River Workbench via the *Kernel Configuration*. After the driver installation described in the previous chapter you will have a new node in your Workbench VxWorks kernel parameter tree with the name *esd gmbh Driver, Protocol Stacks and Software*.

To integrate the device driver for your CAN Interface include the device driver for this CAN Interface Family (see chapter 1.4) by opening the context menu and choose the option *Include* as shown in the picture below for the C331 Family device driver.

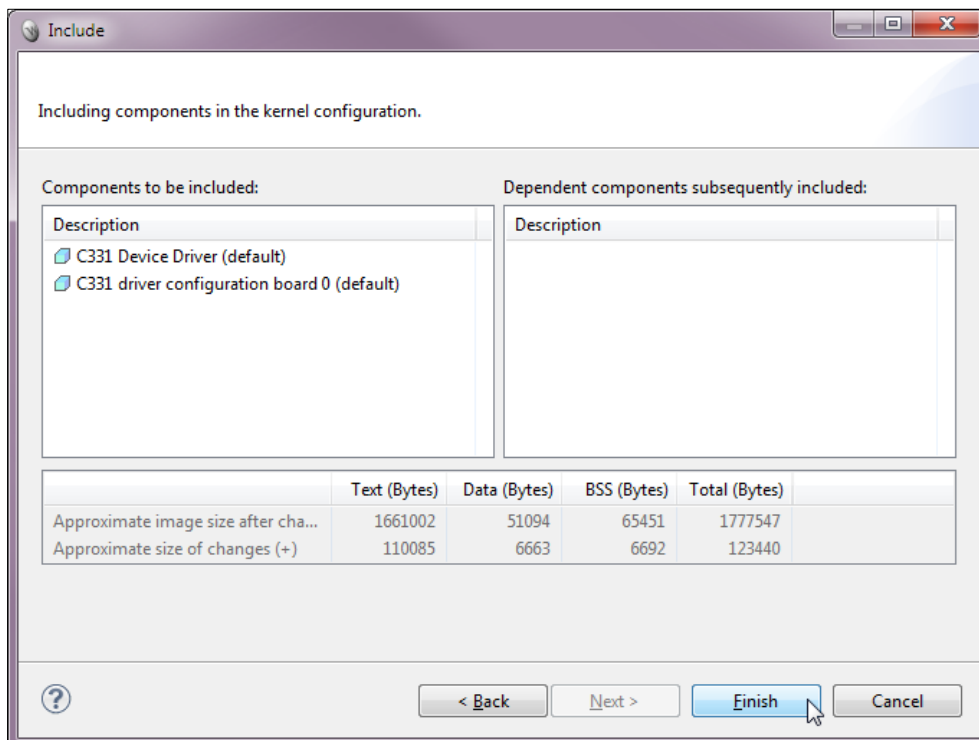


In the next step you will have the opportunity to configure how many boards you want to use.

In the following dialogue box, you have to define the number of CAN Interfaces of this CAN Interface Family you want to support before you click *Next*.



Complete the integration of the CAN device driver by clicking *Finish* in the next dialogue.



Continue configuring some additional options.

Description	Name	Type	Value
esd gmbh Driver, Protocol Stacks and Software	FOLDER_ESD_ROOT		
CAN Protocol Components	FOLDER_ESD_CAN_PROTOCOLS		
CAN fieldbus	FOLDER_ESD_CAN_NETWORK		
CAN board VxBus drivers	FOLDER_ESD_CAN_BOARDS_VXBUS		
CAN board drivers	FOLDER_ESD_CAN_BOARDS		
C200 Family CAN boards	FOLDER_ESD_CAN_C200_BOARDS		
C200 driver configuration board 0 (default)	SELECT_ESD_CAN_C200_BOARD_CFG		
Backend priority of 1st board	ESD_CAN_C200_BACKEND_0	uint	50
Base net of 1st board	ESD_CAN_C200_BASENET_0	uint	0
PCI IRQ vector of 1st board	ESD_CAN_C200_IRQ_VECTOR_0	uint	0xFF
PCI memory base address of 1st board	ESD_CAN_C200_BASEMEM_0	uint	0xFFFFFFFF
C200 driver configuration board 1	INCLUDE_ESD_CAN_C200_BRD_1		
C200 driver configuration board 2	INCLUDE_ESD_CAN_C200_BRD_2		
C200 driver configuration board 3	INCLUDE_ESD_CAN_C200_BRD_3		
C200 driver configuration board 4	INCLUDE_ESD_CAN_C200_BRD_4		
C200 driver configuration board 5	INCLUDE_ESD_CAN_C200_BRD_5		
C200 driver configuration board 6	INCLUDE_ESD_CAN_C200_BRD_6		
C200 driver configuration board 7	INCLUDE_ESD_CAN_C200_BRD_7		
C200 Device Driver (default)	INCLUDE_ESD_CAN_C200		
Autostart behaviour of driver	ESD_CAN_C200_AUTOSTART	BOOL	TRUE
Driver flags	ESD_CAN_C200_FLAGS	uint	1
Target timestamp counter frequency in kHz	ESD_CAN_C200_TIMESTAMP_FREQUENCY	uint	0
C331 Family CAN boards	FOLDER_ESD_CAN_C331_BOARDS		
VME-CAN4 boards	FOLDER_ESD_CAN_ICAN4_BOARDS		
CAN support components (private)	FOLDER_ESD_CAN_SUPPORT		
BSP PCI PnP quirk for PCI CAN boards	FOLDER_ESD_CAN_BSP_PNP QUIRK		
Adapt IRQ and/or PCI BARx addresses to BSP	SELECT_PNP QUIRK_TYPE		
PNP BSP quirk (type 1)	INCLUDE_ESD_CAN_BSP_PNP QUIRK		
PCI IRQ offset	ESD_CAN_PCI_IRQ_OFFSET	uint	0
PCI address offset	ESD_CAN_PCI_ADR_OFFSET	uint	0x0
PNP BSP quirk (type 2) (default)	INCLUDE_ESD_CAN_BSP_PNP QUIRK_2		
PCI IRQ to Vector Mapping	ESD_CAN_PCI_IRQ_ROUTINE	string	irq = INT_NUM_GET(irq);
PCI address offset	ESD_CAN_PCI_ADR_OFFSET	uint	0x0
esd PCI CAN board initialization	INCLUDE_ESD_CAN_PCI_CONFIG		
Basic CAN test application	INCLUDE_ESD_CANTEST		
NTCAN-API support	INCLUDE_ESD_NTCAN		

Synopsis Log

0x01 = No banner, 0x02 = Enable PCI delayed read 0x04 = Non default IRQ connector

Defined at:

[E:\vworks-6.9\target\config\comps\vxWorks\52esd\\_can\\_pci.cdf](#)

Optionally you can adapt the *Backend priority of driver* option (2) which defines the priority of the task for CAN messages post processing to your requirements. Refer to the description of the parameter *prio* in chapter 4.1.3.1 for more details.

Adapt the logical base net of the CAN interface with the option *Base net of board x* (1) to your requirements. If you use just the device driver for one CAN Interface Family leave the default values.

Usually you can leave the *PCI IRQ Vector* (3) set to the default of 0xFF and the *PCI memory base address* set to the default of 0xFFFFFFFF (4). Refer to the description of the parameter *irq* and *base* in chapter 4.1.3.1 for more details.

Set the *Autostart behaviour of driver* option (5) to `FALSE` for a first test and start the driver manually as described in the next chapter. If everything works you can change this parameter to `TRUE` to start the driver with your VxWorks.

Reset bit 0 of the *Driver Flags* option (6) for a first test to start the driver with a start-up banner. If everything works you can set this bit to suppress this banner. Refer to the *Synopsis* box for further driver specific flags.

CAN Interfaces with software timestamp support have the additional option *Target timestamp*

*counter frequency* (7) which defines the frequency of the counter used for software timestamping in kHz. If this parameter is set to 0 the driver probes this frequency (which causes a delay of driver initialization where all interrupts and the scheduler are disabled) otherwise it uses the configured value. For **x86** architectures the high resolution timestamp used runs at processor frequency for most **PPC** targets it runs at 1/4 of the processor frequency.

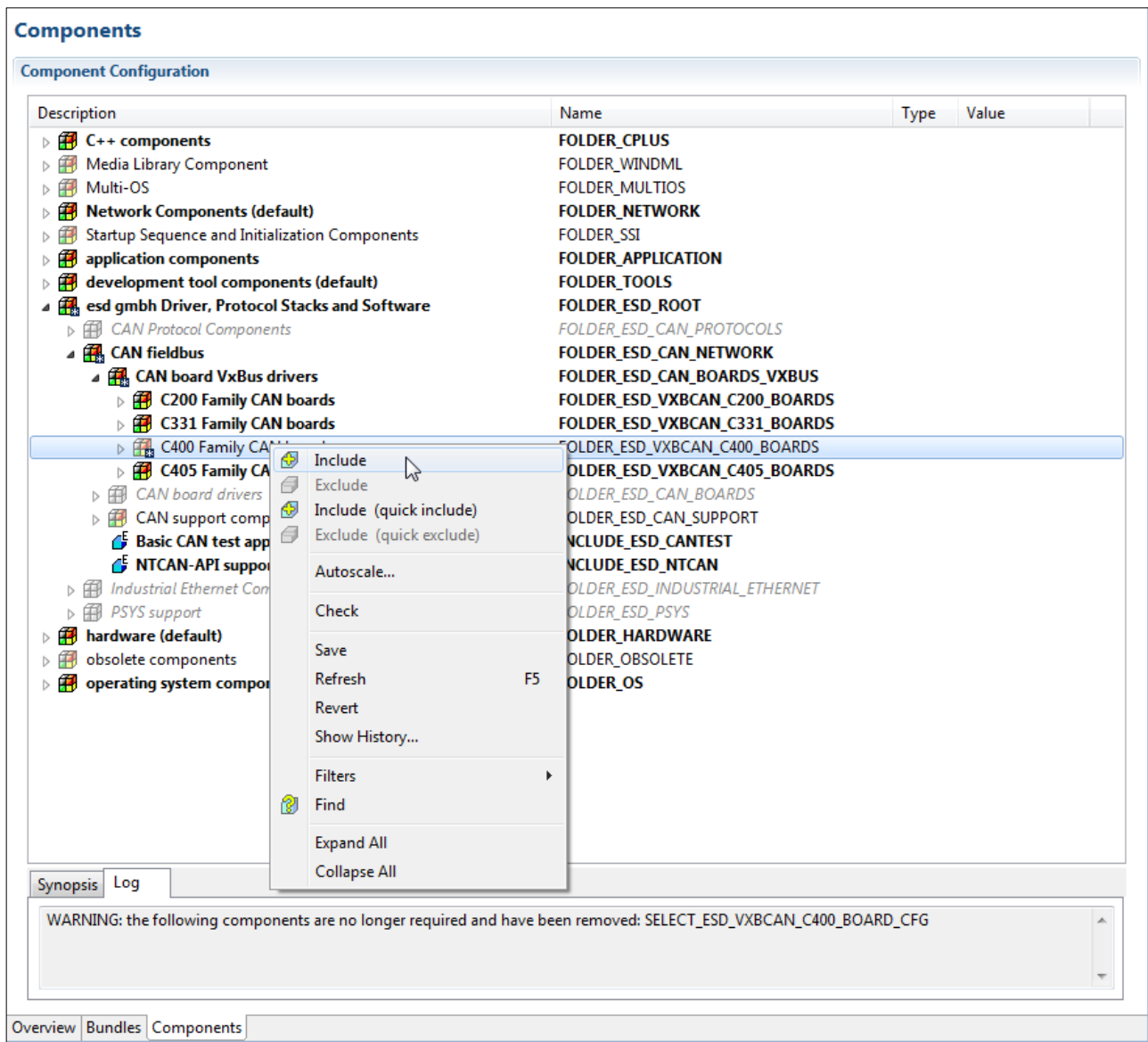
Include *NTCAN-API support* component (12) as this is the common API /1/ for all device drivers you use by your application. This component is automatically included if you include the *Basic CAN Test Application* component (13) which is the *canTest* application described in chapter 4.1.5.3.

Now you can rebuild your VxWorks image.

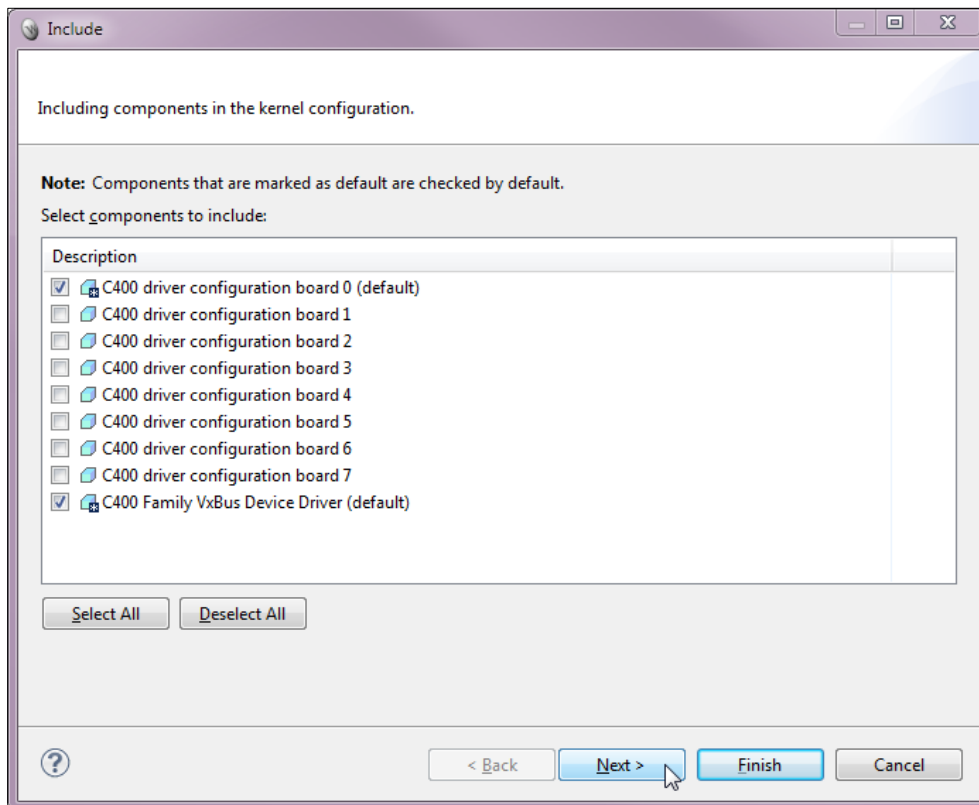
### 4.1.3.3 VxWorks 6.x (VxBus)

The integration of the VxBus CAN device driver into your VxWorks image and the configuration of the driver parameter is done in the Wind River Workbench via the *Kernel Configuration*. After the driver installation described in the previous chapter you will have a new node in your Workbench VxWorks kernel parameter tree with the name *esd gmbh Driver, Protocol Stacks and Software*.

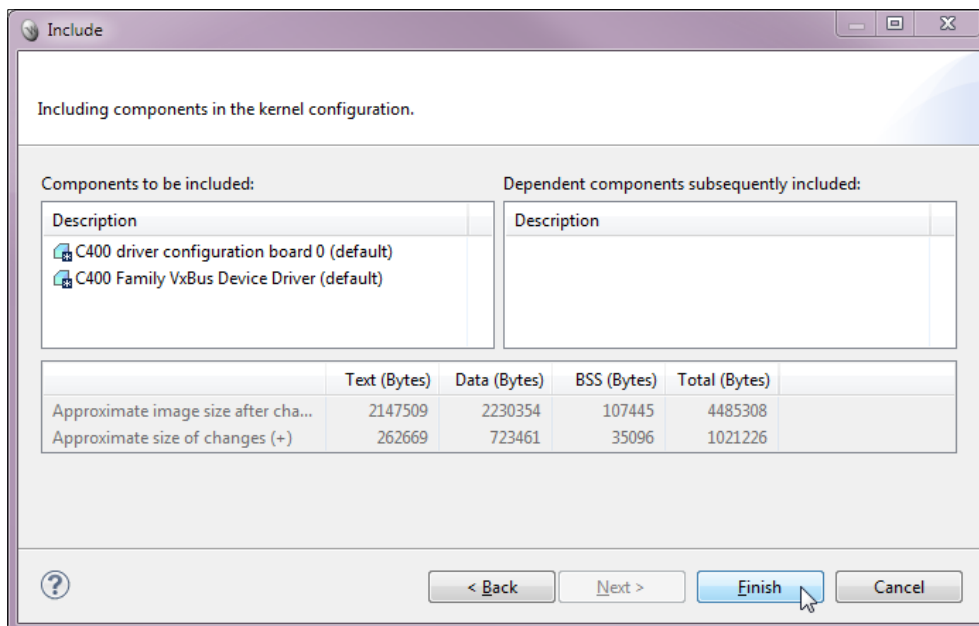
To integrate the device driver for your CAN Interface include the device driver for this CAN Interface Family (see chapter 1.4) by opening the context menu and choose the option *Include* as shown in the picture below for the C400 Family VxBus device driver.



In the following dialogue box you have to define the number of CAN Interfaces of this CAN Interface Family you want to support before you click *Next*.



Complete the integration of the CAN device driver by clicking *Finish* in the next dialogue.



Continue configuring some additional options.

Description	Name	Type	Value
development tool components (default)	FOLDER_TOOLS		
esd gmbh Driver, Protocol Stacks and Software	FOLDER_ESD_ROOT		
CAN Protocol Components	FOLDER_ESD_CAN_PROTOCOLS		
CAN fieldbus	FOLDER_ESD_CAN_NETWORK		
CAN board VxBus drivers	FOLDER_ESD_CAN_BOARDS_VXBUS		
C200 Family CAN boards	FOLDER_ESD_VXBCAN_C200_BOARDS		
C331 Family CAN boards	FOLDER_ESD_VXBCAN_C331_BOARDS		
C400 Family CAN boards	FOLDER_ESD_VXBCAN_C400_BOARDS		
C400 Family Board Configurations (default)	SELECT_ESD_VXBCAN_C400_BOARD_CFG		
C400 driver configuration board 0 (default)	INCLUDE_ESD_VXBCAN_C400_BRD_0		
Base net board 0	ESD_VXBCAN_C400_BASENET_0	uint	0
C400 driver configuration board 1	INCLUDE_ESD_VXBCAN_C400_BRD_1		
C400 driver configuration board 2	INCLUDE_ESD_VXBCAN_C400_BRD_2		
C400 driver configuration board 3	INCLUDE_ESD_VXBCAN_C400_BRD_3		
C400 driver configuration board 4	INCLUDE_ESD_VXBCAN_C400_BRD_4		
C400 driver configuration board 5	INCLUDE_ESD_VXBCAN_C400_BRD_5		
C400 driver configuration board 6	INCLUDE_ESD_VXBCAN_C400_BRD_6		
C400 driver configuration board 7	INCLUDE_ESD_VXBCAN_C400_BRD_7		
C400 Family VxBus Device Driver (default)	DRV_ESD_VXBCAN_C400		
Autostart behaviour of driver	ESD_VXBCAN_C400_AUTOSTART	BOOL	FALSE
Backend priority of driver	ESD_VXBCAN_C400_BACKEND	uint	50
Driver flags	ESD_VXBCAN_C400_FLAGS	uint	0
C405 Family CAN boards	FOLDER_ESD_VXBCAN_C405_BOARDS		
CAN board drivers	FOLDER_ESD_CAN_BOARDS		
CAN support components (private)	FOLDER_ESD_CAN_SUPPORT		
Basic CAN test application	INCLUDE_ESD_CANTEST		
NTCAN-API support	INCLUDE_ESD_NTCCAN		
Industrial Ethernet Components	FOLDER_ESD_INDUSTRIAL_ETHERNET		
PSYS support	FOLDER_ESD_PSYS		
hardware (default)	FOLDER_HARDWARE		
obsolete components	FOLDER_OBSOLETE		

**Synopsis** Log

VxBus device driver for esd CAN-PCI/400, CPCI-CAN/400, CAN-PMC/400 and CAN-PCIe/400 boards

Defined at:

Overview Bundles Components

Adapt the logical base net of the CAN interface with the option *Base net of board x* (1) to your requirements. If you use just the device driver for one CAN Interface Family leave the default values.

Set the *Autostart behaviour of driver* option (2) to `FALSE` for a first test and start the driver manually as described in the next chapter. If everything works you can change this parameter to `TRUE` to start the driver with your VxWorks.

Optionally you can adapt the *Backend priority of driver* option (3) which defines the priority of the task for CAN messages post processing to your requirements. Refer to the description of the parameter *prio* in chapter 4.1.3.1 for more details.

Reset bit 0 of the *Driver Flags* option (4) for a first test to start the driver with a start-up banner. If everything works you can set this bit to suppress this banner. Refer to the *Synopsis* box for further driver specific flags.



CAN Interfaces with software timestamp support have the additional option *Target timestamp counter frequency* (not in the picture above) which defines the frequency of the counter used for software timestamping in kHz. If this parameter is set to 0 the driver probes this frequency (which causes a delay of driver initialization where all interrupts and the scheduler are disabled) otherwise it uses the configured value. For **x86** architectures the high resolution timestamp used runs at processor frequency for most **PPC** targets it runs at 1/4 of the processor frequency.

Include *NTCAN-API support* component (6) as this is the common API /1/ for all device drivers you use by your application. This component is automatically included if you include the *Basic CAN Test Application* component (5) which is the *canTest* application described in chapter 4.1.5.3.

Now you can rebuild your VxWorks image.

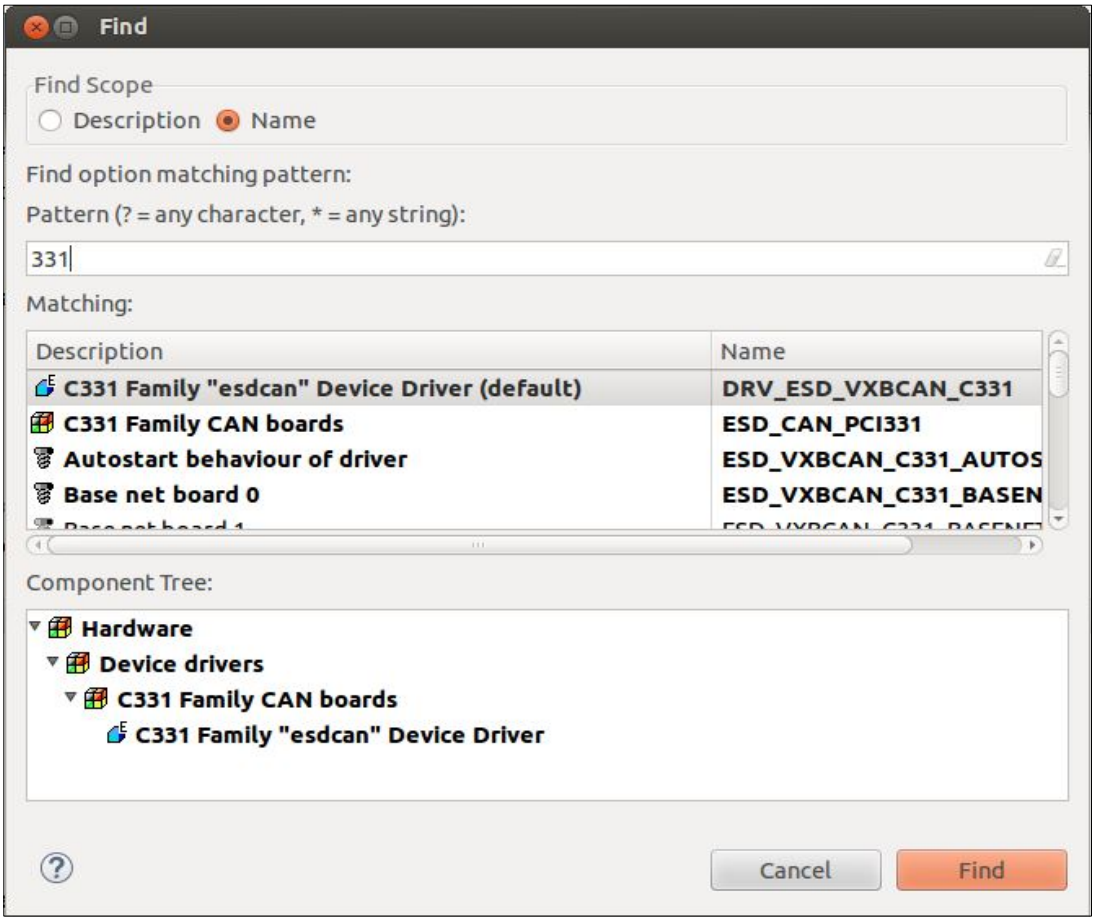
#### 4.1.3.4 VxWorks 7.x (VxBus GEN2)

The integration of the VxBus CAN device driver and related software into your VxWorks image and the configuration of the driver parameter is done in the Wind River Workbench. After the driver installation described in the previous chapter you will have new nodes showing up when building a Source Build. By default all nodes are enabled which a reasonable starting point. Eventually the *cantest* utility can be removed, unless required.

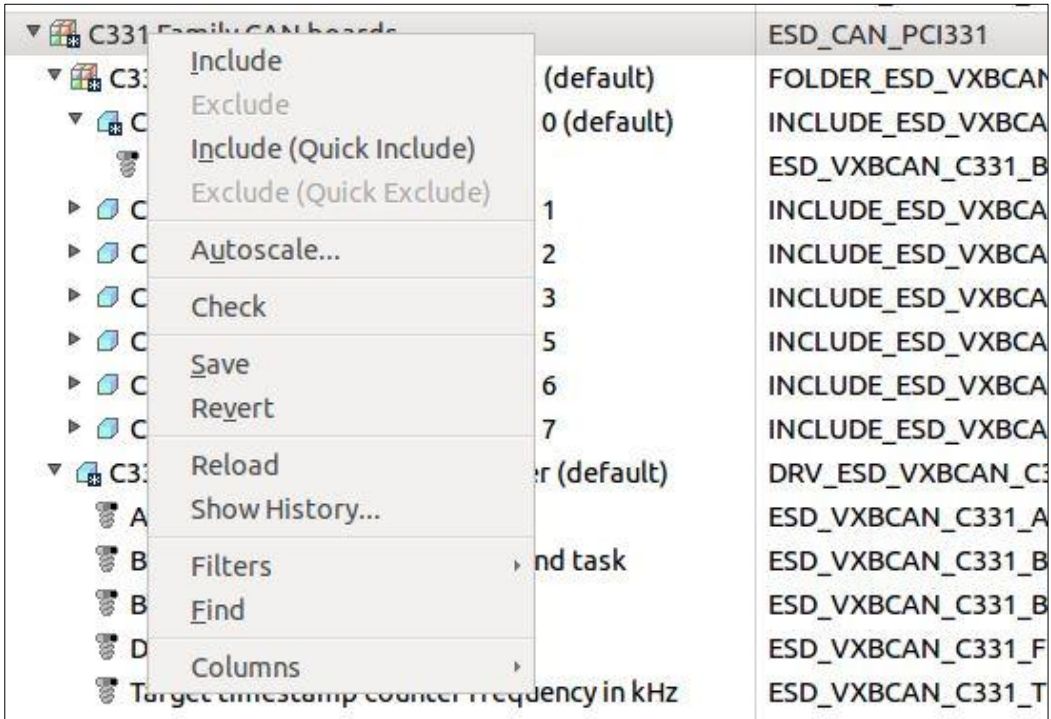
Option	Name	Value
▼ cantest_layer_bin_ATOM_gnu_smp		
▼ Enable esd can test application	ESD_CAN_CANTEST	y
▼ Choose ESD_CAN_CANTEST version	CH_ESD_CAN_CANTEST	esd can test application 3.0.7.0_B_PENTIUM4gnu_smp
▼ esd can test application 3.0.7.0_B_PENTIUM4gnu_smp	ESD_CAN_CANTEST_3_0_7_0_B_PENTIUM4gnu_smp	y
▼ ntcantest_layer_bin_ATOM_gnu_smp		
▼ Enable esd ntcantest library	ESD_CAN_NTCAN	y
▼ Choose ESD_CAN_NTCAN version	CH_ESD_CAN_NTCAN	esd ntcantest library 3.7.4.0_B_PENTIUM4gnu_smp
▼ esd ntcantest library 3.7.4.0_B_PENTIUM4gnu_smp	ESD_CAN_NTCAN_3_7_4_0_B_PENTIUM4gnu_smp	y
▼ pci331_layer_bin_ATOM_gnu_PAE		
▼ Enable esd CAN C331 driver	ESD_CAN_PCI331	y
▼ Choose ESD_CAN_PCI331 version	CH_ESD_CAN_PCI331	esd CAN C331 driver 4.0.0.0_B_PENTIUM4gnu_smp_pae
▼ esd CAN C331 driver 4.0.0.0_B_PENTIUM4gnu_smp_pae	ESD_CAN_PCI331_4_0_0_0_B_PENTIUM4gnu_smp_pae	y

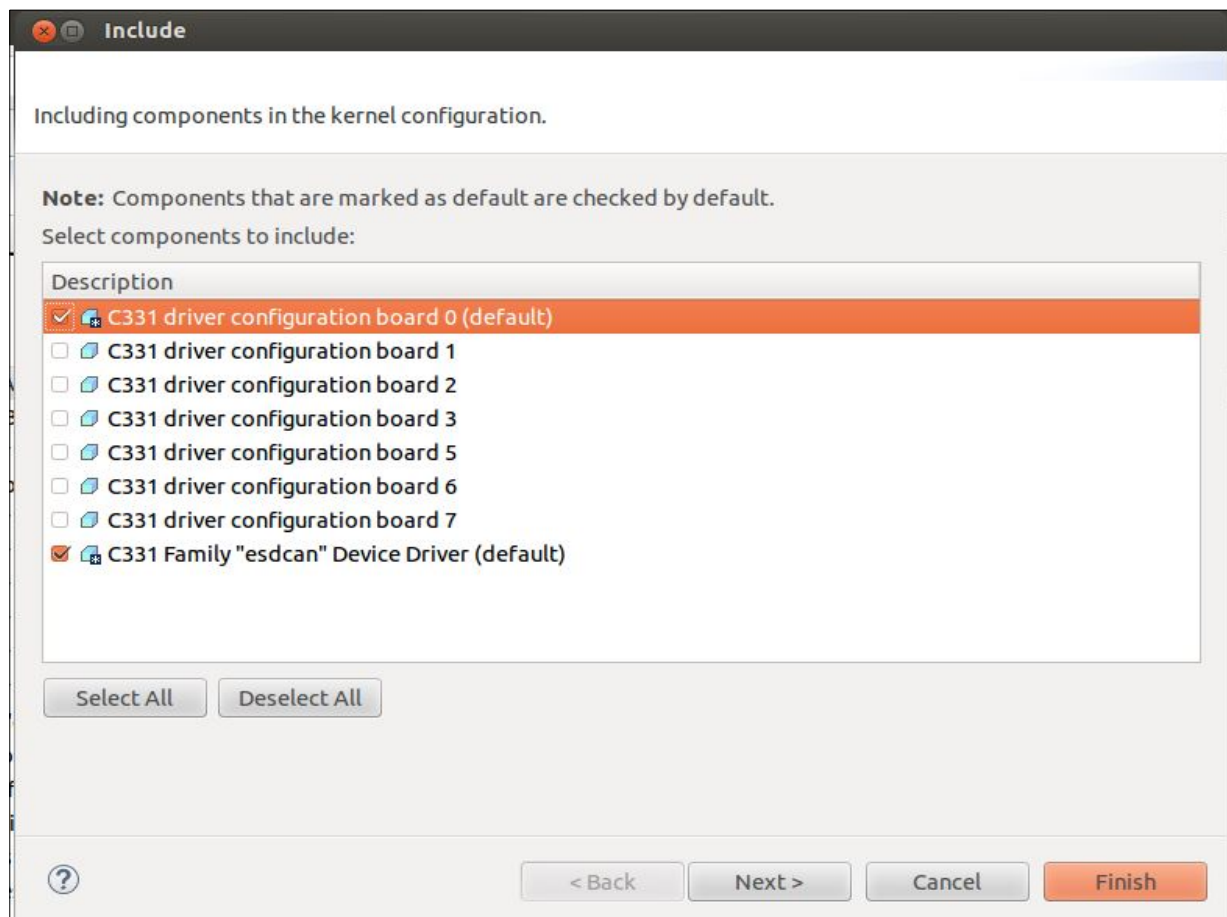
There are a number of combinations the supported by the driver with the appropriate combination (e.g. NEHALEM, SMP, LP64) selected automatically. Don't be confused by the name "...bin\_ATOM\_..." as the layer base name which is generated during the driver build stage. Only the chosen version (...PENTIUM4gnu\_smp in the sample above) is of any meaning at this point!

Based on that source build, a VxWorks Image can be build. When opening the *Kernel Configuration*, you can search for the can driver, library or tools:



By default, the driver and software is **not** included in the kernel image. Choose "include" on the driver, library and the test application to be able to test the installation.





Once all components have been included the configuration should look like this:

Device drivers	FOLDER_DRIVERS		
ADC	FOLDER_DRIVERS_ADC		
Busses	FOLDER_DRIVERS_BUS		
C331 Family CAN boards	ESD_CAN_PCI331		
C331 Family Board Configurations (default)	FOLDER_ESD_VXBCAN_C331_BOARD_0		
C331 driver configuration board 0 (default)	INCLUDE_ESD_VXBCAN_C331_BRD_0		
Base net board 0	ESD_VXBCAN_C331_BASENET_0	uint	0
C331 driver configuration board 1	INCLUDE_ESD_VXBCAN_C331_BRD_1		
C331 driver configuration board 2	INCLUDE_ESD_VXBCAN_C331_BRD_2		
C331 driver configuration board 3	INCLUDE_ESD_VXBCAN_C331_BRD_3		
C331 driver configuration board 5	INCLUDE_ESD_VXBCAN_C331_BRD_5		
C331 driver configuration board 6	INCLUDE_ESD_VXBCAN_C331_BRD_6		
C331 driver configuration board 7	INCLUDE_ESD_VXBCAN_C331_BRD_7		
C331 Family "esdcan" Device Driver (default)	DRV_ESD_VXBCAN_C331		
Autostart behaviour of driver	ESD_VXBCAN_C331_AUTOSTART	BOOL	FALSE
Backend affinity of driver backend task	ESD_VXBCAN_C331_BKND_AFFINITY	uint	0
Backend priority of driver	ESD_VXBCAN_C331_BKND_PRIO	uint	50
Driver flags	ESD_VXBCAN_C331_FLAGS	uint	1
Target timestamp counter frequency in kHz	ESD_VXBCAN_C331_TIMESTAMP_FREQ	uint	0

You can modify the parameter as described for VxWorks 6.x (see chapter 4.1.3.3).

### 4.1.4 Driver Start

This chapter describes the CAN device driver start for the various versions of VxWorks after the CAN driver is integrated into the VxWorks image and configured.

#### 4.1.4.1 VxWorks 5.x

If you need a specific configuration we recommend to use `caninit.c` as a template, change the source code according to your needs, compile and load it after the driver binary.

If you use a modified `caninit.c` to start your driver you should call **`xxxStart()`** otherwise you have to call the **`xxx_install()`** routine which is exported by the driver with an array of initialized `XXX_CAN_INFO` structures. In both cases 'xxx' is the CAN Family device driver specific prefix as described in chapter 1.4. In case of a CAN device driver which comes as part of a VxWorks BSP

for an **esd** embedded board, the driver is usually started just with **`canStart()`**.

If the driver start-up banner is enabled you will see some informations to the hardware/software environment similar to the example below:

```
C200: "CAN_PCI266" with 2 Nets identified
C200: Hardware-Version = 1.0.00
C200: Firmware-Version = 0.0.00
C200: Driver-Version   = 2.5.08
C200: Net 0 successfully created on card 0
C200: Net 1 successfully created on card 0
```

In the example above it is the start of a CAN-PCI/266 CAN Interface which belongs to the C200 Family. Refer to chapter 1.4 for the prefix overview of other device drivers.

#### 4.1.4.2 VxWorks 6.x (Non-VxBus)

For a first test you should configure the *Autostart behaviour of driver* option in the Wind River Workbench to `FALSE` so you can start the driver manually on the target shell with **`xxxStart()`** where 'xxx' is the CAN Family device driver specific prefix as described in chapter 1.4. In case of a CAN device driver which comes as part of a VxWorks BSP for an **esd** embedded board, the driver is usually started just with **`canStart()`**.

If the driver start-up banner is enabled via the *Driver Flags* option in the Wind River Workbench you will see some informations to the hardware/software environment similar to the example below:

```
C331: Using IRQ/Vector* (0x0b/0x158) shared for card 0
C331: "CAN_PCI331" with 2 Nets identified
C331: Hardware-Version = 1.1.00
C331: Firmware-Version = 0.c.1e
C331: Driver-Version   = 2.7.00
C331: Timestamp frequency is 1476535544 Hz (Software)
C331: Net 0 successfully created on card 0
C331: Net 1 successfully created on card 0
C331: CAN 2.0A firmware active
C331: PCI delayed read: Disabled (CNTRL: 0x18780fd7)
```

In the example above it is the start of a CAN-PCI/331 CAN Interface which belongs to the C331 Family. Refer to chapter 1.4 for the prefix overview of other device drivers.

If there are no problems you can configure the *Autostart behaviour of driver* option in the Wind River Workbench to `TRUE` so the driver is started automatically every time the VxWorks image is booted.

#### 4.1.4.3 VxWorks 6.x/7.x (VxBus)

For a first test you should configure the *Autostart behaviour of driver* option in the Wind River Workbench to `FALSE` so you can start the driver manually on the target shell with **`xxxStart()`** where 'xxx' is the CAN Family device driver specific prefix as described in chapter 1.4. In case of a CAN device driver which comes as part of a VxWorks BSP for an **esd** embedded board, the driver is usually started just with **`canStart()`**.

If the driver start-up banner is enabled via the *Driver Flags* option in the Wind River Workbench you will see some informations to the hardware/software environment similar to the example below:

```
C405: "CAN_PCI405" with 4 Nets identified
C405: Hardware-Version = 1.3.01
C405: Firmware-Version = 3.8.16
C405: Driver-Version   = 3.9.00 (SMP / VxBus Rev. 4)
C405: Net 0 successfully created on card 0
C405: Net 1 successfully created on card 0
C405: Net 2 successfully created on card 0
C405: Net 3 successfully created on card 0
```

In the example above it is the start of a CAN-PCI/405 CAN Interface which belongs to the C405 Family. Refer to chapter 1.4 for the prefix overview of other device drivers.

If there are no problems, you can configure the *Autostart behaviour of driver* option in the Wind River Workbench to `TRUE` so the driver is started automatically every time the VxWorks image is booted.



### 4.1.5 Miscellaneous

This chapter covers several topics which should preclude problems building the VxWorks image and using the NTCAN architecture on VxWorks.

#### 4.1.5.1 Unresolved Symbols Building the VxWorks Image

Using the NTCAN library the linker might fail with unresolved symbols like `__udivdi3` or `__umodi3`. The reason for this is that in order to support 64 bit arithmetic (for the NTCAN timestamps) on a 32 bit CPU the GCC compiler requires some helper functions which are part of the GNU GCC run-time libraries which is not included in any case. To prevent error messages about unresolved symbols you have to include the GNU GCC run-time libraries in your VxWorks image by defining `INCLUDE_GNU_INTRINSICS`. Please refer to the *VxWorks Kernel Programmer's Guide* for more details.

#### 4.1.5.2 Number of Available NTCAN Handles

A CAN driver supports up to 1024 individual CAN handles but the maximum number is also limited by the maximum number of open handles of the VxWorks system. The related VxWorks configuration parameter is `NUM_FILES` with a default value of 50. Changing the value of `NUM_FILES` can have some other undesirable effects on the system if certain limits are not observed. In VxWorks 5.4.x setting `NUM_FILES` to a value greater than 256 could cause problems for code that use the ***select()*** function. The `FD_SETSIZE` was limited to 256 in this version. Starting with VxWorks 5.5 this limitation was removed but you might still adapt the value of `FD_SETSIZE` if opening a NTCAN handle failed with a resource error.

#### 4.1.5.3 Test Program 'canTest'

The driver package comes with the test program ***canTest*** which can be either optionally configured into your VxWorks image via the Workbench (VxWorks 6.x) or can be loaded with the VxWorks target shell (VxWorks 5.x). With the help of this program you can do basic functionality checks of the CAN interface. The program and its parameters are described in more detail in */1/*.



**Note:**

As VxWorks allows less parameters in the command line as required by *canTest*, the parameters have to be provided separated by spaces as a single string in quote signs.

#### 4.1.5.4 Unexpected Behaviour of Software Timestamps

With some VxWorks versions and BSPs the driver may show unexpected behaviour with software timestamps. In this case the measured timestamp frequency as shown in the driver's startup banner may be negative or far beyond the expected value. Or the time differences calculated between to received CAN frames have surprising values.

The reason for that behaviour can be for instance on the **x86** architecture that the high resolution time stamp counter (TSC) is reset every timer tick. This is done in some BSPs because WindView needs that behaviour. For further information refer to this website:

<http://borkhuis.home.xs4all.nl/vxworks/pc-bsp.html#pc-bsp.5>

#### 4.1.5.5 Correct Interpretation of Error Codes Returned by the Driver

The interpretation of error codes returned by the driver is guided by a multi level approach. The first resource where you should search for a specific error code is the header file `ntcan.h` that is delivered in the driver package. This file contains definitions of macros for the NTCAN error codes for example:

```
#define NTCAN_SUCCESS                0
#define NTCAN_RX_TIMEOUT             0x00001001
```

Please observe that the upper word of all defined errors is zero. In `ntcan.h` you can search for a certain error code value and map it to the macro name. The meaning of these macro names are described in **chapter 'Return Codes' in /1/**.

If you did not find the macro name for the specific error code you want to look up then consider the error numbers the NTCAN API borrows from the system via the `errno.h` file, for example:

```
#define NTCAN_INVALID_PARAMETER      EINVAL
#define NTCAN_INVALID_HANDLE        EBADF
```

Therefore on the second level you have to lookup that error code in the VxWorks `errno.h` that is installed in the target tree of your VxWorks installation at `target/h/errno.h`. This way a return code of `0x00000016` maps to `EINVAL` and therefore in turn to `NTCAN_INVALID_PARAMETER`.

Another kind of error code could occur with the upper word of the unsigned error code not being zero. To decode the VxWorks error code of `0x000d0003` you should refer additionally to the file `vwModNum.h` also being in the `target/h` directory.

```
From <vwModNum.h>:
#define M_iosLib (13 << 16)
```

```
From <errno.h>:
#define ESRCH 3 /* No such process */
```

This means that the VxWorks `iosLib` wants to tell us that there is "no such process". Our driver code seems not to be reached since already the `iosLib` returns an error code. From this we may deduce that something is wrong with the handle itself.

The most common error code returned by the `<drvname>_install()` routine is 19 i. e. `ENODEV`. That means that the install routine did not find all boards that were requested to initialize by the `<drvsig>_CAN_INFO` structure array.

The described method applies if you need to translate NTCAN error codes by manually. Since revision 3.3.0 the NTCAN library provides the function `canFormatError()` that should be used by an application to translate error codes into error strings.

### 4.1.5.6 Support of the CAN Extended Frame Format

All CAN Interfaces support the Extended Frame Format (29 Bit CAN-IDs) according to the CAN 2.0B specification [2]. The active CAN Interfaces which belong to the C331-, C360-, C331I- and ICAN4-Family (see chapter 1.4) are usually configured in a 29-Bit passive mode which means that CAN messages in the *Extended Frame Format* are tolerated but can not be received or transmitted by an application.

Starting with firmware revision 0.C.09 of these active CAN Interfaces the firmware can be switched between this 11 bit active/29 bit passive factory mode and the 11/29 bit active mode.

The firmware operation mode is indicated in the start-up banner and can be queried at runtime on the target shell with

```
xxx_switch(0, net)
```

where '**xxx**' is the driver specific prefix defined in chapter 1.4 and *net* is a logical net number assigned to one of the CAN Interface's ports.

You can toggle between the two modes of operation while the driver is running with the same command changing the first parameter.

```
xxx_switch(1, net)
```



**Note:**

After the operation mode is toggled you have to reboot your target to make the new configuration active.



### 4.1.6 Troubleshooting Hints

The following paragraphs will detail some common problems that may get in the way of a successful driver start. Also here you should find some advice to diagnose the problem and information about the internal working of the driver as far as installation is concerned.

Unfortunately the hardware abstraction under VxWorks is in a state of flow where the way the things are done is very dependent on the VxWorks version, the CPU-Architecture and even the BSP of your board. This is the reason why it is advisable to provide the exact platform you will be working with (VxWorks version, CPU-Architecture, BSP type and version, SMP requirements) when you order a CAN driver.

The first thing you must be sure of is to use the right driver binary for your architecture. If this is guaranteed the reason for the driver not working should only be a configuration problem.

If you have a driver that is installed in the VxWorks target tree and configured via the Workbench it should show up and be selectable in the components tab of the kernel configuration pane. If that is not the case you may use an CPU architecture for your kernel that the driver doesn't support. Look for `libesdCAN.a` in the target tree of your VxWorks installation. There must be one for your target architecture or the Workbench won't show the driver as selectable. Review chapter 4.1.2.2 for installation information.

The CAN driver needs to be able to access the board's registers and / or memory spaces. Also the driver tries to connect an interrupt service routine to handle the interrupts generated by the board. The chapter 4.1.6.4 deals with problems related to board access and the chapter 4.1.6.5 deals with problems with the interrupt handling.

In the case of the driver not starting successfully (`<drvname>Start()` calling `<drvname>_install()` failed) or not working properly there may be either issues with the access to the board or with the interrupt connection. Depending on the kind of CAN board the behaviour is a little bit different.

For passive boards (no local CPU) like the C200 family boards the driver start may seem to be successful. But the IRQ is not exercised while starting the driver. Therefore an error or crash may appear later when for example `canWrite()` is called the first time. This is the reason you should test a successful execution of `canWrite()` or `canRead()` for passive boards immediately with `canTest` from the target shell.

As a rule of thumb you can deduce that it is an address translation / board access issue if the driver start fails immediately. If the problems occur later (i. e. single CAN transfer successful) or crash it is more probably an IRQ connection issue.

For active boards (with local CPU or FPGA) like the C331 family boards the IRQ is exercised during installation. At first the driver needs access to the registers or dual-ported RAM areas of the boards then it connects the IRQ handler. The driver start should fail if anything goes wrong.

#### 4.1.6.1 Where to Implement Needed Configuration Changes

A 2.x version driver consists of two parts. One part is the prebuilt `<drvname>.sys` binary-object (the driver core), the other part is the `caninit.c` source file that should be compiled as Downloadable Kernel Module (DKM). This is a wrapper around the driver core that provides configuration information to the driver and a simple `<drvname>Start()` function. We deliver for convenience a precompiled default version of `caninit.c` named `<drvname>.ini`.

Both should be loaded into your VxWorks target with `ld()`. An example load script is delivered as `ldcan`. Then you can use `<drvname>Start()` to get the driver running. If the default configuration fails you should end up here in the troubleshooting chapter.

Now you can try to add any changes in the configuration mentioned in the following paragraphs to the `caninit.c` file and compile this file **yourself** in your build environment. Then try the start of the driver again with the prebuilt `<drvname>.sys` and your self-built `caninit.o`.

If you have a 2.x version driver that is configured using the Workbench you can change all parameters by setting macros from the Workbench GUI. The VxWorks kernel build will integrate the prebuilt driver core `<drvname>.o` from the `libesdCAN.a` into the kernel image. Also a source file `caninit_<drvname>.c` (called a configlette) from the target tree is compiled using the macros defined via the GUI to change the configuration. It works the same way as the `caninit.c`. As a last resort you may edit the `caninit_<drvname>.c` file directly (perhaps adding a BSP specific IRQ vector translation routine via the `<drvname>GetIrqVector` function pointer?).

The 3.x version drivers are always configured via the VxWorks Workbench. But there are two build variants of the driver for a supported hardware. One variant is configured using a configlette like it is described for a 2.x version driver in the previous chapter. This kind of driver is included using macros named like `"*_ESD_CAN_*`".

The other variant relies completely on the VxBus Hardware Abstraction Layer and is included using macros named like `"*_ESD_VXBCAN_*`". Therefore, the VxBus implementation of your board must work correctly to make the driver work, i. e. if the driver doesn't work you need to fix the VxBus implementation of your BSP to provide the correct information to the driver. VxBus related issues are discussed in chapter 4.1.6.6.

#### 4.1.6.2 Public Interface of the Version 2.x Driver Core

The following table shows the functions and variables that comprise the public API of the version 2.x driver core for the driver families C331I, C331, C200 and C200I as examples. The API for other board families can be deduced from that table. The next chapters will refer to some of these functions and variables and explain their usage.

<code>c331i_install()</code>	<code>c331_install()</code>	<code>c200_install()</code>	<code>c200i_install()</code>
<code>c331i_uninstall()</code>	<code>c331_uninstall()</code>	<code>c200_uninstall()</code>	<code>c200i_uninstall()</code>
<code>c331i_switch()</code>	<code>c331_switch()</code>		
	<code>c331PciOffset</code>	<code>c200PciOffset</code>	
<code>c331iIrqOffset</code>	<code>c331IrqOffset</code>	<code>c200IrqOffset</code>	
<code>c331iGetIrqVector</code>	<code>c331GetIrqVector</code>	<code>c200GetIrqVector</code>	<code>c200iGetIrqVector</code>
<code>c331iAttachIrqHandler</code>	<code>c331AttachIrqHandler</code>	<code>c200AttachIrqHandler</code>	<code>c200iAttachIrqHandler</code>
<code>c331iDetachIrqHandler</code>	<code>c331DetachIrqHandler</code>	<code>c200DetachIrqHandler</code>	<code>c200iDetachIrqHandler</code>

##### Notes:

- The `<drvname>_switch()` function is only provided for active boards with onboard microcontroller.

- The `<drvname>PciOffset` variable is only present for boards on the PCI bus.
- The `<drvname>*IrqHandler` and `<drvname>GetIrqVector` variables are present since version 2.7.x.

#### 4.1.6.3 Public Interface of the VME-CAN4 Driver Core

The public interface of the VME-CAN4 driver core differs from the previous table and is detailed in the following table.

<b><code>can4_install()</code></b>	install routine
<b><code>can4_uninstall()</code></b>	uninstall routine
<b><code>can4_switch()</code></b>	switch CAN-2.0A / CAN-2.0B (see 4.1.5.6)
<b><code>can4_flags</code></b>	driver flags (see 4.1.3.1)
<b><code>can4_tickFreq</code></b>	timestamp frequency (see 4.1.3.1)
<b><code>can4_irqNumOffset</code></b>	like <code>&lt;drvname&gt;IrqOffset</code>
<b><code>can4_pflntVectorGet</code></b>	like <code>&lt;drvname&gt;GetIrqVector</code>
<b><code>can4_irqConnectMode</code></b>	select the used interrupt connect routine
<b><code>can4_pflntConnect</code></b>	pointer to a user provided interrupt connect routine
<b><code>can4_irqDisconnectMode</code></b>	select the used interrupt disconnect routine
<b><code>can4_pflntDisconnect</code></b>	pointer to a user provided interrupt disconnect routine

The variables `can4_flags` and `can4_tickFreq` are implemented as global variables because they don't need to be specified per board but only for the whole driver. Also there was no space left in the per board parameter structure.

For further information refer to the comments in the `caninit.c` / `caninit_ican4.c` files.

#### 4.1.6.4 Address Translation and Board Access Issues

All addresses that you specify in the `base` member of the `<drvsig>_CAN_INFO` structure are always bus addresses. I. e. depending on the bus type the board is attached to these addresses will be ISA-Bus addresses for ISA-Boards, PCI-Bus addresses for PCI-Boards and VME-Bus addresses for VME-boards.

##### 4.1.6.4.1 ISA-Bus Address Translation

The driver binary `<drvname>.sys` uses the following functions to access the board on the ISA Bus:

- **`sysOutWord()`**
- **`sysOutByte()`**
- **`sysInWord()`**
- **`sysInByte()`**

Any special accesses or address translation must be handled by these functions provided by your BSP. I. e. calling **`sysInWord()`** with a parameter `0x1e8` should generate an ISA-Bus word (16 bit) read to address `0x1e8`. Please refer to the board's hardware manual for the required number of I/O addresses and make shure that no ISA address ranges overlap with other devices.

### 4.1.6.4.2 PCI Bus Access Issues

The driver must be able to access the board on the PCI Bus. To test the accessibility you can find the board manually from the target shell and try to access it with the memory dump command **d()**.

**Attention:**

To have the PCI show functions used below included in your kernel image you need to include different modules into your kernel image build depending on the VxWorks version and bus configuration:

- For VxWorks versions before VxWorks 6.x or kernel images without VxBus support you must include `INCLUDE_PCI_CFGSHOW`.
- For VxWorks versions from VxWorks 6.x and later if you have VxBus enabled (`INCLUDE_VXBUS`) you must include `INCLUDE_VXBUS_SHOW` and `INCLUDE_PCI_BUS_SHOW`. Be aware that having VxBus enabled breaks the old PCI show module enabled with `INCLUDE_PCI_CFGSHOW`.

Use **pciConfigTopoShow()** to display all PCI devices VxWorks is aware of. This will also show you if all PCI buses have been detected that are implemented on your host board.



Watch out for PCI bridges that may need to be explicitly enabled when configuring your VxWorks kernel.

An example is the PCI-PCI bridge located on the PMCSPAN extension board for Emerson (former Motorola) MVME boards (i. e. MVME5110) that has to be explicitly included in the PCI Autoconfiguration run for that BSP.

After you checked that all expected PCI buses are present you may use **pciDeviceShow()** with a bus number parameter to look for **esd** CAN interface boards. Refer to chapter 1.5.1 to find the PCI IDs you have to look for. **pciDeviceShow()** only shows the PCI Vendor ID and the PCI Device ID. To fully identify the board you need to display also the PCI Subsystem Vendor ID and the PCI Subsystem Device ID.

Here follows some example output of a successful search for a CAN-PCI/331 board. The data used for the identification is marked **red** there:

```
-> pciConfigTopoShow
[0,1,0] type=P2P BRIDGE to [1,0,0]
      base/limit:
          mem= 0xe8000000/0xffffffff
          preMem=0xffff0000/0x000fffff
          I/O= 0xd000/0xdfff
      status=0x2220 ( 66MHZ DEVSEL=1 MSTR_ABORT_RCV )
      command=0x0007 ( IO_ENABLE MEM_ENABLE MASTER_ENABLE )
[1,0,0] type=DISP_CNTRLR
      status=0x0210 ( CAP DEVSEL=1 )
      command=0x0007 ( IO_ENABLE MEM_ENABLE MASTER_ENABLE )
      bar0 in 32-bit mem space @ 0xe8000000
[0,18,0] type=NET_CNTRLR
      status=0x0210 ( CAP DEVSEL=1 )
      command=0x0007 ( IO_ENABLE MEM_ENABLE MASTER_ENABLE )
      bar0 in I/O space @ 0x0000e800
      bar1 in 32-bit mem space @ 0xf5202000
```

```

[0,19,0] type=UNKNOWN (0x80) BRIDGE
        status=0x0280 ( FBTB DEVSEL=1 )
        command=0x0003 ( IO_ENABLE MEM_ENABLE )
        bar0 in 32-bit mem space @ 0xf5200000
        bar1 in I/O space @ 0x0000ec00
        bar2 in 32-bit mem space @ 0xf5100000
        bar3 in 32-bit mem space @ 0xf5000000
        bar4 in 32-bit mem space @ 0xf5201000
value = 0 = 0x0
-> pciDeviceShow 0
Scanning functions of each PCI device on bus 0
bus      device    function  vendorID  deviceID  class/rev
    0         0         0    0x1106    0x0598  0x06000004
    0         1         0    0x1106    0x8598  0x06040000
    0        18         0    0x10b7    0x9200  0x02000078
    0        19         0    0x10b5    0x9050  0x06800001
value = 0 = 0x0
-> pciHeaderShow 0,19,0
vendor ID =                0x10b5
device ID =                0x9050
command register =         0x0003
status register =          0x0280
revision ID =              0x01
class code =               0x06
sub class code =           0x80
programming interface =    0x00
cache line =               0x08
latency time =             0x00
header type =              0x00
BIST =                     0x00
base address 0 =           0xf5200000
base address 1 =           0x0000ec01
base address 2 =           0xf5100000
base address 3 =           0xf5000000    <- BAR3 base needed in 166
base address 4 =           0xf5201000
base address 5 =           0x00000000
cardBus CIS pointer =      0x00000000
sub system vendor ID =     0x12fe
sub system ID =            0x0001
expansion ROM base address = 0x00000000
interrupt line =           0x09
interrupt pin =            0x01
min Grant =                0x00
max Latency =              0x00
value = 0 = 0x0

```

This dump shows the conditions after a PCI PnP (Plug-and-play) run has successfully configured device addresses.

After you have found the device in the PCI config space you could try to access the device using the **d()** command from the target shell. Before trying that the next chapter will explain if an address translation is needed and then show how to access the device.

#### 4.1.6.4.3 PCI Bus Address Translation

In most cases for a **x86** PC-style motherboard the CPU address (*cpuAdrs*) to access the PCI board at its configured PCI address (*pciAdrs*) is the same. But on some (often PowerPC based) host boards the *cpuAdrs* differs from the *pciAdrs* and the following equation needs to be applied:

$$cpuAdrs = pciAdrs + pciOffset$$

Therefore the driver reads the *pciAdrs* from the BARx registers of the PCI device on the CAN interface board and calculates the *cpuAdrs* it will use to access the device by adding *pciOffset*.

Our driver binary `<drvname>.sys` exports a variable `<drvname>PciOffset` that is used by the driver in this calculation. Therefore you need to set up that variable to a value that fits to your BSP. This is normally done in the `caninit.c` file before the `<drvname>_install()` function is called.



For a non-VxBus driver configured from the VxWorks Workbench you should enable the `INCLUDE_ESD_CAN_BSP_PNP_QUIRK` which will allow you to set the offset `<drvname>PciOffset` using the macro `ESD_CAN_PCI_ADR_OFFSET` in the configlet `caninit_<drvname>.c` located in the target tree.

To find the right value for the `<drvname>PciOffset` you should refer to the documentation of your target BSP. To test the validity of that value you could test the access from the VxWorks target shell.

#### CAN-PCI/331 Access Test Example

The following example is for the CAN-PCI/331 board. It is also valid for all boards of the C331 family (consider the different PCI IDs for the other boards of that family). You could verify the *c331PciOffset* by first extracting the BAR3 value from the output of `pciHeaderShow()` (how to find the device and get that value is described in 4.1.6.4.2), then calculate the resulting *cpuAddr* for that area to use the `d()` command to look into the board's memory. From section 4.1.6.4.2 we find the *pciAddr* for BAR3 being `0xf5000000`. With a *c331PciOffset* of zero in this case we get also `0xf5000000` for the *cpuAddr* here.

```
-> d 0xf5000000
NOTE: memory values are displayed in hexadecimal.
0xf5000000: 4000 fe07 0000 1800 1000 4f43 444c 2020 *.@.....COLD *
0xf5000010: 0000 0000 0000 4f4e f945 ff00 00fa 7c35 *.....NOE.....5|*
0xf5000020: 8f7f 0400 0070 0053 fc66 7c35 4100 0000 *....p.S.f.5|.A.*
0xf5000030: 7c35 0400 2000 0970 fa41 6e00 ea43 4c00 *5|... p.A..nC..L*
0xf5000040: d832 d832 c851 faff 0f70 f941 4000 000f *2.2.Q...p.A..@..*
0xf5000050: fa43 7e00 d910 c851 fcff b94e 4000 000f *C...~..Q...N..@..*
0xf5000060: 7c35 0700 5c00 7c35 0700 6000 fc23 aaaa *5|...`5|...`#...*
0xf5000070: aaaa 0f00 f0ff fc23 5555 5555 0300 f0ff *.....#.UUUU....*
0xf5000080: b90c aaaa aaaa 0f00 f0ff 0c67 7c35 0500 *.....g.5|..*
0xf5000090: 5c00 7c35 0500 6000 7c35 0000 1c00 7c35 *.`5|...`5|....5|*
0xf50000a0: 4200 1e00 fa4e 5200 0000 0000 0000 0000 *.B..N..R.....*
0xf50000b0: 0000 0000 0080 f07f 0740 303c 0740 305c *.....@.<0@.\0*
0xf50000c0: 0000 0000 0090 307c 0091 307c 0092 307c *.....|0..|0..|0*
0xf50000d0: 7c25 07c0 b078 4800 2f00 c000 0100 754e *%|..x..H./....Nu*
0xf50000e0: 0d00 60af 0d00 8686 1000 754e 3763 352e *...`.....Nuc7.5*
0xf50000f0: 0000 0008 8827 4f4e 0060 e200 0060 b419 *....'.NO`....`...*
value = 0 = 0x0
```

The values marked in **red** in the dump before are some kind of magic numbers that you can find in the BAR3 area of a C331 board. This way you can verify the *c331PciOffset* needed to be zero in this case.

## CAN-PCI/200 Access Test Example

The following example is for the CAN-PC/200 board. It is also valid for all boards of the C200 family (consider the different PCI IDs for the other boards of that family). You could verify the *c200PciOffset* by first extracting the BAR2 value from the output of *pciHeaderShow()*, then calculate the resulting *cpuAddr* for that area to use the *d()* command to look into the board's memory. From the output below we find the *pciAddr* for BAR2 being 0xf5201000. With a *c200PciOffset* of zero in this case we get also 0xf5201000 for the *cpuAddr* here.

```
-> pciHeaderShow 0,17,0
vendor ID =                0x10b5
device ID =                0x9050
command register =         0x0003
status register =          0x0280
revision ID =              0x02
class code =               0x0c
sub class code =           0x09
programming interface =    0x00
cache line =               0x08
latency time =             0x00
header type =              0x00
BIST =                     0x00
base address 0 =           0xf5204000
base address 1 =           0x0000d801
base address 2 =           0xf5201000    <- BAR2 needed for access test
base address 3 =           0x00000000
base address 4 =           0x00000000
base address 5 =           0x00000000
cardBus CIS pointer =      0x00000000
sub system vendor ID =     0x12fe
sub system ID =            0x0004
expansion ROM base address = 0x00000000
interrupt line =           0x0b
interrupt pin =            0x01
min Grant =                0x00
max Latency =              0x00
value = 0 = 0x0
-> d 0xf5201000,0x20
NOTE: memory values are displayed in hexadecimal.
0xf5201000:  ff21 e00c 0000 0000 0000 ffff ffff ffff  *!.....*
0xf5201010:  ffff ffff ffff fbff 0900 0900 fffe 00ff  *.....*
0xf5201020:  ff21 e00c 0000 0000 0000 ffff ffff ffff  *!.....*
0xf5201030:  ffff ffff ffff fbff 0900 0900 fffe 00ff  *.....*
value = 0 = 0x0
```

The hex values 0x21 in the dump before show that you have found the CAN-PCI/200 (these values can only be seen after a power on reset). This way you can verify the *c200PciOffset* needed to be zero in this case.

#### 4.1.6.4.4 Manual Configuration of PCI Addresses

If you have the case of a host board / VxWorks BSP without a PCI PnP run you need to setup the PCI stuff by hand. To do that you enter in the *base* member of your driver parameter structure a value different from `0xFFFFFFFF`. Then the driver will write PCI addresses into the PCI device bridges of our **esd** CAN boards. See the `readme` files from the driver packages how much memory space is needed for that specific board.

It is your responsibility to administer available memory ranges. Check all PCI-PCI bridges for being transparent at the right addresses. Verify that the PCI command register enables PCI device's memory spaces. After that you can check the accessibility via the `d()` command as described before.

#### 4.1.6.4.5 VME Bus Access Issues (VME-CAN4)

In the *base* member of the driver parameter structure you specify the VME bus address you want to use. The driver uses the BSP specific `sysBusToLocalAdrs()` function to translate the configured VME address to the CPU address needed to access the VME-CAN4 board. Because the VME-CAN4 can be configured to appear either in the VME-A24 address space or in the VME-A32 address space you need to select the desired address space using the *vmeSpace* member of the configuration structure. The possible values defined in the provided `caninit.c` file for that variable refer to definitions from VxWorks' `vme.h`:

```
#define VME_A16      VME_AM_SUP_SHORT_IO      /* 0x2d */
#define VME_A24      VME_AM_STD_SUP_DATA      /* 0x3d */
#define VME_A32      VME_AM_EXT_SUP_DATA      /* 0x0d */
```

Be aware of the fact that after a reset the VME-CAN4 board only appears in the VME-A16 address space and waits for further configuration. When the driver starts it takes the desired VME addresses from the configuration structure and programs it into the VME-CAN4 hardware. To see how to do that refer to the VME-CAN4 hardware manual. Only after that step the VME-CAN4 responds in the configured address space and range.

#### VME-CAN4 Access Test Example

In this example we assume a configuration of `VME_A32 (0x0d)` for *vmeSpace* and `0x08000000` for *sramBase* in `CAN4_CAN_INFO` structure. After the driver started to verify that the board can be accessed by the CPU you may do the following on the target shell:

```
-> adrPointerBuf=0
New symbol "adrPointerBuf" added to kernel symbol table.
adrPointerBuf = 0x17611fd0: value = 0 = 0x0
-> sysBusToLocalAdrs(0xd,0x08000000,&adrPointerBuf)
value = 0 = 0x0
-> adrPointerBuf
adrPointerBuf = 0x17611fd0: value = 536870912 = 0x20000000
-> d 0x20000000,0x10
NOTE: memory values are displayed in hexadecimal.
0x20000000:  4341 4e5f 564d 455f 3034 5f56 5f30 2e43  *CAN_VME_04_V_0.C*
0x20000010:  0001 0000 0001 0000 1500 0000 0000 0000  *.....*
value = 0 = 0x0
```

This way you can manually verify that the driver can access the dual-ported RAM of the VME-CAN4. After translation of the `VME_A32` address `0x08000000` to a valid CPU address (here `0x20000000`) you should be able to dump the **red** marked string from the start of the dual-ported RAM area.



#### 4.1.6.5 Interrupt Connection Issues

The driver needs a correctly connected interrupt handler to work. There are three main categories that interrupt issues could belong to:

- The functions to connect a C interrupt handler differ depending on CPU architecture or the BSP of the host board. There is an assortment of functions possible that may only be provided on a part of architectures and BSPs. Some of them are ***intConnect()*** and ***pcilntConnect()***. Some BSPs provide neither of both routines. Also for some interrupts you may need to use ***intConnect()*** and for others on the same board ***pcilntConnect()***.
- To connect a C interrupt handler the VxWorks routines need the interrupt vector address as parameter. How this interrupt vector address is determined from the interrupt number depends on CPU architecture and may also depend on the BSP and other environmental conditions. Even the interrupt number itself may depend on the BSP. To determine the interrupt number the BSP's documentation should be the first source.
- The driver uses the configured IRQ number to enable the interrupt utilizing ***sysIntEnable()*** or ***sysIntEnablePIC()*** depending on the CPU architecture.

These categories lead to an unusable driver. Please check at first with your BSP's documentation if the right functions are used for the interrupt numbers in question. Also check if the IRQ number is correct for your BSP and board (the driver prints the used IRQ number in the start-up banner). In a second step you may try to find out if any interrupt number translation is needed.

##### 4.1.6.5.1 Interrupt Connection: Call of Wrong Function

Sometimes the failed interrupt connection has its cause by the driver using the wrong function to connect the interrupt. In some cases you will not be able to load the driver object because the BSP lacks some of the needed functions. This may for example happen if the driver is compiled to use the ***pcilntConnect()*** function but your BSP doesn't provide one. In this case check if there is another build variant of the driver available that uses the ***intConnect()*** call instead (see chapter 4.1.4.1 on how to identify build variants).

In any case you could inspect the ***<drvname>.sys*** file with the nm tool on the host to see which connect functions are utilized. The nm tool's full name changes depending on the target architecture (e. g. nmpppc, nmpentium or the like).

There are some releases of the driver that support the selection whether ***intConnect()*** or ***pcilntConnect()*** is used to connect the interrupt. The behaviour is controlled via the `flags` member of the parameter structure (see also chapter 4.1.3.1 for information). To decide which function needs to be used you should refer to the documentation of your BSP.



For a non-VxBus driver configured from the VxWorks Workbench you can change the `flags` by adapting the `ESD_CAN_<drvsig>_FLAGS` macro to your needs.

To alleviate the problems arising from the use of the wrong interrupt connect or disconnect routines recent driver versions export function pointers that can be used by the user to provide callback functions to the driver. These callback function will be used by the driver core to connect or disconnect the interrupt handler.

## User specific driver core IRQ connect function selection

Drivers that support this feature export the function pointers `<drvname>AttachIrqHandler` and `<drvname>DetachIrqHandler`. Addresses of user specific function should be assigned to these pointers. The driver core calls these functions via the function pointers to attach or detach the interrupt handler. If the user specified functions return the error code `ENOSYS` (from `errno.h`) then the driver core will execute as fallback its default connect and disconnect routines.

Please refer to the `caninit.c` / `caninit_<drvname>.c` file that belongs to your driver to see how these function pointers can be initialized. It can be done either manually in these files or via macros changeable via the Windriver Workbench.

## VME-CAN4 Connect Function Selection

The CAN4 driver calls connect and disconnect functions using a function pointer that the user has to initialise. The `caninit.c` / `caninit_ican4.c` file provide the means to change this initialisation. You have to select a connection mechanism via the variable `can4_irqConnectMode` and provide an entry for a connect function via the function pointer `can4_pflntConnect`. The macros to change these values from the WindRiver workbench are `ESD_CAN_ICAN4_IRQ_CON_MODE` and `ESD_CAN_ICAN4_IRQ_CON_FUNC`. For the disconnect functions the same mechanism with other variables is implemented. To make this more clear please refer to the source files mentioned in this paragraph.

### 4.1.6.5.2 Interrupt Connection: Wrong IRQ Number to Vector Translation

The vxWorks functions to connect an interrupt take unfortunately an interrupt vector address as parameter and not an interrupt number which would be more sound. This way the task to translate an interrupt number to the correct interrupt vector address is imposed on the driver. Most unfortunately the way to do this translation also involves a macro `INUM_TO_IVEC()` that is at least architecture specific.

In the following paragraphs it is described how the driver does the IRQ number to IRQ vector translation by default and how it may be tuned to fit to your BSP.

The default formula for the number to vector translation involves the value of `sysVectorIRQ0` which may be either a global variable of the BSP or a macro used at build time. The typical values of the macro used at build time are `0x20` for the **x86** architecture and `0x00` for the **PPC** architecture.

$$irqVector = INUM\_TO\_IVEC( irqNumber + sysVectorIRQ0 )$$

The `irqNumber` mentioned in the formula is the IRQ number provided via the parameter structure except for PCI based boards where the interrupt number is taken by default from the PCI config space. To compensate for a wrong value of `sysVectorIRQ0` or any peculiar differences of your BSP most drivers provide an additional IRQ offset variable that can be used to tune the translation to your needs as the formula below shows.

$$irqVector = INUM\_TO\_IVEC( irqNumber + irqOffset + sysVectorIRQ0 )$$

Every driver that supports this variable exports this as `<drvname>IrqOffset`. It should be adjusted to your needs in the `caninit.c` file before `<drvname>_install()` is called.



For a non-VxBus driver configured from the VxWorks Workbench you should enable the `INCLUDE_ESD_CAN_BSP_PNP_QUIRK` which will allow you to set the offset `<drvname>IrqOffset` using the macro `ESD_CAN_PCI_IRQ_OFFSET` in the configlet `caninit_<drvname>.c` located in the target tree.

Using only this approach there are still some translation issues left that can't yet be solved:

- Still the `INUM_TO_IVEC()` macro is compiled into the driver binary. If this macros is BSP specific the driver won't work.
- With a single *irqOffset* value the IRQ numbers can only be shifted. Some BSPs that support virtual IRQ numbers or need a IRQ number reordering (use `INT_NUM_GET()`) can't be supported.

To provide a solution more recent versions of the driver export for that reason a function pointer variable `<drvname>GetIrqVector`. If that variable doesn't contain a NULL pointer then the driver calls that function to translate the *irqNumber* to the *irqVector*. You have there the opportunity to implement any interrupt number to interrupt vector translation formula that you need for your environment. A sample implementation of that function that avoids the `INUM_TO_IVEC()` and `INT_NUM_GET()` issues looks like this (for a C200I board):

```
/*-----*/
/* Callback to return assigned IRQ vector */
/*-----*/
LOCAL VOIDFUNCPTR * c200iIrqVector(int irq)
{
    VOIDFUNCPTR *irqVec;

    irqVec = INUM_TO_IVEC(INT_NUM_GET(irq));

    printf("--> BSP specific IRQ vector is 0x%x\n", irqVec);

    return(irqVec);
}
```

Now setup the pointer `c200iGetIrqVector` with the address of `c200iIrqVector()` and the driver will call it to do the IRQ number to IRQ vector translation.

## VME-CAN4 Interrupt Number Restrictions and Offset

Please be aware that the IRQ number you provide to the driver via the `irq_num` member of the parameter structure needs to be a multiple of eight (8). This has its reason in a hardware limitation and is described in more detail in the hardware manual of the VME-CAN4.

If you need to provide an IRQ number offset you should set the variable `can4_irqNumOffset` accordingly. If you configure the driver from the VxWorks Workbench you should adapt the macro `ESD_CAN_ICAN4_IRQ_OFFSET` as needed which in turn will be used to setup `can4_irqNumOffset` in `caninit_ican4.c`.

### 4.1.6.6 VxBus Driver Prerequisites

The VxBus API provides the first time something that could be called a Hardware Abstraction Layer under VxWorks. Therefore the problems to find and claim the right resources for a driver should become less evident. But on the other hand it is now the duty of the BSP to setup and configure the VxBus libraries provided by WindRiver to establish the hardware abstraction. Here will follow some hints to check whether the BSP has done its job good enough to support our VxBus drivers for PCI interface boards.

The following things must be added to your kernel configuration to do these tests:

- INCLUDE\_VXBUS                      adds VxBus support itself
- INCLUDE\_VXBUS\_SHOW                adds VxBus show routines
- INCLUDE\_PCI\_BUS\_SHOW              adds PCI show routines based on the VxBus libraries

In the following paragraphs we will try to show you how to test if your BSP does not only have simplistic support for VxBus but at least includes the VxBus PCI hardware abstraction that our driver needs. The **vxBusShow()** routine will show you if the VxBus system is aware of the PCI device you plugged in. You may also use **vxvTopoShow()** to get a more condensed output that shows only the buses and devices found.

#### 4.1.6.6.1 Example of Working VxBus Implementation

Here follows the output of a working implementation with the important parts marked in **red** and some unimportant entries removed. It shows the conditions without any **esd** CAN driver included into the VxWorks image but with an **esd** CAN board plugged in:

```
-> vxBusShow
Registered Bus Types:
  MII_Bus @ 0x004c52ac
  PCI_Bus @ 0x004c5840
  PLB_Bus @ 0x004c58c0

Registered Device Drivers:
  pentiumPci at 0x004c4ae0 on bus PLB_Bus, funcs @ 0x004c4b68
  i8253TimerDev at 0x004c55e0 on bus PLB_Bus, funcs @ 0x004c5660
  ns16550 at 0x004c5480 on bus PLB_Bus, funcs @ 0x004c5548
  m6845Vga at 0x004c4c20 on bus PLB_Bus, funcs @ 0x004c4c88
  i8042Kbd at 0x004c4b80 on bus PLB_Bus, funcs @ 0x004c4c00
  genericPhy at 0x004c5300 on bus MII_Bus, funcs @ 0x004c5340
  miiBus at 0x004c5260 on bus PCI_Bus, funcs @ 0x004c52c8
  elPci at 0x004c4f20 on bus PCI_Bus, funcs @ 0x004c4fb8
  plbCtrlr at 0x004c5880 on bus PLB_Bus, funcs @ 0x004c5978

Busses and Devices Present:
  PLB_Bus @ 0x004fb4d8 with bridge @ 0x004c58e0
    Device Instances:
      ns16550 unit 0 on PLB_Bus @ 0x004fc498 with busInfo 0x00000000
      pentiumPci unit 0 on PLB_Bus @ 0x004fc898 with busInfo 0x004fb718
      i8042Kbd unit 0 on PLB_Bus @ 0x004fe598 with busInfo 0x00000000
      m6845Vga unit 0 on PLB_Bus @ 0x004fe798 with busInfo 0x00000000
      i8253TimerDev unit 0 on PLB_Bus @ 0x004fe898 with busInfo 0x00000000
    Orphan Devices:
      i8042Mse unit 0 on PLB_Bus @ 0x004fe698 with busInfo 0x00000000
      PCI_Bus @ 0x004fb718 with bridge @ 0x004fc898
        Device Instances:
          elPci unit 0 on PCI_Bus @ 0x004fd398 with busInfo 0x00000000
          miiBus unit 0 on PCI_Bus @ 0x004fec98 with busInfo 0x004fc218
        Orphan Devices:
          (null) unit 0 on PCI_Bus @ 0x004fd198 with busInfo 0x00000000
          (null) unit 0 on PCI_Bus @ 0x004fd298 with busInfo 0x00000000
```

```

        (null) unit 0 on PCI_Bus @ 0x004fe498 with busInfo 0x00000000
MII_Bus @ 0x004fc218 with bridge @ 0x004fec98
Device Instances:
    genericPhy unit 0 on MII_Bus @ 0x004fed98 with busInfo 0x00000000
Orphan Devices:

```

```
value = 1 = 0x1
```

This output tells us in the “Busses and Devices Present” section that the `pentiumPci` device instance is at `0x004fc898` (this is the PCI host controller). It provides access to the `PCI_Bus` instance at `0x004fb718` which has a backlink to the host controller instance at `0x004fc898`.

The `PCI_Bus` itself has a list of “Device Instances” for which some device driver claims responsibility (i. e. `elPci`). Also the `PCI_Bus` has a list of “Orphan Devices”. This list has to grow after you plugged in the **esd** CAN board. You may inspect the device instance for being an **esd** CAN board (check PCI IDs) by using ***pciDevShow()***. For the current example selecting the orphaned device at `0x004fd298` you will get:

```

-> pciDevShow 0x004fd298
pDev @ 0x004fd298    [0,17,0]
    devID    = 0x9050
    vendID    = 0x10b5
value = 17 = 0x11

```

At least from these PCI IDs it may be an **esd** CAN board. To be sure you can interrogate the `PCI_Bus` instance at `0x004fc898` with ***vxbPciHeaderShow()*** like this:

```

-> vxbPciHeaderShow 0x004fc898,0,17,0
vendor ID = 0x10b5
device ID = 0x9050
command register = 0x0003
status register = 0x0280
revision ID = 0x02
class code = 0x0c
sub class code = 0x09
programming interface = 0x00
cache line = 0x08
latency time = 0x00
header type = 0x00
BIST = 0x00
base address 0 = 0xf5204000
base address 1 = 0x0000d801
base address 2 = 0xf5201000
base address 3 = 0x00000000
base address 4 = 0x00000000
base address 5 = 0x00000000
cardBus CIS pointer = 0x00000000
sub system vendor ID = 0x12fe
sub system ID = 0x0004
expansion ROM base address = 0x00000000
interrupt line = 0x0b
interrupt pin = 0x01
min Grant = 0x00
max Latency = 0x00
value = 0 = 0x0

```

Judging from this output you know now that this device is a CAN-PCI/200 and that the `VxBus` implementation is aware of the hardware so far. Even the interrupt assignment could now be displayed with ***plbIntrShow()*** for the same orphaned device at `0x004fd298`:

```
-> plbIntrShow 0x004fd298,1
(null)_0:
    numVectors      = 1 (defined in hwconf.c for device)
    intVecList      @ 0x00000158
    intLvlList      @ 0x0000000b
    intCtlrList     @ 0x00000000
    pIntCtlrTable   @ 0x00000000
value = 0 = 0x0
```

To display all this information, we used native VxBus routines to be sure that we really talk to the VxBus implementation. If you have seen all this information on your board the VxBus driver should work. If you only need to see some information about the PCI configuration or want to test if the board itself is accessible you may use the functions described in chapter 4.1.6.4.2.

### 4.1.6.6.2 Example of not Working VxBus Implementation

In the following section is some output from an insufficient VxBus implementation shown that won't be able to support our VxBus CAN driver. This is the condition with the CAN board plugged in but without the esd VxBus CAN driver compiled into the VxWorks kernel image.

```
-> vxBusShow
Registered Bus Types:
    MII_Bus @ 0x00311a04
    PCI_Bus @ 0x0031162c
    PLB_Bus @ 0x00311648

Registered Device Drivers:
    mottsec at 0x003118e8 on bus PLB_Bus, funcs @ 0x003117d0
    mdio at 0x00311be0 on bus PLB_Bus, funcs @ 0x00311bac
    mv88E1x11Phy at 0x00311b00 on bus MII_Bus, funcs @ 0x00311ad0
    bcm54xxPhy at 0x00311a70 on bus MII_Bus, funcs @ 0x00311a40
    genericPhy at 0x00311b6c on bus MII_Bus, funcs @ 0x00311b60
    miiBus at 0x003119bc on bus PCI_Bus, funcs @ 0x00311960
    miiBus at 0x0031197c on bus PLB_Bus, funcs @ 0x00311960
    plbCtlr at 0x00311670 on bus PLB_Bus, funcs @ 0x00311664

Busses and Devices Present:
    PLB_Bus @ 0x0035ae78 with bridge @ 0x003116b0
        Device Instances:
            mottsec unit 0 on PLB_Bus @ 0x0035be38 with busInfo 0x00000000
            mottsec unit 1 on PLB_Bus @ 0x0035bf38 with busInfo 0x00000000
            miiBus unit 0 on PLB_Bus @ 0x0035c238 with busInfo 0x0035b1b8
            miiBus unit 1 on PLB_Bus @ 0x0035c538 with busInfo 0x0035b1f8
        Orphan Devices:
    MII_Bus @ 0x0035b1b8 with bridge @ 0x0035c238
        Device Instances:
            genericPhy unit 0 on MII_Bus @ 0x0035c338 with busInfo 0x00000000
        Orphan Devices:
    MII_Bus @ 0x0035b1f8 with bridge @ 0x0035c538
        Device Instances:
            genericPhy unit 1 on MII_Bus @ 0x0035c638 with busInfo 0x00000000
        Orphan Devices:

value = 1 = 0x1
```

At least the `PCI_Bus` shows up in the “Registered Bus Types” section. But in the “Buses and Devices Present” section there is no instance of the `PCI_Bus` present. Therefore the VxBus layer doesn't know anything about the PCI devices present and cannot provide any information to our VxBus PCI driver. Instead we miss a driver for a PCI host bridge in the “Registered Device Drivers” section and based on that driver an instance on the `PLB_Bus` that could serve as root bridge device for a `PCI_Bus` instance.

## 4.2 QNX

The installation on QNX® requires administrator rights on the host.

### 4.2.1 QNX 6 and QNX 7

#### 4.2.1.1 Driver Package Content

The software drivers for the QNX package are shipped on CD-ROM. The CD contains the following files:

<i>File</i>	<i>Description</i>
Readme	Current notes and installation information
CHANGELOG	Change list / driver history
devcan-xxx-yyy	CAN resource manager for a CAN device family. The character combination devcan-xxx-yyy follows the driver naming convention described in Table 2: Overview of the CAN Interface Families. For a CAN-PCI/331 it would be 'devcan-pci331-i20'.
upd-isa331 upd-pci331 upd-pci360	Programs for firmware update (shipping on request)
pciclass-pci200 pciclass-pci266 pciclass-pci331	Programs for setting the PCI-Class
libntcan.so.x	dynamic CAN-API library (x... library version)
ntcan.h	header of CAN-API
cantest.c	source code of example program 'cantest' (see CAN-API manual part 1 /1/)
cantest	binary file of example program 'cantest'

### 4.2.1.2 Sequence of Installation Under QNX6 and QNX7

#### 4.2.1.2.1 Installation of ISA-Boards

First install the hardware of the module. When installing ISA boards, please make sure to set the board address correctly. The address corresponding to the standard configuration (e.g. 0x1E8 for CAN-ISA/200) should be set before shipping already.

#### Calling the Resource Manager

Log in as 'root'.

Start the resource manager (e.g. for CAN-ISA/200):

```
devcan-isa200-sja1000 -v
```

After this call an output similar to the following is to appear on the screen:

```
devcan-isa200-sja1000: version 3.9.3 build 16:52:53 Mar 19 2013
(0)CAN_ISA200: Clock=0x00000000 Baud=0x7fffffff Mode=0x00000000 Nodes=can0
(0)CAN_ISA200: Hardware-version 1.0.0
```

Depending on the module the resource manager is called by means of different commands. The code `isa200-sja1000` in the character combination '`devcan-isa200-sja1000`' shown in the example above, has to be substituted for other modules as described in 4.2.1.1.



**Command Line of the Resource Manager for ISA Boards:**

With the command

```
devcan-isa200-sja1000 -h
```

or

```
use devcan-isa200-sja1000
```

the input syntax of all available parameters (here CAN-ISA/200) is listed:

```
devcan-isa200-sja1000 [Options]
```

The resource manager can be configured by means of the parameters explained below.

<i>Parameter</i>	<i>Function</i>
<code>-b <i>baudrate</i></code>	Initial baud rate (default value: module is 'OFF Bus')
<code>-m <i>mode</i></code>	Mode flags (default value = 0x00000000) 0x00000040: Listen-Only mode only 0x00000080: Go OFF Bus after last handle is closed 0x20000000: Do not use firmware fast mode
<code>-n <i>net</i></code>	Base net of card. net = 0...255 (default value = 0 for first card)
<code>-v[<i>v...</i>]</code>	Verbose level
<code>-p <i>prio</i></code>	Back-end handling priority (default value = 19)
<code>-P <i>Prio</i></code>	Prio of IRQ handling thread (default: No IRQ threads)
<code>-R <i>rxbuffer</i></code>	Rx-buffer of CAN node (default: 128)
<code>-e <i>errorinfo</i></code>	Extended error info (default = 1): 0=Off 1=On
<code>-C <i>frequency</i></code>	Override CAN controller clock in [Hz]
<code>-t <i>maxthreadpool</i></code>	Maximal size of threadpool (1...64) (default = 4* CAN nodes)
<code>-h</code>	Showing a help text
<code>-c <i>card_options</i></code>	Setting of the I/O-address and the interrupts of a CAN module: io = <i>address</i> (I/O-address) irq = <i>interrupt</i> (An unused interrupt of the system has to be determined and set with this parameter.)

**Example** for the installation of two CAN-ISA/200 boards in one system:

```
devcan-isa200-sja1000 -n0 -c io=0x1e8,irq=5 -n1 -c io=0x1e0,irq=7
```

Now you can test the communication on the CAN bus by means of the example program `cantest`.

### 4.2.1.2.2 Installation of PCI-Boards

First install the module hardware.

Log in as 'root'.

Start the resource manager (example CAN-PCI/331):

```
devcan-pci331-i20 -v
```

Now an output similar to the one below has to appear on the screen:

```
devcan-pci331-i20: version 3.9.3 build 16:52:53 Mar 19 2013
(0)CAN_PCI331: Clock=0x00000000 Baud=0x7fffffff Mode=0x00000000 Nodes=can0
(0)CAN_PCI331: Hardware-version 1.0.0
```

Depending on the module the resource manager might be called by means of different commands. The code `pci331-i20` in the character combination 'devcan-pci331-i20', as shown in the example above, has to be substituted for other modules as described in 4.2.1.1.

**Command Line of the Resource Manager for PCI Boards:**

With the command

```
devcan-pci331-i20 -h
```

or

```
use devcan-pci331-i20
```

the input syntax of all available parameters (here CAN-PCI/331) is listed:

```
devcan-pci331-i20 [Options]
```

The resource manager can be configured by means of the parameters described below.

<i>Parameter</i>	<i>Function</i>
<code>-b baudrate</code>	Initial baud rate (default value: module is 'OFF Bus')
<code>-m mode</code>	Mode flags (default value = 0x00000000) 0x00000040: Listen-Only mode only 0x00000080: Go OFF Bus after last handle is closed 0x20000000: Do not use firmware fast mode
<code>-n net</code>	Base net of card. net = 0...255 (default value = 0 for first card)
<code>-v[v...]</code>	Verbose level
<code>-p prio</code>	Back-end handling priority (default value = 19)
<code>-P Prio</code>	Prio of IRQ handling thread (default: No IRQ threads)
<code>-R rxbuffer</code>	Rx-buffer of CAN node (default: 128)
<code>-e errorinfo</code>	Extended error info (default = 1): 0=Off 1=On
<code>-C frequency</code>	Override CAN controller clock in [Hz]
<code>-t maxthreadpool</code>	Maximal size of threadpool (1...64) (default = 4* CAN nodes)
<code>-h</code>	Showing a help text
<code>-c card_options</code>	PCI index and/or <b>esd</b> device-id of card (default: all CAN cards will be attached)  <div> <div>Pci</div> <div>= index</div> <div>(= "PCI Index" as indicated by the QNX Shell command "pci -v")</div> </div> <div> <div>sdid</div> <div>= id</div> <div>(= "Subsystem ID" as indicated by the QNX Shell command "pci -v")</div> </div> <div> <div>irq</div> <div>= interrupt</div> <div>(= Overload the PCI P&amp;P interrupt)</div> </div> <div> <div>nomsi</div> <div></div> <div>(= Forbit the use of MSI interrupts)</div> </div>

**Example** for the installation of two CAN-PCI/331 boards in one system:

```
devcan-pci331-i20 -n0 -c pci=0 -n2 -c pci=1
```

Now you can test the communication on the CAN bus by means of the example program `cantest`.

## 4.2.2 QNX4



### Attention!

As the date of the last QNX 4.x release was more than 20 years ago, technical support and maintenance by **esd** for this QNX version is also terminated but the latest version of the device driver files are kept available.

### 4.2.2.1 Files of the QNX4 Packages

The software drivers for the QNX package are shipped on CD-ROM. The CD contains the following files:

File	Description										
readme	current notes and installation information	The character combination 'c331' signifies files of the CAN-PCI/331 module. For other modules the characters have to be changed as follows: <table><tr><th>CAN Module</th><th>File Name</th></tr><tr><td>CAN-ISA/200</td><td rowspan="2">c200i</td></tr><tr><td>CAN-PC104/200 (SJA1000 version)</td></tr><tr><td>CAN-ISA/331 CAN-PC104/331</td><td>c331i</td></tr><tr><td>CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331</td><td>c331</td></tr></table>	CAN Module	File Name	CAN-ISA/200	c200i	CAN-PC104/200 (SJA1000 version)	CAN-ISA/331 CAN-PC104/331	c331i	CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331
CAN Module	File Name										
CAN-ISA/200	c200i										
CAN-PC104/200 (SJA1000 version)											
CAN-ISA/331 CAN-PC104/331	c331i										
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331										
c331	CAN resource manager for CAN-PCI/331										
updc331	program for firmware update										
libntcan3s.lib	CAN-API library (stack parameter)										
libntcan3r.lib	CAN-API library (register parameter)										
ntcan.h	header for CAN-API										
cantest.c	source code of example program 'cantest' (see CAN-API manual part 1 /1/)										
cantest	binary file of example program 'cantest'										

### 4.2.2.2 Sequence of Installation Under QNX4

#### 4.2.2.2.1 Installation of ISA-Boards

First install the hardware of the module. When installing ISA boards, please make sure to set the board address correctly. The address corresponding to the standard configuration (e.g. 0x1E8 for CAN-ISA/200) should be set before shipping already.

#### Calling the Resource Manager

Log in as 'root'.

Start the resource manager (e.g. for QNX4, CAN-ISA/200): ``c200i &``

After this call an output similar to the following is to appear on the screen:

```
C200i[0x1e8]: Using I/O-Base 0x1E8
C200i[0x1e8]: Using Interrupt 5
C200i[0x1e8]: "CAN_ISA200" with 1 Nets identified
C200i[0x1e8]: Hardware-Version=1.0.00
C200i[0x1e8]: Firmware-Version=0.0.00
C200i[0x1e8]: Driver-Version =1.0.00
C200i[0x1e8]: Net 0 successfully created
```

Depending on the module the resource manager is called by means of different commands. The character combination 'c200i' for QNX4 shown in the example above, has to be substituted for other modules as follows:

Input syntax when calling the resource manager:

<i>CAN Module</i>	<i>Entry Syntax QNX4</i>
CAN-ISA/200	c200i
CAN-PC104/200 (SJA1000 version)	
CAN-ISA/331 CAN-PC104/331	c331i

### Command Line of the Resource Manager for ISA Boards

<i>Parameter</i>	<i>Function</i>
-h	Showing a help text
-n <i>net</i>	Assigning the logical network number. <i>net</i> = 0...255 default: <i>net</i> = 0
-p <i>port</i>	Selecting the ISA board in the system by means of the port address set via the hardware (see also hardware manual of the module).
-i <i>irq</i>	Setting the interrupt to be used by the board. You have to determine an available interrupt which has to be set here. If this parameter is not specified, interrupt 5 will be used.

**Example** for the installation of two CAN-ISA/200 boards in one system:

<i>Call</i>	<i>Function</i>
c200i -p0x1e8 -i5 -n0 &	Interrupt 5 and the CAN network number 0 are assigned to the CAN-ISA/200 module with the address 0x1E8 .
c200i -p0x1e0 -i7 -n1 &	Interrupt 7 and the CAN network number 1 are assigned to the CAN-ISA/200 module with the address 0x1E0 .

Now you can test the communication on the CAN bus by means of the example program `cantest`.

#### 4.2.2.2.2 Installation of PCI-Boards

First install the module hardware.

Log in as 'root'.

Start the resource manager (example CAN-PCI/331): `'c331 &'`

Now an output similar to the one below has to appear on the screen:

```
C331[0]: Using Interrupt 12
C331[0]: "CAN_PCI331" with 2 Nets identified
C331[0]: Hardware-Version=1.1.00
C331[0]: Firmware-Version=0.c.00
C331[0]: Driver-Version  =1.0.00
C331[0]: Net 0: Successfully created
C331[0]: Net 1: Successfully created
```

Depending on the module the resource manager might be called by means of different commands. The character combination 'c331', as shown in the example above, has to be substituted for other modules as shown below:

Input syntax when calling the resource manager on PCI boards:

<i>CAN Module</i>	<i>Entry Syntax QNX4</i>
CAN-PCI/200 CPCI-CAN/200 CAN-PCI/266	–
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	C331

-... has not been implemented yet

## Command Line of the Resource Manager for PCI Boards

The resource manager can be configured by means of the parameters described below.

<i>Parameter</i>	<i>Function</i>
-h	Showing a help text.
-n <i>net</i>	Assigning the logical network number.      default: net = 0 <i>net</i> = 0...255
-p <i>index</i>	Selecting the <b>esd</b> CAN-PCI boards in the system. The boards are numbered starting with '0'. The assignment of numbers and boards is determined by the plug-and-play controller. In case of doubt, the assignment should be checked by means of a test.

**Example** for the installation of two CAN-PCI/331 boards in one system:

<i>Call</i>	<i>Function</i>
c331 -p0 -n0 &	The CAN network numbers 0 and 1 are assigned to the CAN-PCI/331 module with the PCI number 0.
c331 -p1 -n2 &	The CAN network numbers 2 and 3 are assigned to the CAN-PCI/331 module with the PCI number 1.

Now you can test the communication on the CAN bus by means of the example program `cantest`.

## 4.3 IntervalZero RTX® and RTX64®

This chapter covers the necessary steps to install, configure and start the device drivers for **esd** CAN boards supporting the real time operating system *IntervalZero RTX and RTX64®* of *Interval Zero, Inc.*

RTX and RTX64 are no stand alone real-time operating systems but a kernel mode add-on that extends Microsoft Windows (32 or 64 bit) with high-speed and deterministic real-time capabilities.

As long as the information in this chapter cover the 32 bit version as well as the 64 bit version it is just referred to RTX. If the text refers explicitly to the 64 bit version the text will refer to RTX64.



### Attention!

As the IntervalZero support and maintenance for all RTX (32-bit) versions is expired, technical support and maintenance by **esd** is also terminated but the latest version of the device driver files are kept available.



### Note:

All **esd** CAN device drivers require an interrupt. RTX does not support sharing IRQ lines with Windows® devices, so the interrupt line used by the driver must be available for exclusive use by RTX but can be shared between RTX devices.

Finding an exclusive IRQ often requires physically moving hardware in the system or disabling other Windows® devices.

For the PCIe bus the most trouble-free solution is using the **CAN-PCIe/402** because this CAN board supports MSI which means interrupts are never shared and the problems described above do not arise.

A device driver package for RTX contains the following files where **<drvname>** is the device family specific driver name following driver naming convention I (see chapter 1.4).

<b>Filename</b>	<b>Description</b>
<b>bin/rtss/&lt;drvname&gt;.rtss</b>	The hardware specific CAN device driver.
<b>bin/rtss/ntcan.rtss</b>	The NTCAN library (only RTX)
<b>bin/rtss/ntcan.rtdll</b>	The NTCAN library (only RTX64)
<b>bin/rtss/cantest.rtss</b>	The <i>cantest</i> application
<b>doc/</b>	Folder with API documentation, release notes, etc.
<b>include/ntcan.h</b>	NTCAN header to compile the NTCAN based application.
<b>inf/</b>	Folder with digitally signed installation files (only RTX64).
<b>lib/vc/ntcan_rtss.lib</b>	Lib file for Visual Studio to link the NTCAN based application
<b>samples/cantest.c</b>	Source code of the <i>cantest</i> application

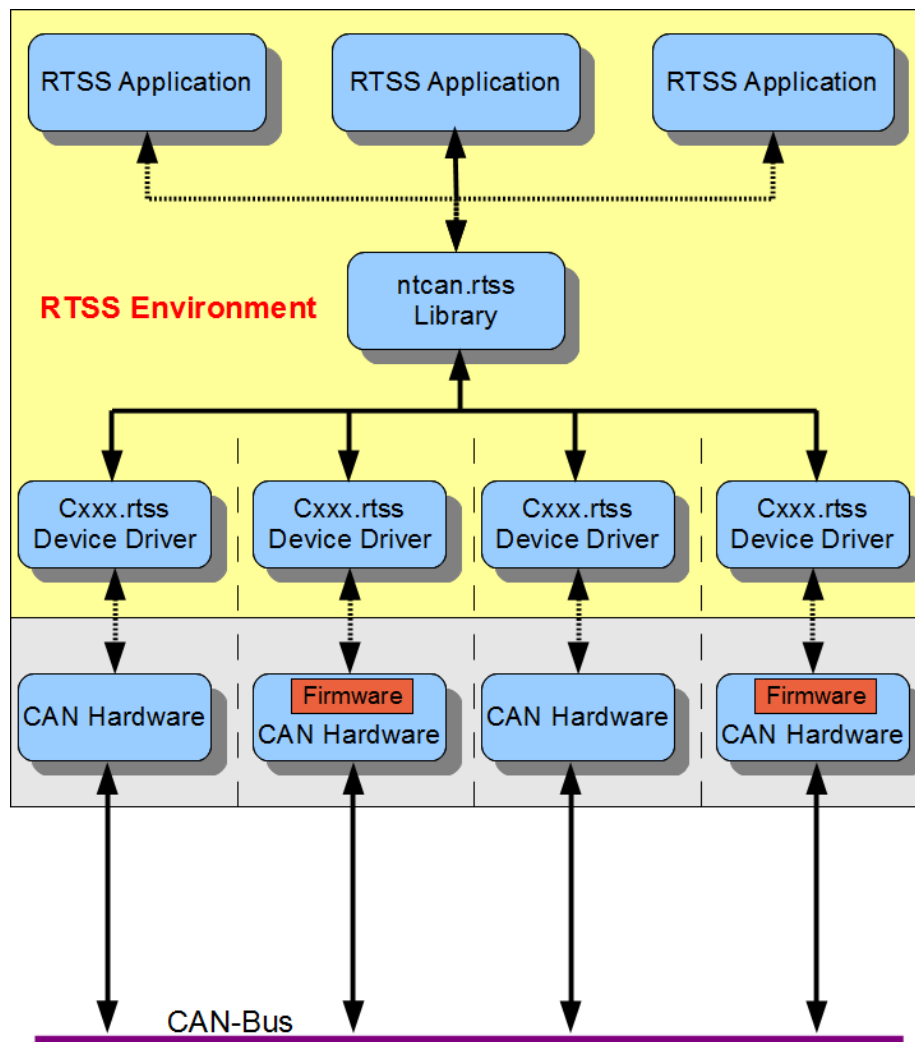
**Table 14:** Files of RTX/RTX64 CAN driver package



### 4.3.1 Driver Integration

#### 4.3.1.1 RTX

The support for an **esd** CAN interface consists of a device driver (Cxxx.rtss) and the NTCAN library (ntcan.rtss) as shown in the picture below. A hardware specific device driver (which usually covers a complete CAN device family) has to be loaded together with the CAN hardware independent NTCAN library into the RTSS environment. Device driver and NTCAN library are implemented as RTSS DLLs which means that they are individual RTSS processes exporting entries to any RTSS application, share a common address space with other RTSS processes and accurately mirror the automatic resolution of references to exported functions like implicitly linked Windows DLLs.



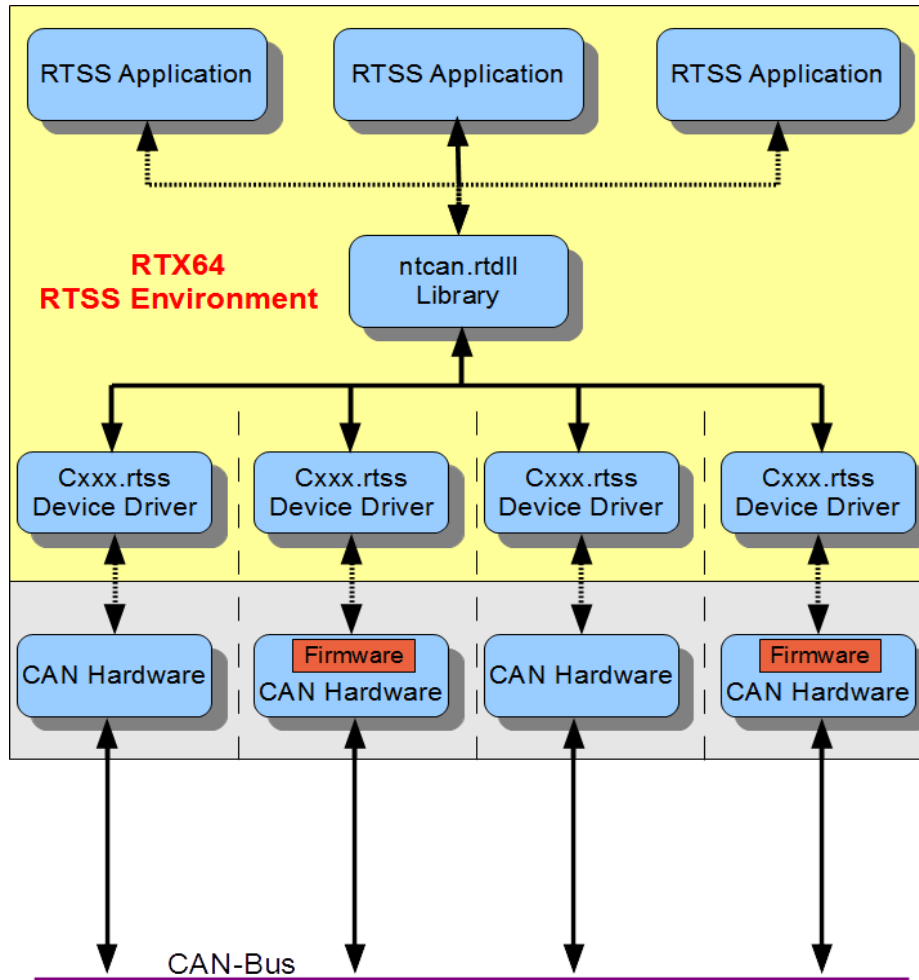
**Figure 3:** RTX Driver Architecture

This approach allows using different CAN drivers in the RTSS environment with the identical API and allows several RTSS processes using the same CAN hardware as well as one RTSS process using different CAN hardware.

The entries exported by the CAN device driver are only referenced by the NTCAN RTSS library. A CAN based RTSS application will always reference to the NTCAN API exported by the NTCAN RTSS DLL described in /1/. The NTCAN RTSS DLL may also be loaded without a driver to create RTSS applications which refer to the NTCAN API even if the CAN hardware and its device driver is not present.

#### 4.3.1.2 RTX64

The support for an **esd** CAN interface consists of a device driver (Cxxx.rtss) and the NTCAN library (ntcan.rtdll) as shown in the picture below. A hardware specific device driver (which usually covers a complete CAN device family) implemented as an RTSS process has to be loaded and the application has to link explicitly or implicitly to the NTCAN library implemented as an RTDLL.



**Figure 4:** RTX64 Driver Architecture

This approach allows using different CAN driver in the RTSS environment with the identical API and allows several RTSS processes using the same CAN hardware as well as one RTSS process using different CAN hardware.

The entries exported by the CAN device driver are only referenced by the NTCAN RTSS library. A CAN based RTSS application will always reference to the NTCAN API exported by the NTCAN RTSS DLL described in /1/.



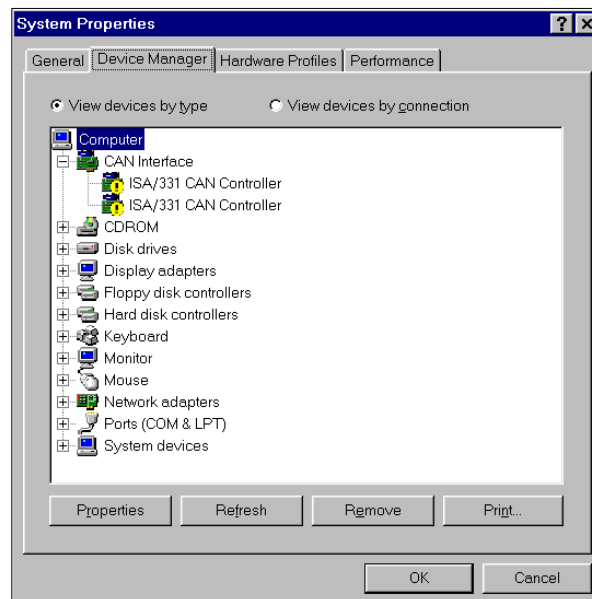
For implicitly linked RTDLLs or explicitly linking RTDLLs without specifying the full path name in LoadLibrary, the RTDLL files must be located in the same directory where the RTSS executable file exists.

### 4.3.2 Driver Installation

In order to support an **esd** CAN interface in the RTSS environment it is necessary to convert it into a RTX managed device as described below (for further details refer to the *RTX Runtime Documentation*).

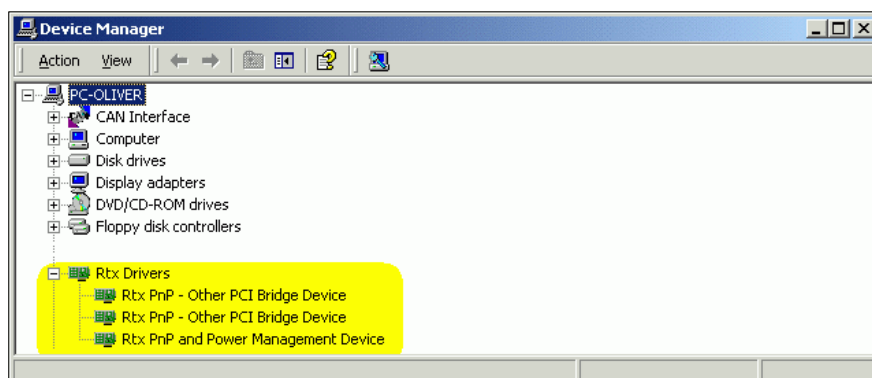
#### 4.3.2.1 RTX

Before the RTSS device driver can control the device the CAN interface has to be assigned to RTX. This is performed using the **RTX Properties** control panel applet via the **Plug and Play** Tab. In the device tree the CAN interface has to be selected and assigned to the RTX driver according to the following figure:



If a Windows driver is already installed for this CAN interface the device will be listed under the device class *CAN-Interface*. If no Windows driver has been installed, it will be listed depending on the CAN interface and Windows version as *Other PCI Bridge Device*, *Network Interface* or *CAN Interface*.

After this logical assignment the CAN interface must be removed from the Windows device management by selecting Remove in the device manager context menu followed by a scan for new hardware or a reboot of the system. On success the CAN interface should be listed in the RTX Drivers class as shown below:

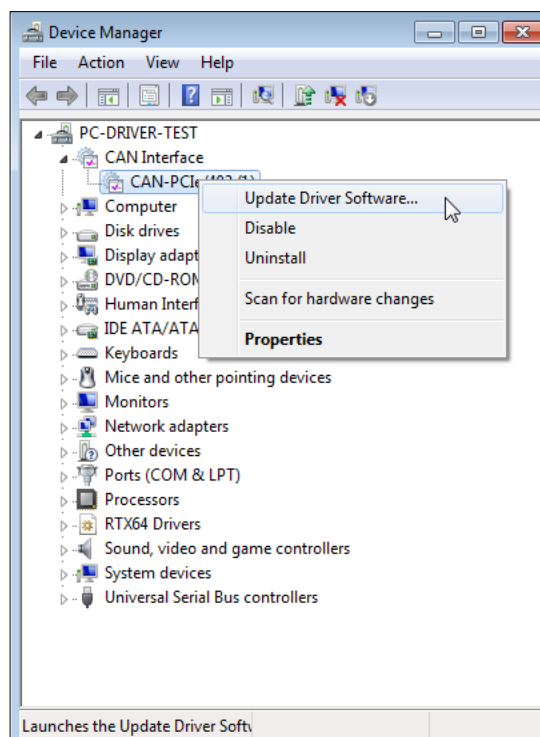


### 4.3.2.2 RTX64

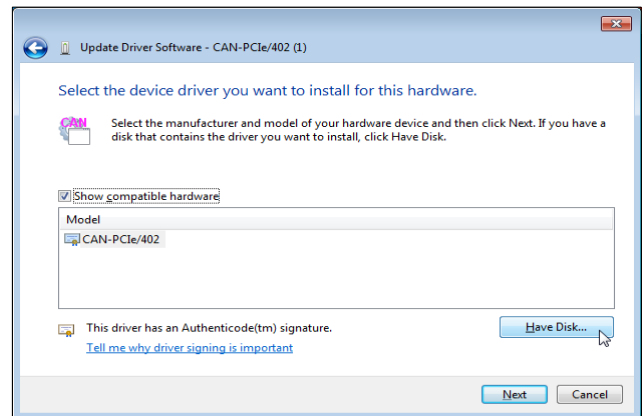
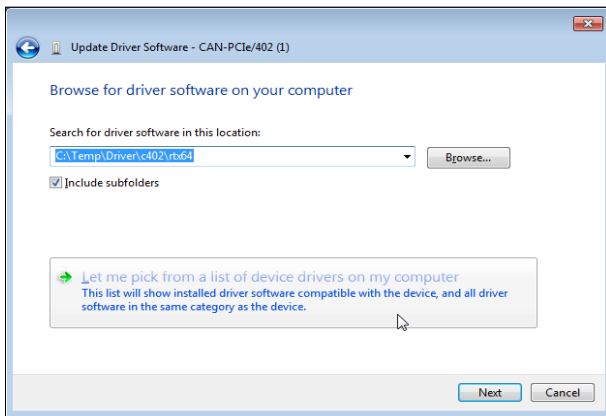
The 64-bit versions of Windows only accept digitally signed drivers. For this reason the RTX64 device driver package comes with an INF file and a related CAT file which will allow to install the RTX64PNP driver for the CAN interface. The steps to this are similar to the installation process of a standard Windows device driver described in chapter 2 for the various (RTX64 supported) versions of Windows.

For a general detailed instructions how to convert a device to RTX64 please refer to the chapter *Converting a PCI/PCIe Device to RTX64* in the RTX64 SDK Help.

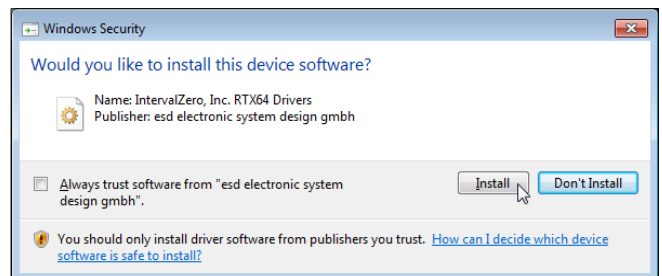
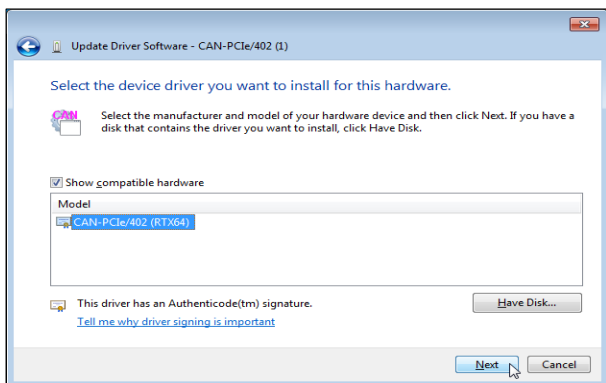
You have to use the *Windows Device Manager* to assign the RTX64PNP driver to the CAN interface. In the *Device Manager* windows there will be a device under *Other Devices* with a yellow exclamation point next to the icon (if there is no device driver installed yet. The text next to the device will depend on the CAN module attached) or the device is listed under *CAN Interfaces* if a Windows driver is already installed (see the picture below).



Follow the installation instructions and click the **Have Disk...** button to browse for the signed copy of the `esdrtx64.inf` file.



Proceed with the installation and Windows will present you a security dialogue box. The RTX64 device driver which are loaded in the RTSS environment are not digitally signed but the INF file which assigns the CAN interface to the IntervalZero RTX64PNP driver has to. Please refer to chapter 2.8 for more details about *Digital Signatures*.

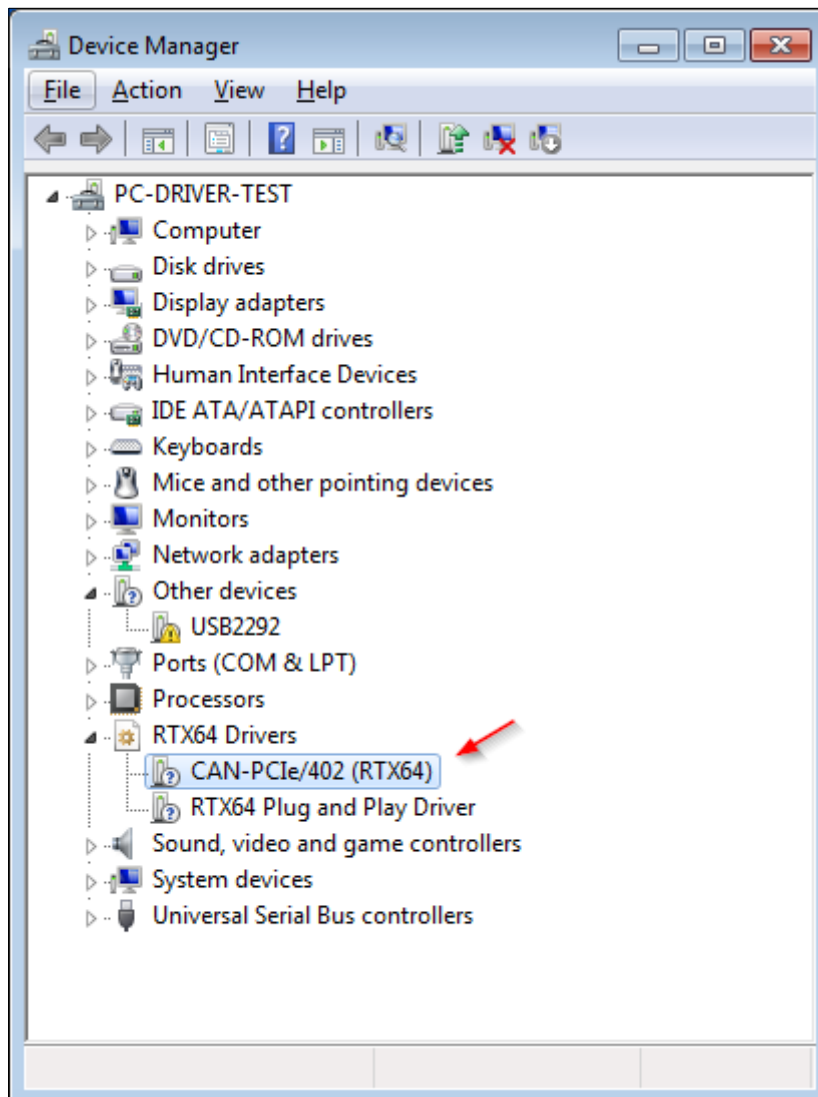


### Note:

If you activate the check box "Always trust software from *esd electronic system design gmbh*" you will not have to confirm this dialogue in the future during the installation of another digitally signed driver for an **esd** device.

Press the **Install** button to continue.

If device conversion to RTX64 completes successfully the CAN interface has to appear under *RTX64 Drivers* in the Device manager..



If a Windows driver is already installed for this device it is possible that Windows prevents migration to RTX/RTX64 with an indication that “The Best Driver is already Installed” for this device. Refer to chapter 2.4.4 to overcome this situation.

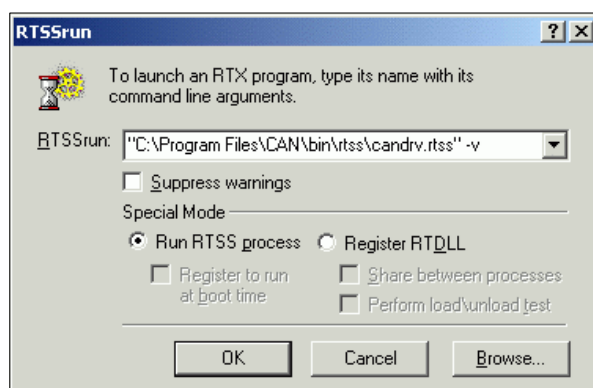
### 4.3.3 Driver Start

The driver is loaded into the RTSS environment and started with the console or the GUI version of the RTX tool *RTSSrun*. Type in a Windows console the following command:

```
rtssrun driver.rtss [Arguments]
```

where *driver.rtss* has to be replaced with the name of the driver (e.g. *c405.rtss*) and *Arguments* contain the device driver configuration parameter described in the next chapter.

In the GUI-version this may be performed like this:



**Figure 5:** Starting the driver

If an error occurs at start the numeric error code is displayed in the RTX console and the RTSS-process is terminated. In this case the driver can be started with the option '-v' followed by a numerical hexadecimal value to increase the verbosity of the debug trace messages reporting initialization errors.

#### For RTX (not RTX 64):

In a second step the NTCAN library has to be loaded into the RTSS environment and started in the same manner. Type in a Windows console the following command:

```
rtssrun ntcana.rtss [Arguments]
```

If an error occurs at start the numeric error code is displayed in the RTX console and the RTSS-process is terminated.

### 4.3.4 Driver Configuration

The RTX device drivers are configured with parameters to the *RTSSrun* command.

#### 4.3.4.1 Command Line Parameter

The RTX device drivers are configured via options listed in the table below which are passed as arguments to the driver start command described in the previous chapter:

Option	Argument	Description
-V	-	After the start the driver displays driver version information in the RTX console and terminates afterwards.
-v	<i>mask</i>	Display additional (debug) messages during startup in the RTX console. The mask parameter may be given as a decimal value or as a hexadecimal value prepended by a '0x'
-h	-	Display a list of all possible command line options and terminate afterwards.
-p	<i>prio</i>	This option configures the priority of the IST thread that processes the CAN messages. The default value for <i>prio</i> is 127.
-n	<i>net number</i>	This option configures the logical base net number which is assigned to the first physical port of the CAN interface. The logical net numbers are incremented by one for each additional physical port. The default value for <i>net number</i> is 0.
-k	-	Force clean unload of a resident driver (after closing all applications).

**Table 15:** Command Line Parameter of RTX Driver

#### Example:

The command

```
rtssrun c405.rtss -v0xFF -n2 -p118
```

- Starts the CAN-PCI/405 device driver with additional (debug) output
- Assigns the logical base-net number 2
- Configures an IST (interrupt service thread) priority of 118.

If a driver is started without any command line options the logical net number 0 is assigned to the first physical CAN port, all IST threads get a priority of 127 and the messages shown in the RTX console are reduced to a minimum.

#### 4.3.4.2 SMP Support

If RTX is assigned more than one physical processor on a SMP system the IRQ and the IST of the driver are handled on the processor the process runs on. You can influence this with the *RTSSrun* utility to optimize your overall system performance by using the '/a' parameter to configure the affinity and/or using the '/p' parameter to define an ideal processor (see RTX runtime help).

#### Example:

The command starts the CAN-PCI/405 device driver as in the example above but forces IRQ/IST execution to the processor with the system-wide processor number 2.

```
rtssrun /p 2 c405.rtss -v0xFF -n2 -p118
```



### 4.3.5 Miscellaneous

This chapter covers several topics using the NTCAN architecture on RTX.

#### 4.3.5.1 Application Development

To develop your own NTCAN based applications with Microsoft® Visual Studio the driver package contains the header `ntcan.h` and the library file `ntcan_rtss.lib`. To compile for RTX you have to define `UNDER_RTSS` before the header is included.

You can also start developing your application in the Win32 non-real-time environment before you change into the real-time RTSS environment to use the more comfortable possibilities to debug your application and take advantage of the **esd** CAN tools which just support the Windows environment. In this case you must install the Windows device driver and the CAN SDK as described in chapter 2. Use the header `ntcan.h` which comes with your RTX driver package (to prevent implementing features which are already available on Windows but not yet available on RTX) but do not define `UNDER_RTSS` before you include the header.

**Note:**

The NTCAN API on Windows and RTX is identical with the exception that overlapped operations are not supported in the RTSS environment.

#### 4.3.5.2 Example Application

The driver package comes with the RTSS application `cantest.rtss` which can be loaded into the RTSS environment in the same way as described for the NTCAN library. With the help of this program you can do basic functionality checks of the CAN interface. The program and its parameters are described in more detail in */1/*.

## 4.4 TenAsys® INtime®

This chapter covers the necessary steps to install, configure and start the device drivers for **esd** CAN boards supporting the real time operating system *INtime®* of *TenAsys®*

INtime® is a hard real-time, event-driven OS for the x86 architecture which can operate stand-alone (*INtime dustributed RTOS*) or side-by-side with Windows (*INtime for Windows*). The RT kernel supports Real-Time Applications (RTAs) and Real-Time Shared Libraries (RSLs).

A device driver package for INtime® contains the following files where **<drvname>** is the device family specific driver name following driver naming convention I (see chapter 1.4) and **<ver>** is an INtime major version.

<b>Filename</b>	<b>Description</b>
<code>bin/&lt;ver&gt;/&lt;drvname&gt;.rta</code>	The hardware specific CAN device driver.
<code>bin/&lt;ver&gt;/ntcan.rsl</code>	The NTCAN library
<code>bin/&lt;ver&gt;/cantest.rta</code>	The <i>cantest</i> application
<code>doc/</code>	API documentation, release notes, etc.
<code>include/ntcan.h</code>	NTCAN header to compile the NTCAN based application.
<code>lib/vc/ntcan.lib</code>	Lib file for Visual Studio to link the NTCAN based application
<code>samples/cantest.c</code>	Source code of the <i>cantest</i> application

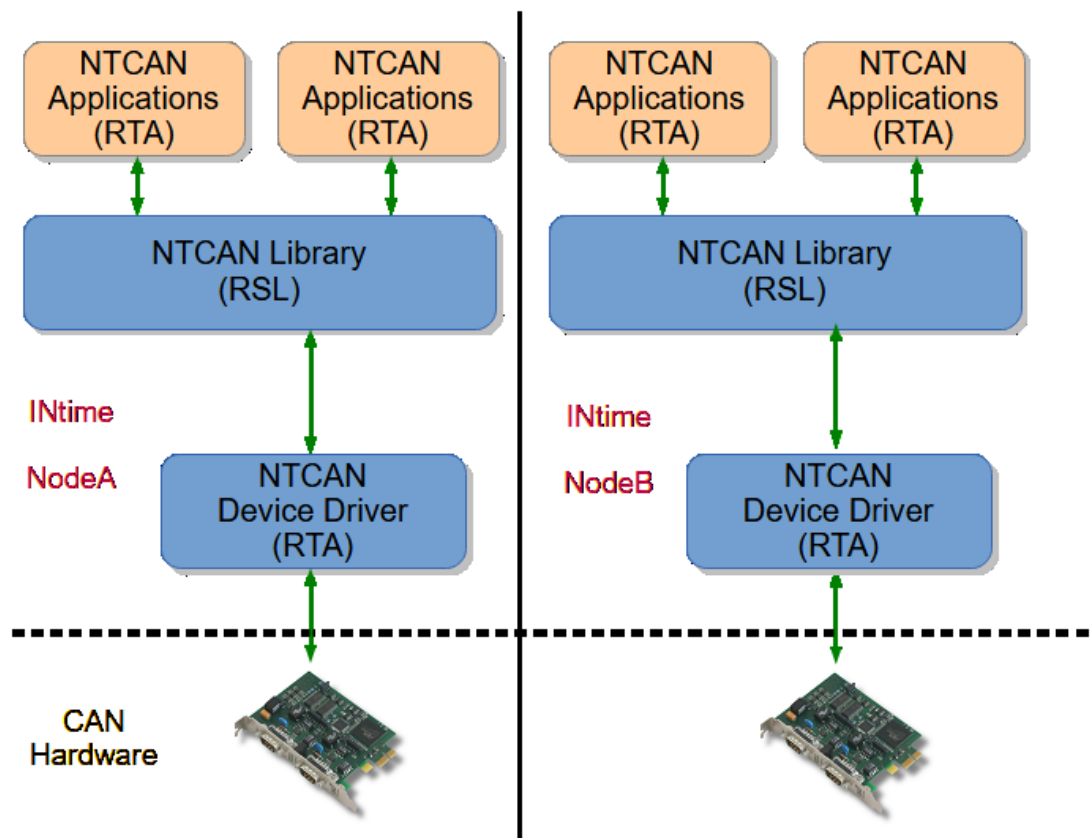
**Table 16:** Files of Intime® CAN driver package

#### 4.4.1 Driver Integration

The support for an **esd** CAN interface consists of a device driver (cxxx.rta) and the NTCAN library (ntcan.rsl) as shown in the picture below. The hardware specific device driver (which usually covers a complete CAN device family) is implemented as an INtime® Real-Time Application (RTA) which creates a resident service after the successful start. The driver must be started before any other RTA which links explicitly or implicitly to the NTCAN library implemented as an INtime® Real-time Shared Library (RSL).

This approach allows using different CAN driver in the INtime® environment with the identical API and allows several INtime® RTAs using the same CAN hardware as well as one INtime® RTA using different CAN hardware.

### INtime for Windows or INtime Distributed RTOS



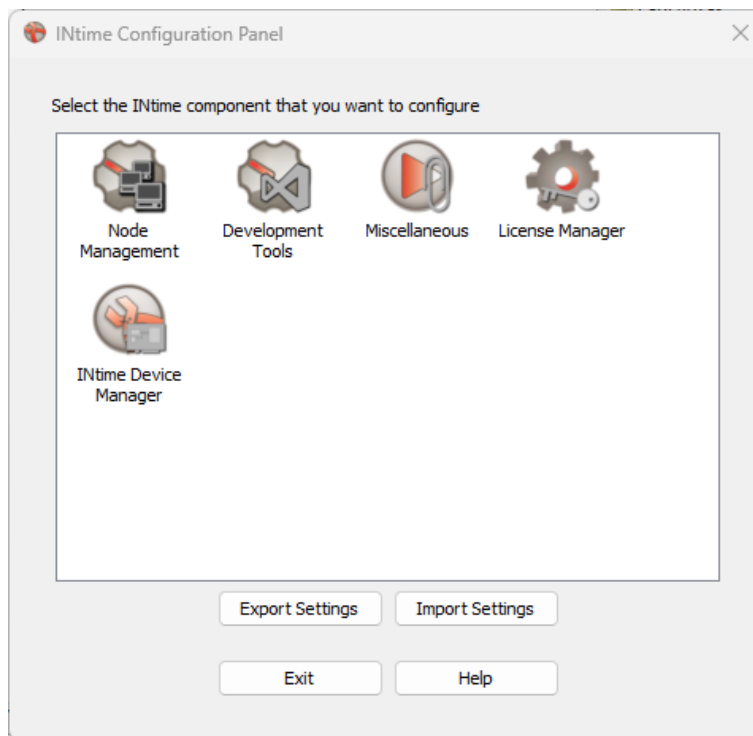
An RTA which uses the CAN hardware must always use the entries exported by the NTCAN RSL described in [11](#) and must communicate directly with the device driver.

### 4.4.2 Driver Installation

To support an **esd** CAN interface in the INtime® environment it is necessary to remove the control from convert it into a RTX managed device as described below (for further details refer to the *INtime SDK Documentation*).

#### 4.4.2.1 INtime for Windows

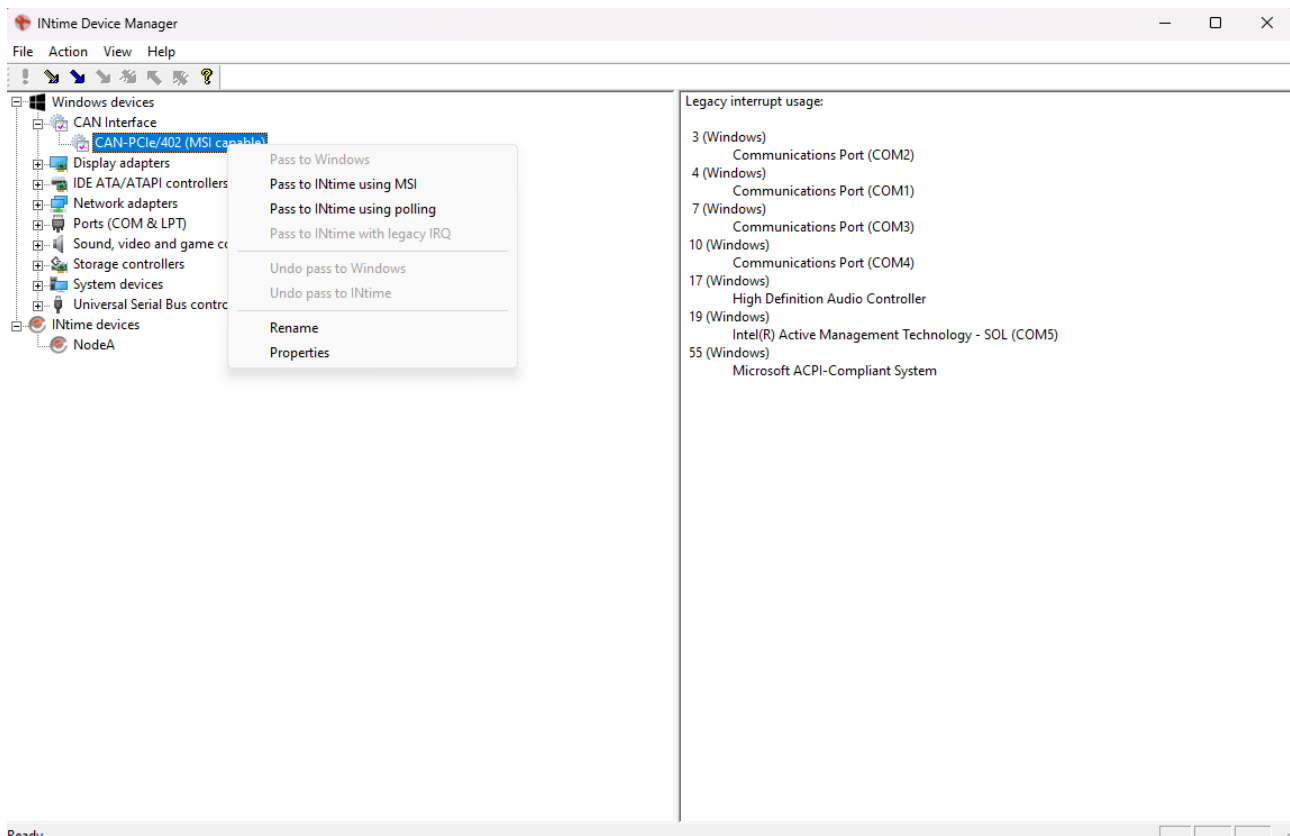
Without additional measures Windows gets control on the CAN hardware and tries to install an associated device drivers as necessary. This is undesired when a device must be controlled from within the INtime environment.



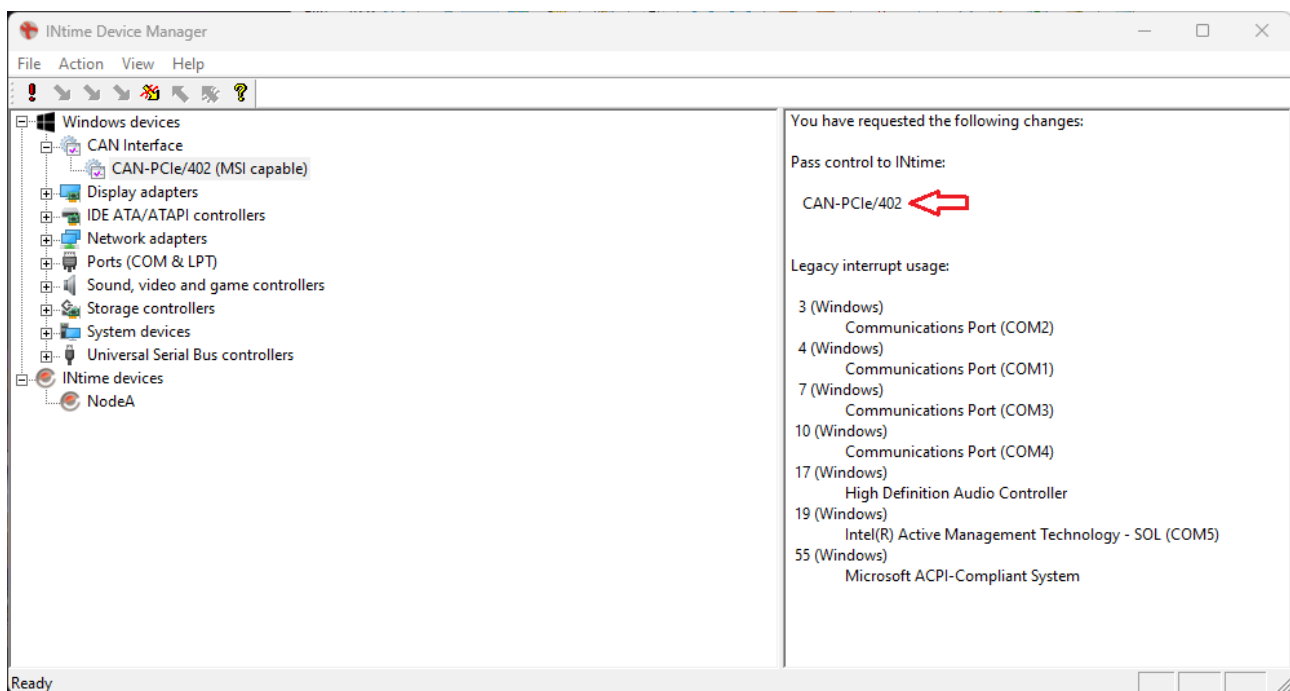
You must use the *INtime Device Manager* to pass control of plug-and-play devices between Windows and a local INtime node. This utility assigns a device to INtime use by installing a simple device driver on the requested device. The device driver does two things:

- It satisfies Windows plug-and-play management to stop it trying to install a normal Windows device driver, and
- It signals to the INtime node that it "owns" the device.

Open the *INtime Device Manager* and open the right-click context menu of the CAN interface which control you will pass from Windows to INtime.



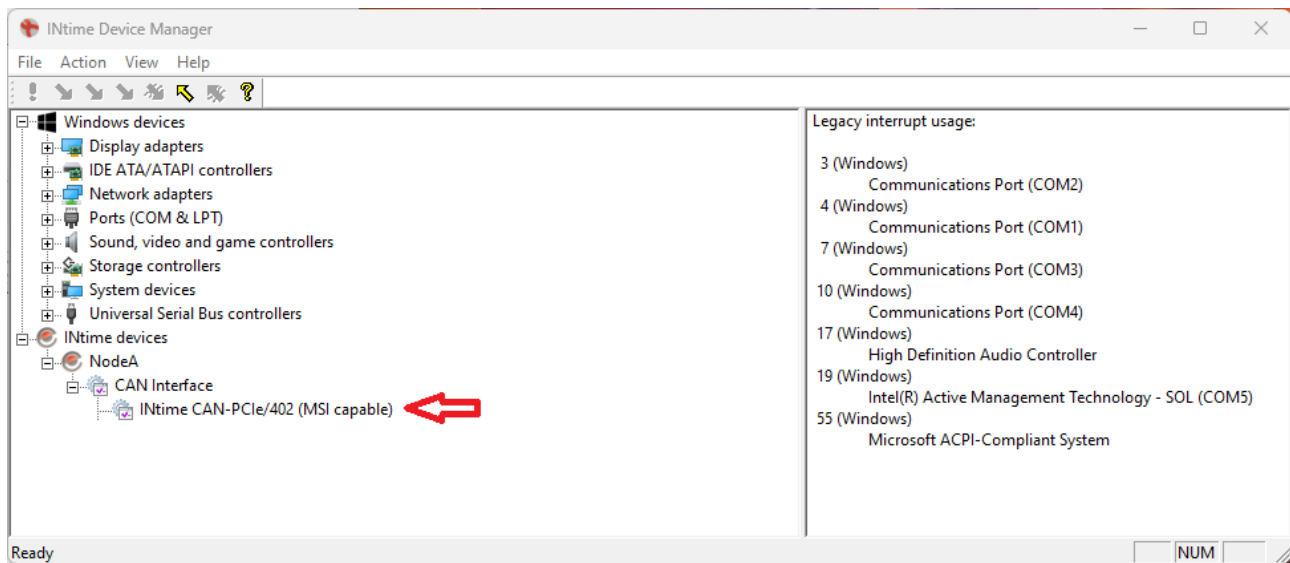
Select “Pass to INtime using MSI” to prepare the use of the CAN hardware in the INtime environment.



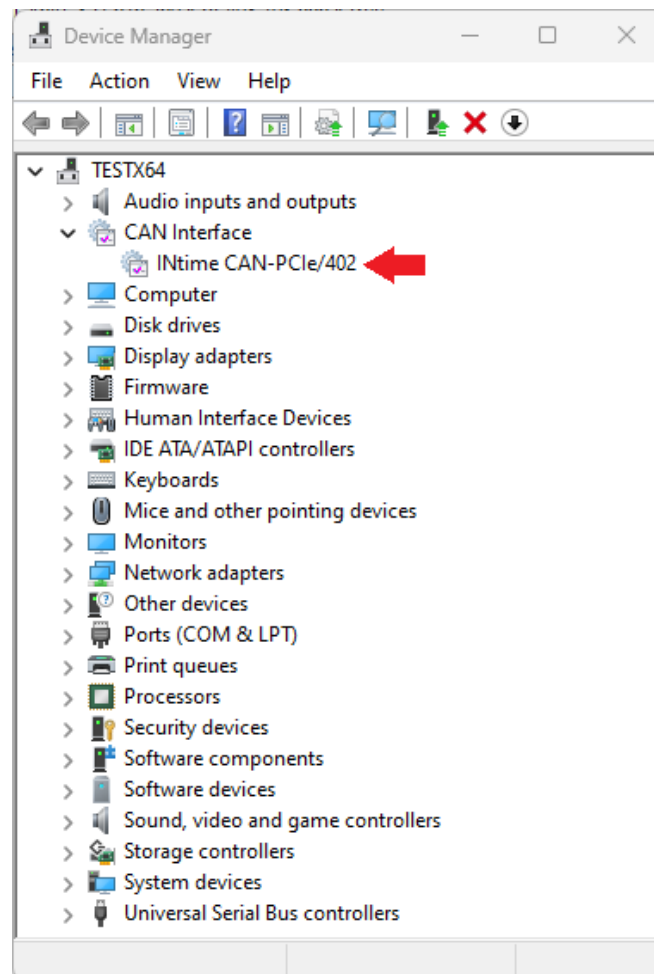
If you exit the *INtime Device Manager*, you will be asked to finalize the requested changes.

## Real-Time Operating Systems

After the changes are applied the *INtime Device Manager* will indicate that the CAN device is assigned to an INtime node.



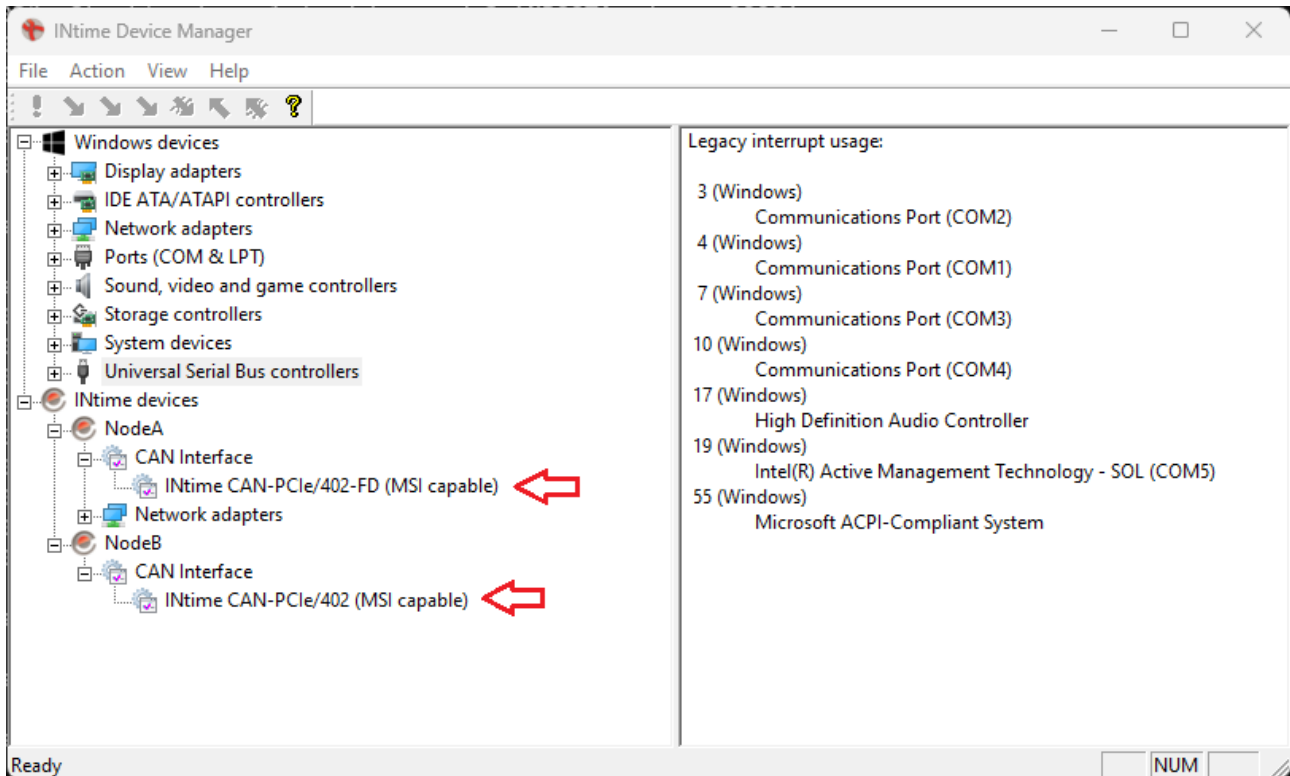
The *Windows Device Manager* will indicate that the device is assigned to INtime.





The INtime CAN device driver is currently limited to supporting one physical CAN board per INtime node. If you need more physical CAN ports available on a single CAN board you must create additional INtime nodes to assign them additional CAN boards.

The picture below shows a configuration with two CAN boards assigned to different INtime nodes. You must start the device driver as described in the next chapter for each node.



### 4.4.3 Driver Start

The INtime kernel is designed to support dynamical loading of executable code. This involves three components:

- **INtime Load Server:** A service within the INtime kernel which handles load requests. The load server manages the loading of both application (RTA) and dynamic libraries (RSL).
- **Windows Load Client:** A Windows application (*ldrta.exe*) which interacts with the load server to load a .RTA file and to create new process with it.
- **INtime Load Client:** An INtime kernel service which allows an INtime application to interact directly with the load server.

This architecture is identical wheter the kernel runs on an INtime node of INtime for Windows or INtime Distributed RTOS.

The driver is loaded with ldrta.exe executing the following command line in a Windows console window:

```
ldrta [-node nodename] [-a "Arguments"] driver.rta
```

where `driver.rta` must be replaced with the name of the driver RTA (e.g. `c402.rta`) and `Arguments` contain the device driver configuration parameter described in chapter 4.4.4.1. For configurations with support for more than one node you also must provide the INtime node name the respective CAN interface is assigned.

After a successful start the root process of the device driver is cataloged with the respective (family) name and is ready to be used by applications (via the NTCAN RSL). In case of any error during the startup a Windows console window is opened to show an error message. In the latter case the driver can be started with the option '-v' followed by a numerical hexadecimal value to increase the verbosity of the debug trace messages reporting initialization errors.



#### 4.4.4 Driver Configuration

The INtime® device drivers are configured with parameters to the *ldrta.exe* command.

##### 4.4.4.1 Command Line Parameter

The INtime® device drivers are configured via options listed in the table below which are passed as arguments to the driver start command described in the previous chapter:

Option	Argument	Description
-V	-	After the start the driver displays driver version information in the Intime® console window and terminates afterwards.
-v	<i>mask</i>	Display additional (debug) messages during startup in the INtime® console window. The mask parameter may be given as a decimal value or as a hexadecimal value prepended by a '0x'
-h	-	Display a list of all possible command line options in the Intime® console window and terminate afterwards.
-p	<i>prio</i>	This option configures the priority of the DPC thread that processes the CAN messages. The default value for <i>prio</i> is 135.
-P	<i>prio</i>	This option configures the priority of the dispatch thread that handles requests from the applications. The default value for <i>prio</i> is 140.
-n	<i>net number</i>	This option configures the logical base net number which is assigned to the first physical port of the CAN interface. The logical net numbers are incremented by one for each additional physical port. The default value for <i>net number</i> is 0.
-w	<i>time in ms</i>	Time window in ms for the HW TX operation mode. The default value is 10 ms.

**Table 17:** Command Line Parameter of INtime® Driver

#### **Example:**

The command

```
ldrta -a "-v0xFF -n2 -p130" c402.rta
```

Starts the C402 device driver with additional (debug) output.

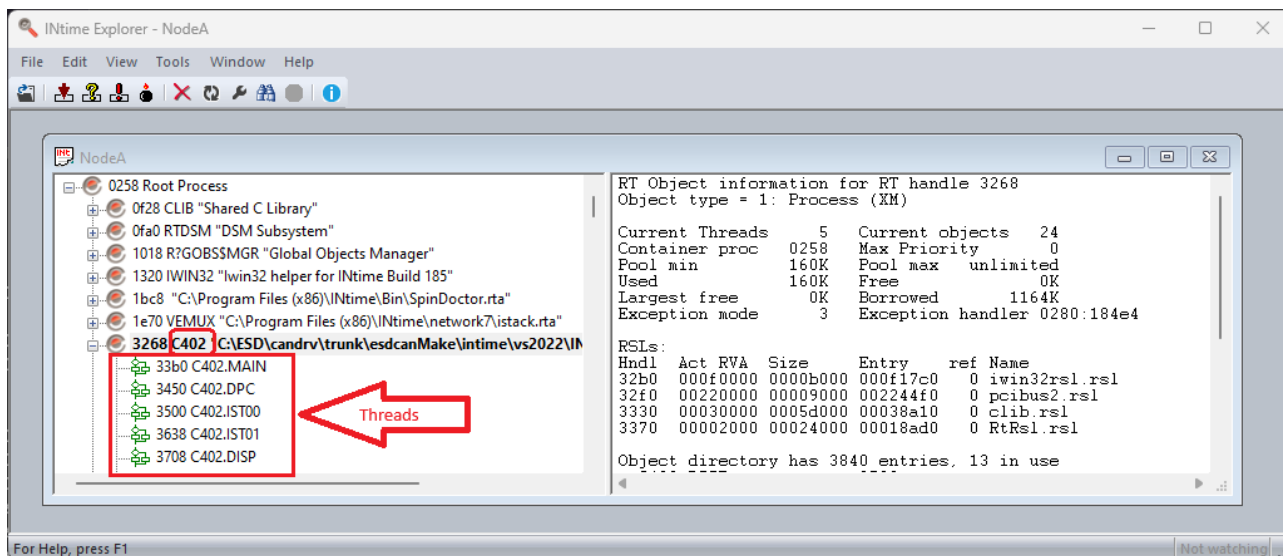
Assigns the logical base-net number 2.

Configures the DPC priority to 130.

If a driver is started without any command line options, the default values according to the table above are applied.

### 4.4.4.2 Priority Layout

The active CAN device driver starts several threads as shown in the picture below for a configuration with two physical CAN boards.



Some of the priorities might be adapted via the command line as described in the previous chapter if the assigned default priorities do not fit into your priority layout. The threads are catalogued with thread names starting with the device driver family name followed by a '.' as described in the table below:

Thread name	Priority	Description
DRV.MAIN	155	The priority of the initial thread created by the INtime load server. Refer to <i>Loading INtime Applications</i> in the <i>INtime SDK</i> for details on how this value can be adapted.
DRV.ISTxx	N/A	One Interrupt Service Thread (IST) for each active physical CAN board. The priority of these threads is assigned by the INtime kernel dynamically in the range from 0 – 130.
DRV.DPC	135	Common CAN driver event handler which priority can be adapted at load time.
DRV.DISP	140	Common CAN driver application message dispatch handler which priority can be adapted at load time.

As a rule, you should always keep the following numerical priority relation:

$$\text{DRV.ISTxx} < \text{DRV.DPC} < \text{DRV.DISP} < \text{RTA}$$

with RTA as priority of the threads which perform NTCAN API calls.

### 4.4.5 Driver Unload

A driver can be gracefully stopped (unloaded) with the INtime tool **killrta.exe** executing the following command line in a Windows console window:

```
killrta [-node nodename] [-name rtaname]
```

or alternatively

```
killrta [-node nodename] [-proc handle]
```

with `rtaname` as cataloged driver name or `handle` as process Id you might figure out with the *INtime Explorer*.

### 4.4.6 Miscellaneous

This chapter covers several topics using the NTCAN architecture on INtime®.

#### 4.4.6.1 Application Development

To develop a NTCAN based applications with Microsoft® Visual Studio the driver package contains the header `ntcan.h` and the library file `ntcan.lib`. To compile for INtime® you must define `__INTIME__` before the header is included. The latter is already defined if you start your project with the Visual Studio INtime wizard to create an RTA. If you



**Note:**

The NTCAN API on Windows and INtime® is identical with the exception that overlapped operations are not supported in the INtime® environment.

#### 4.4.6.2 Example Application

The driver package comes with the INtime® application `cantest.rta` which can be loaded into the INtime® environment with the *RT Application Loader* (`ldrta.exe`) or the *PipeRTA* (`piperta.exe`) tools. Output is displayed in the respective console windows. With the help of this program, you can do basic functionality checks of the CAN interface. The program and its parameters are described in more detail in *I/1* and the source code of this example application is part of the driver package.

Please refer to the chapter *Loading INtime Applications* of the *INtime SDK* for a detailed description of loading RTAs. Any NTCAN API based application requires to link to the NTCAN API RSL which must be accessible by the INtime Load Server e.g. by keeping RTA and RTS in the same folder.



Refer to the *INtime SDK Help* for the search paths which are used by the RT kernel to load RTAs with implicitly or explicitly linked RSLs.

## 4.5 Legacy Support

This chapter covers installation and configuration of CAN device driver for RTOS derivatives which are no longer available at the market and versions of still available RTOS derivatives which have reached the EOL according to the vendors life cycle management.



### Attention!

Active technical support by **esd** and development for these RTOS derivatives or versions have stopped but the latest version of the device driver files are kept available.

### 4.5.1 LynxOS

The software drivers for LynxOS are contained on a CD-ROM with the following files:

File	Description											
instcan	script for loading and unloading the driver											
README.c331	current information and notes	The character combination 'c331' represents the files of module CAN-PCI/331. For other modules this combination is changed as shown below:										
c331	dynamically loadable driver											
c331.info	parameter file for the driver (is read at installation and deinstallation)											
c331dbg	debug version of the dynamically loadable driver (should only be used, if problems arise during installation)											
		<table><tr><th>CAN Module</th><th>File name</th></tr><tr><td>VME-CAN2</td><td>ican2</td></tr><tr><td>VME-CAN4</td><td>ican4</td></tr><tr><td>CAN-ISA/331 CAN-PC104/331</td><td>c331i</td></tr><tr><td>CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331</td><td>c331</td></tr></table>	CAN Module	File name	VME-CAN2	ican2	VME-CAN4	ican4	CAN-ISA/331 CAN-PC104/331	c331i	CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331
CAN Module	File name											
VME-CAN2	ican2											
VME-CAN4	ican4											
CAN-ISA/331 CAN-PC104/331	c331i											
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331											
ntcan.o	ntcan-API											
ntcan.h	header for the ntcan-API											
canupd	program for firmware update											
cantest.c	source code of the example program 'cantest' (see /1/)											
cantest	binary file of the example program 'cantest'											



### Note:

The 'Object mode' of the NTCAN-API is not supported by LynxOS.

### 4.5.1.1 Driver Installation

#### 1. Unpacking the Archive

To unpack the tar-archive the following command has to be called:

```
tar -xvf c331-lynx-v1.0.0.tar
```

'c331' must be entered for the CAN-PCI/331 module. For other modules the character combination shown in the following table must be entered:

Input syntax for unpack the archive:

<i>CAN Module</i>	<i>Entry</i>
VME-CAN2	ican2
VME-CAN4	ican4
CAN-ISA/331 CAN-PC104/331	c331i
CAN-PCI/331 CPCI-CAN/331 PMC-CAN/331	c331

#### 2. Loading the CAN Driver:

```
instcan c331
```

#### 3. Unloading the CAN Driver:

The CAN driver must be unloaded, e.g., after an update of the local firmware in order to reset the processor. In the CAN-PCI/331 module, e.g., the driver can be unloaded by the following command:

```
instcan -u c331
```

### 4.5.2 OnTime RTOS-32

#### 4.5.2.1 Overview

RTOS-32 is a royalty-free hard real-time operating system for x86 CPUs which is (amongst many other things) implementing a subset of the WIN32 API. This operating system and its supporting components are developed and distributed by 'On Time' (<http://www.on-time.com>).

#### 4.5.2.2 Implementation

The support for an **esd** CAN module consists of the NTCAN library part and the hardware specific driver part. For RTOS-32 both parts are linked together into a single, static library. To use the **esd** NTCAN API functionality from within your program, this library has to be linked into your RTOS-32 application.

#### 4.5.2.3 RTOS-32 Software Requirement

The current **esd** NTCAN driver for RTOS-32 has been developed and were tested lately using RTOS-32 version 5.25.

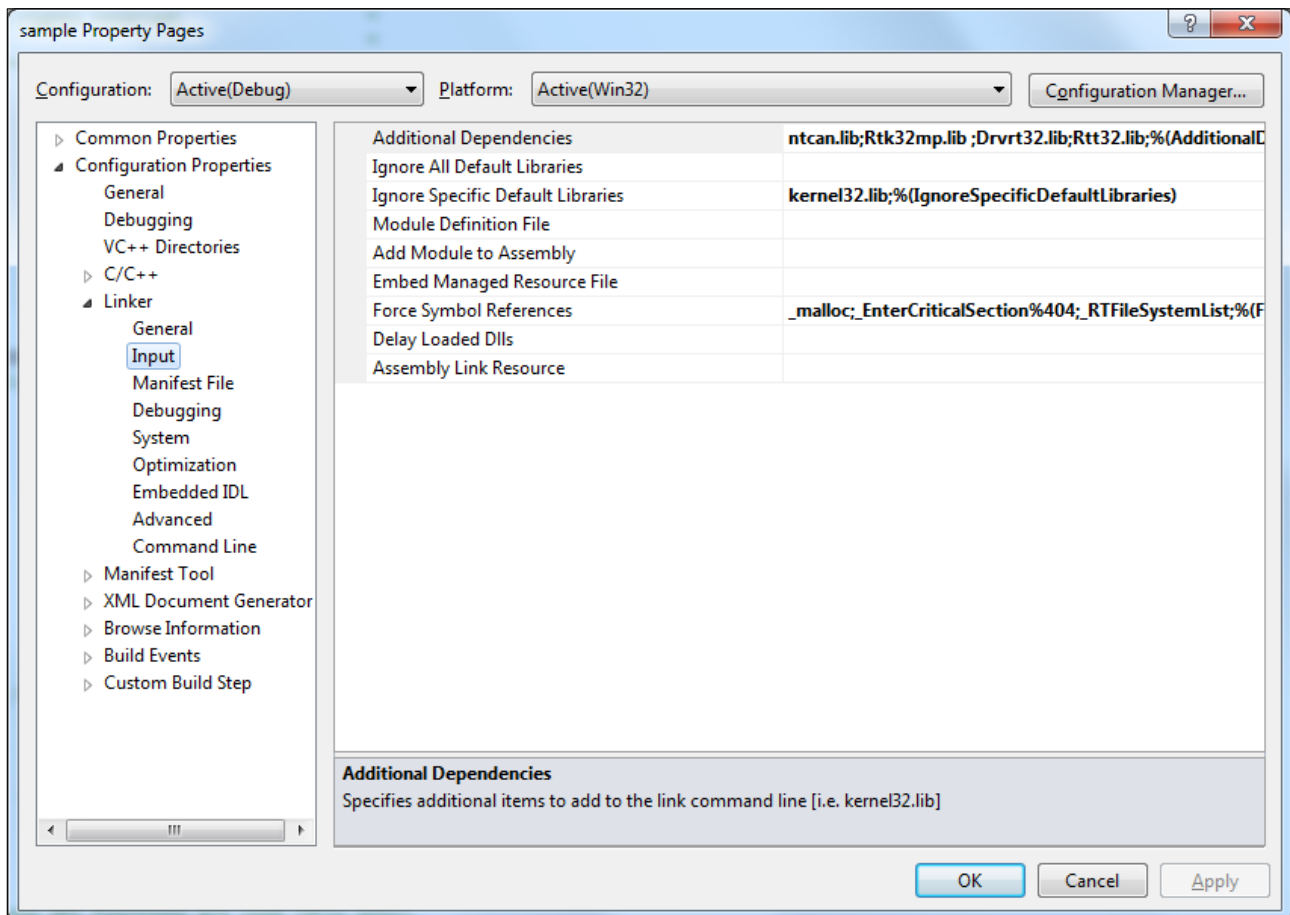
#### 4.5.2.4 Required (RTOS-32) Libraries

The **esd** NTCAN RTOS-32 driver makes use of the following RTOS-32 modules:

- `Rtk32.lib` (or `Rtk32mp.lib` for a multiprocessor kernel). Alternatively use `Rtk32s.lib` or `Rtk32mps.lib` for the standard (non-debug) versions instead
- `Drvrt32.lib` for high-resolution timers, interrupt handling, etc.
- `Rtt32.lib` for the RTTarget-32 native API and Win32-emulation

While linking, the above order of libraries should be retained. See the On Time documentation about 'Order of Libraries'.

In addition, you also must link against `ntcan.lib`.



### 4.5.2.5 Using the esd NTCAN Library

#### 4.5.2.5.1 Folder and File Hierarchy

The distributed **esd** RTOS-32 CAN driver is organised in the following folder hierarchy:

- doc/ - Documentation and release notes
- include/ - Header for application development
- lib/ - Driver library for development with Microsoft Visual Studio
- samples - Sample application (cantest)

#### 4.5.2.5.2 Including Header File `ntcan.h`

First of all, you have to include the header `ntcan.h` into your code. `ntcan.h` is providing function prototypes, type declarations, macros etc. for the **esd** NTCAN API.

Be sure `ntcan.h` is within the preprocessor's search path. When necessary, adapt the preprocessor search path.

#### 4.5.2.5.3 Linking Against `ntcan.lib`

Be sure `ntcan.lib` is within the linker's search path. If needed, adapt the linker search path.

### 4.5.2.5.4 Initialisation Procedure for the esd NTCAN Library

Before using any NTCAN API functionality from within your application you have to do the following three steps:

#### Initialise RTKernel-32

At first you need to call `RTKernelInit()`. Please also consider the On Time documentation.

#### Set up Timer Interrupts

As the second step set up the RTOS-32 timer interrupts. **esd** NTCAN is providing time-outs with a resolution of 1 ms. So we suggest to use a timer interrupt cycle of 1 ms (or below).

```
CLKSetTimerIntVal(1*1000); // 1000µs = 1ms
```

### 4.5.2.5.5 Initialisation of NTCAN itself

As step number three it is mandatory to call `esdcanInit()` once before using the NTCAN API. By means of this function call the **esd** CAN hardware will be setup, an interrupt handler will be installed, etc. Next to this, the esdcan NTCAN driver is ready for use.

Function prototype: `extern int esdcanInit(void)`

On successful completion `esdcanInit` returns 0, in case of an error a value unequal to 0 is returned.

### 4.5.2.5.6 De-initialisation Procedure for the esd NTCAN Library

To properly disconnect the interrupt handler and clean up all other resources claimed by the **esd** NTCAN driver it is mandatory to call `esdcanDeInit` just before leaving your application.

Function prototype: `extern int esdcanDeInit(void)`

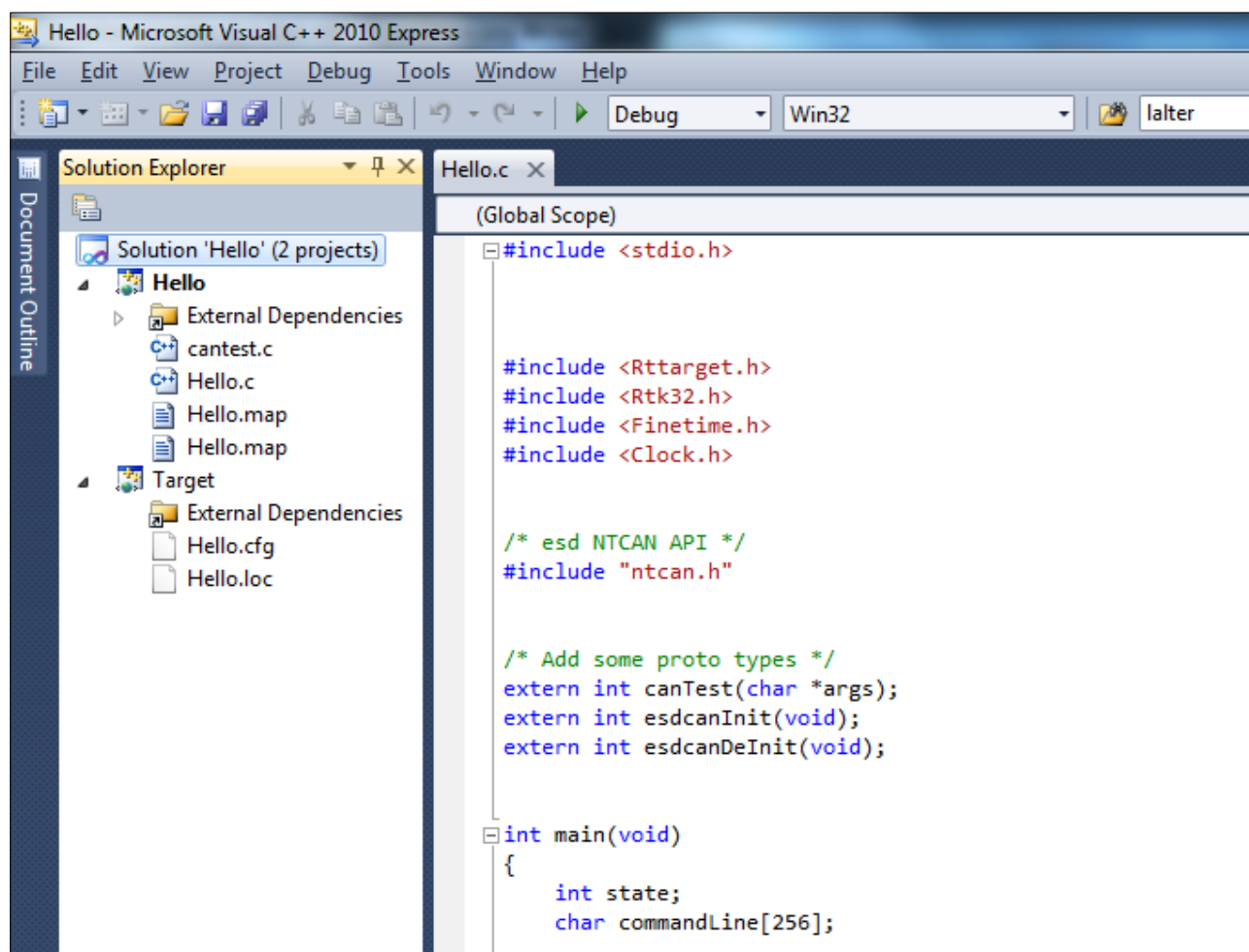


#### 4.5.2.6 Compiling the Sample Application “cantest”

Compiling the sample application cantest by modifying / extending the On Time `hello.c` sample.

##### 4.5.2.6.1 Add / Replace Source Code

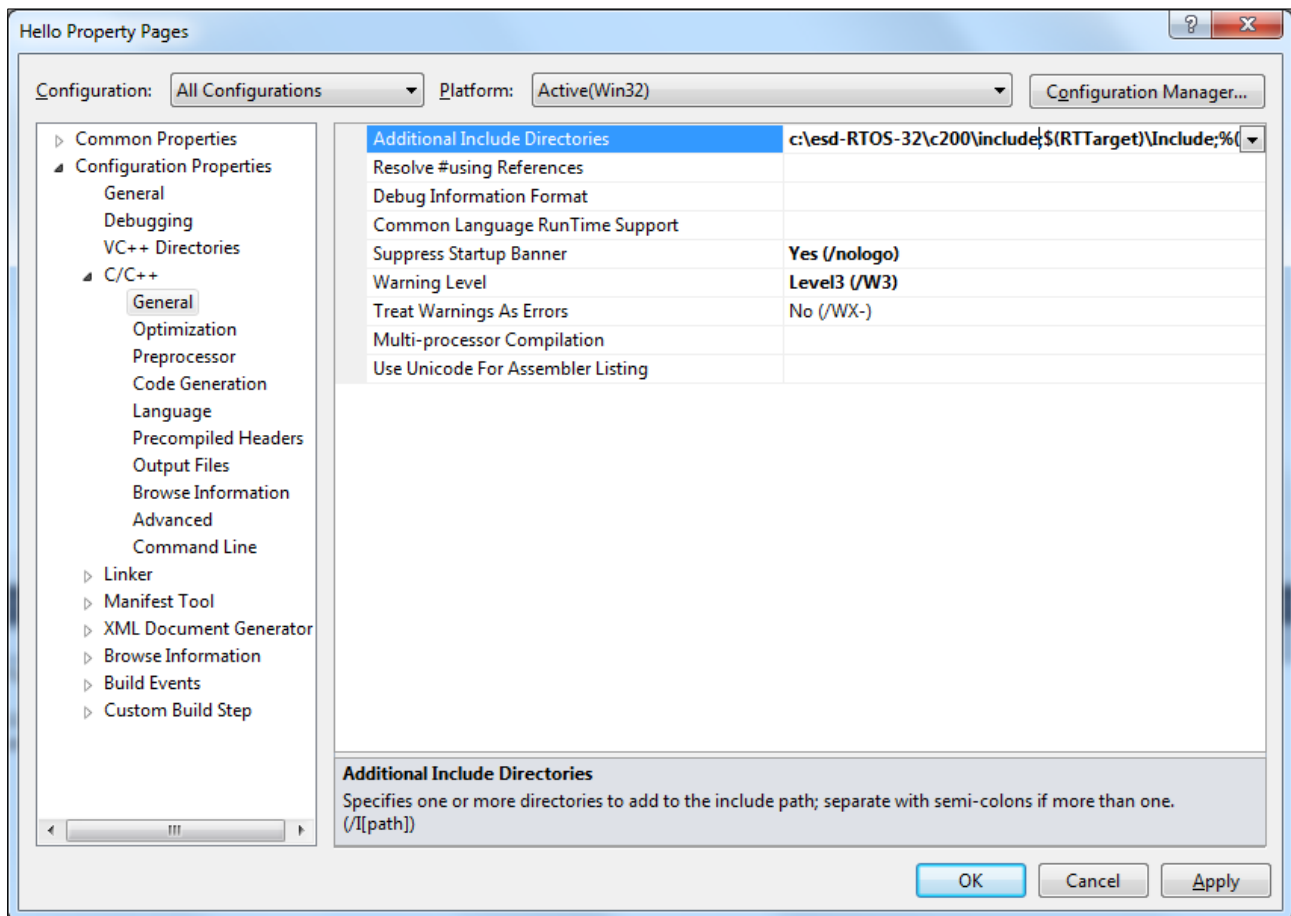
Add `sample/cantest.c` and replace `hello.c` by `sample/hello.c` from within the **esd** RTOS-32 driver distribution:



### 4.5.2.6.2 Adapt Header Search Path

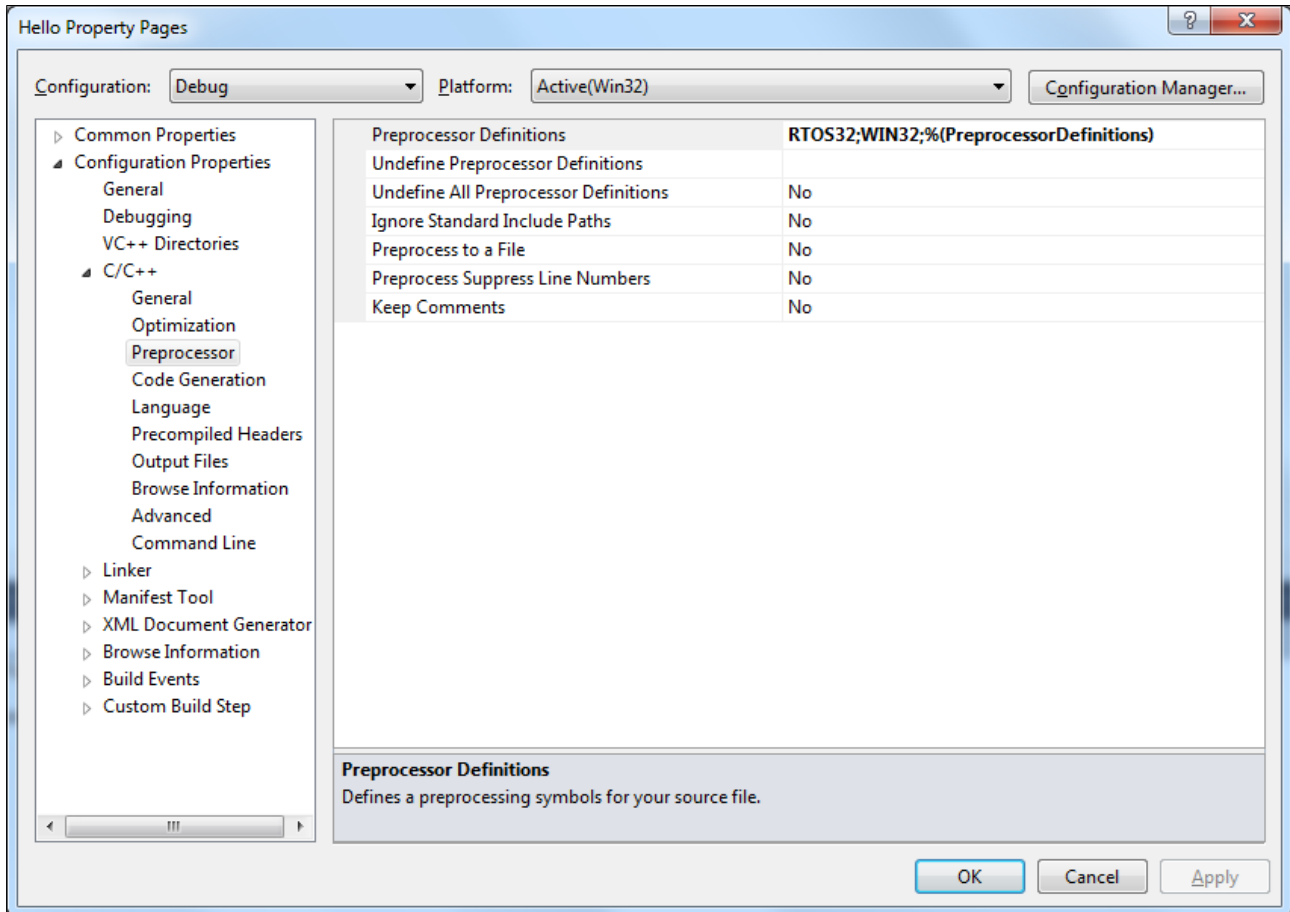
Add the path to `include/ntcan.h` to the preprocessor's additional include directories.

E.g.:



#### 4.5.2.6.3 Define RTOS32

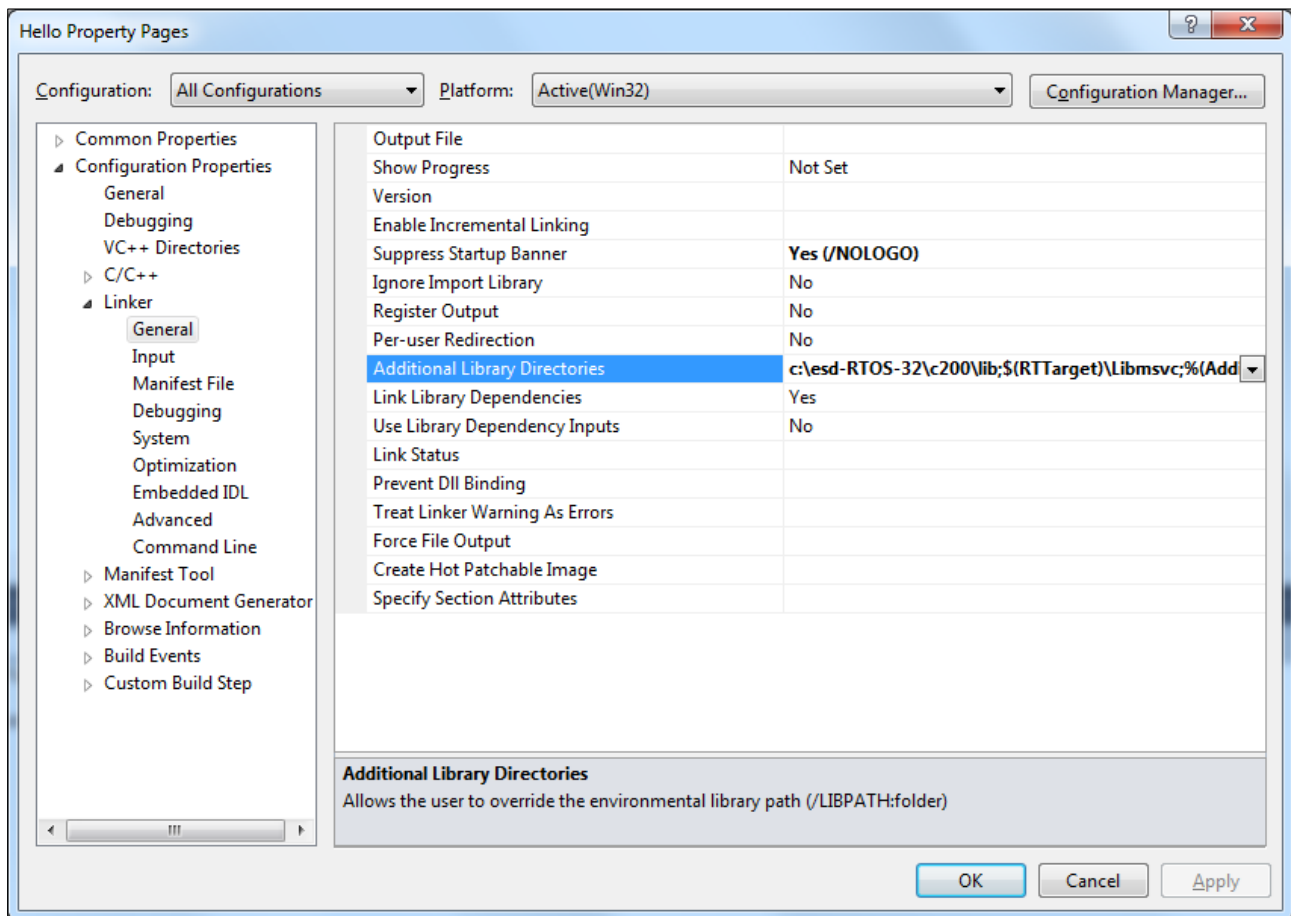
For proper compilation of `cantest.c` add RTOS32 (please use RTOS32 in unhyphenated notation here!) to the preprocessor definitions:



### 4.5.2.6.4 Adapt Library Search Path

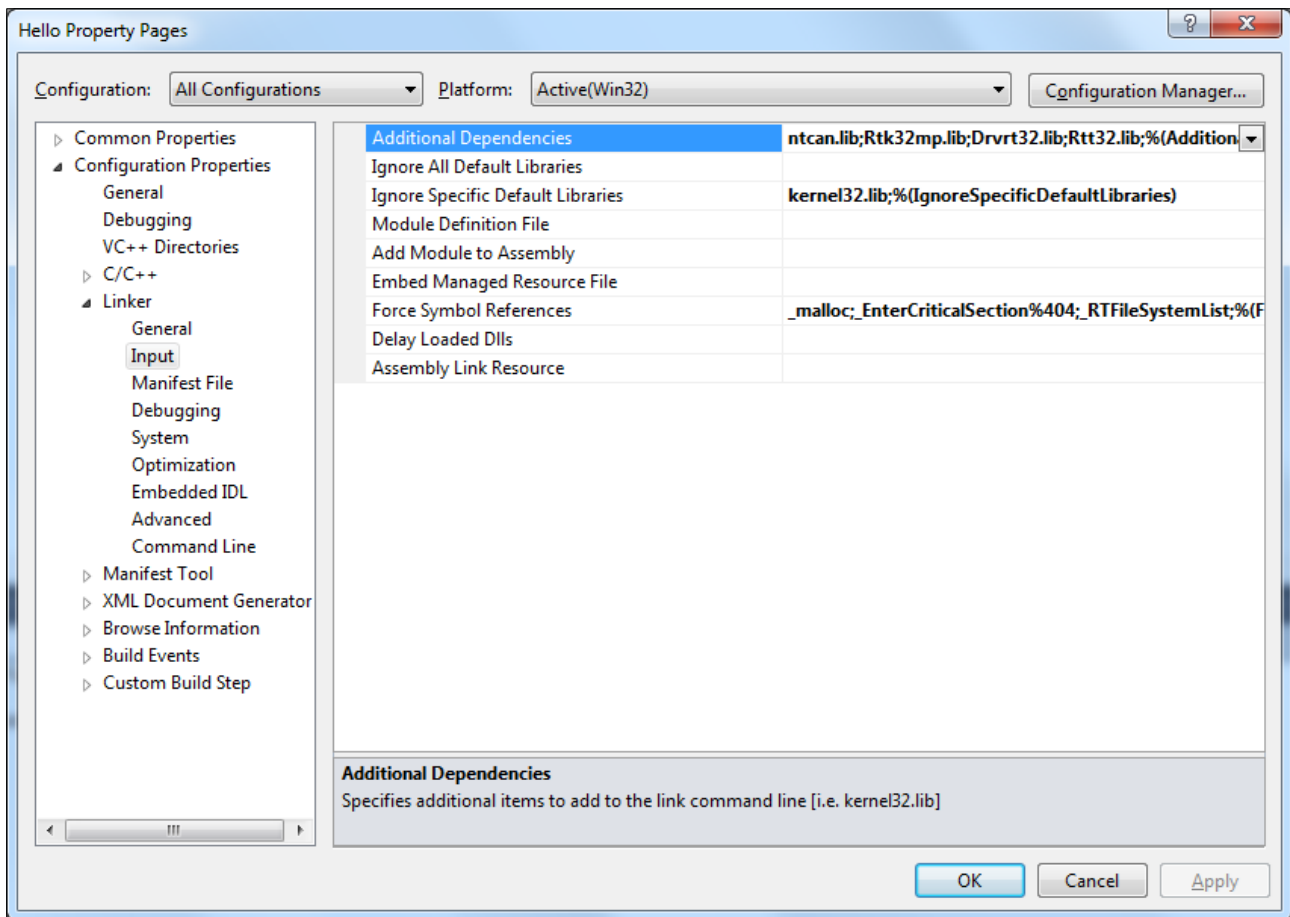
Add the path to `lib/ntcan.lib` to the linker's additional library directories.

E.g.:



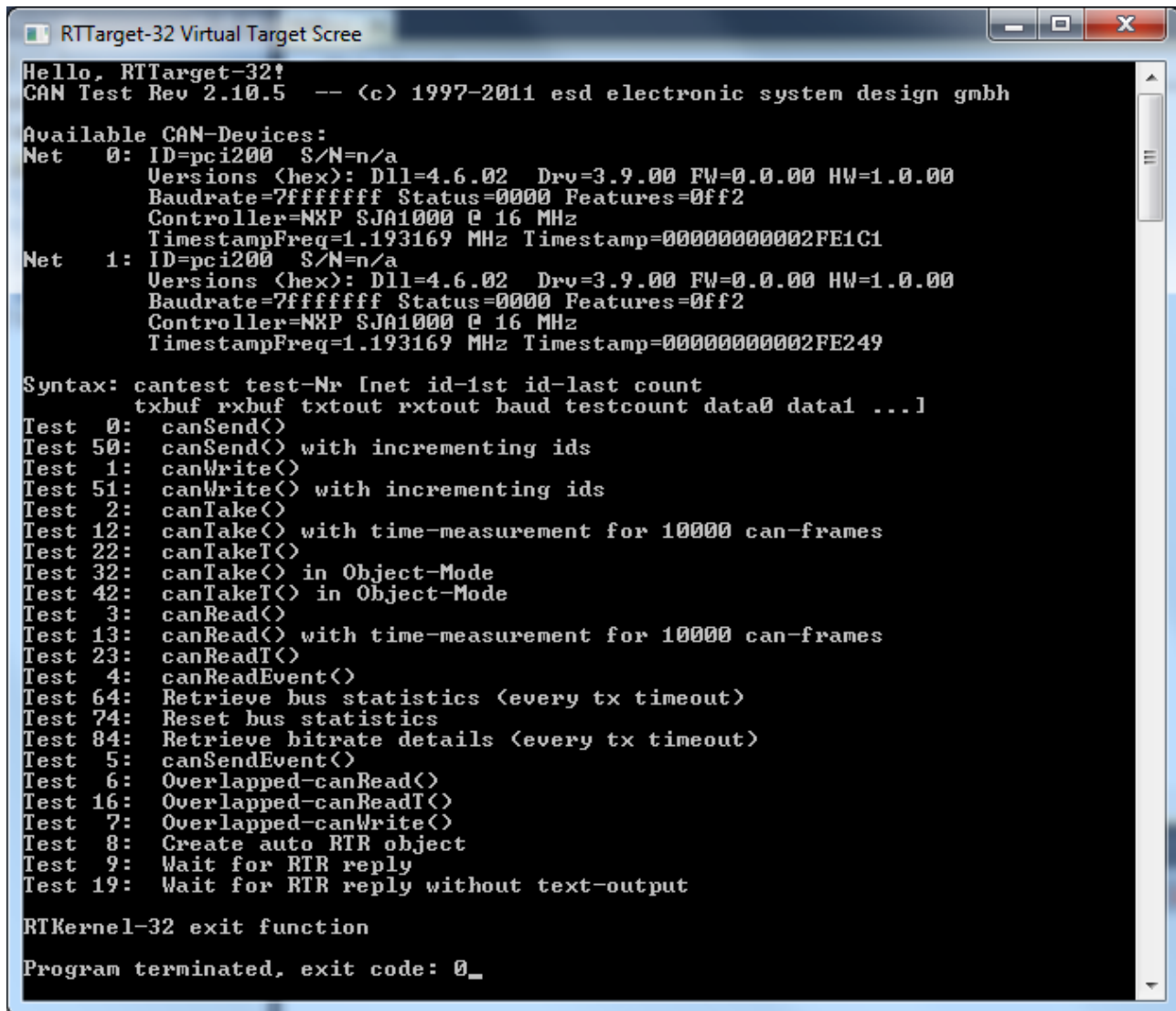
#### 4.5.2.6.5 Add Additional Libraries

Add the needed additional libraries to the linker:



## 4.5.2.6.6 Build and run

Now you are ready to build and execute the solution ...



```

RTTarget-32 Virtual Target Scree
Hello, RTTarget-32!
CAN Test Rev 2.10.5 -- (c) 1997-2011 esd electronic system design gmbh

Available CAN-Devices:
Net 0: ID=pci200 S/N=n/a
      Versions (hex): Dll=4.6.02 Drv=3.9.00 FW=0.0.00 HW=1.0.00
      Baudrate=7fffffff Status=0000 Features=0ff2
      Controller=NXP SJA1000 @ 16 MHz
      TimestampFreq=1.193169 MHz Timestamp=00000000002FE1C1
Net 1: ID=pci200 S/N=n/a
      Versions (hex): Dll=4.6.02 Drv=3.9.00 FW=0.0.00 HW=1.0.00
      Baudrate=7fffffff Status=0000 Features=0ff2
      Controller=NXP SJA1000 @ 16 MHz
      TimestampFreq=1.193169 MHz Timestamp=00000000002FE249

Syntax: cantest test-Nr [net id=1st id-last count
      txbuf rxbuf txtout rxtout baud testcount data0 data1 ...]
Test 0: canSend()
Test 50: canSend() with incrementing ids
Test 1: canWrite()
Test 51: canWrite() with incrementing ids
Test 2: canTake()
Test 12: canTake() with time-measurement for 10000 can-frames
Test 22: canTakeI()
Test 32: canTake() in Object-Mode
Test 42: canTakeI() in Object-Mode
Test 3: canRead()
Test 13: canRead() with time-measurement for 10000 can-frames
Test 23: canReadI()
Test 4: canReadEvent()
Test 64: Retrieve bus statistics (every tx timeout)
Test 74: Reset bus statistics
Test 84: Retrieve bitrate details (every tx timeout)
Test 5: canSendEvent()
Test 6: Overlapped-canRead()
Test 16: Overlapped-canReadI()
Test 7: Overlapped-canWrite()
Test 8: Create auto RTR object
Test 9: Wait for RTR reply
Test 19: Wait for RTR reply without text-output

RTKernel-32 exit function
Program terminated, exit code: 0_
  
```

## 5 Firmware Update Application

Some of the active **esd** CAN boards store their firmware in a NVRAM which is in most cases updatable by the end user. A firmware update to a different version than the one the CAN board is shipped with might be necessary to support new features or to troubleshoot problems. The task of performing the firmware update is handled by a console application which is described in chapter 5.1.

An additional task which is also covered by the same application is switching between active and passive support for CAN messages in the extended frame format (29-bit CAN-IDs) for a certain family of active CAN boards (see chapter 5.2).

The firmware update applications are usually available for Windows and in many cases also as native versions for operating systems with shell support (Linux, LynxOS, QNX, VxWorks, ...). If the firmware update tool is not available as native version for your target operating system you must do the update with a supported OS (e.g. Windows).

The table below gives an overview on the updatable **esd** CAN boards with a firmware stored in the NVRAM. The name of the tool always starts with '**upd**' followed by a CAN board and platform specific extension. An update application for active CAN boards which can be switched between active and passive support for 29-bit CAN-IDs are marked with an asterisk (\*).

<b>CAN Board</b>	<b>Update Tool Name</b>
CAN-PCI/331, CPCI-CAN/331, PMC-CAN/331	updc331*, upd-pci331* or upd-pci331-i20*
CAN-PCI/360, CPCI-CAN/360	updc360*, upd-pci360 or upd-pci360-i20*
CAN-PCIe/402, CAN-PCI/402, CPCIserial-CAN/402, CAN-PCIMini/402, CPCI-CAN/402	updc402
CAN-PCIe/402_FD, CAN-PCI/402-FD, CPCIserial-CAN/402-FD, CAN-PCIMini/402-FD, PMC-CAN/402-FD, XMC-CAN/402-FD	updc402fd
CAN-USB/Mini	updusb331
CAN-USB/Micro	updcanmicro
CAN-USB/2	updusb2292
CAN-USB/2V2	updusb2v2.exe
CAN-USB/3-FD	updusb3fd.exe
CAN-USB/400, CAN-USB/400-IRIG-B	updusb400
CAN-USB/400-FD	updusb400fd
CAN-AIR/2	updcanair2
CAN-ISA/331, CAN-PC104/331	updc331i*, upd-isa331* or upd-isa331-i20*

**Table 18:** Overview of firmware update applications

**Note:**

The EtherCAN and EtherCAN/2 boards also have an updatable firmware but as a network attached device the update is also performed via the network and not with the firmware update tool described here. Please refer to /3/ for further details.

Other active CAN boards not mentioned in the table above have the latest firmware in the device driver binary.

## 5.1 Updating the Firmware

A firmware can be updated with a CAN board and operating system specific console application.

**Attention!**

Even if the updating process is made fail-safe, we do not recommend updating the firmware if the CAN board works without any problems or the update is explicitly recommend/required. For any damages caused by improper operation of updating the firmware **esd** assumes no responsibility.

**Prerequisites:**

For the firmware update the device driver has to be installed and started and you have to make sure that CAN board works properly in your system. You also have to make sure, and no other application is using the CAN board. Otherwise, the update tool will return with an error.

Follow the steps below to update the firmware:

- To start the firmware update you have to open a console window and change into the directory with the firmware update tool for your CAN board.

**Note:**

On Windows the update tool requires a console with Administrator Privileges.

If you start the update tool in a standard console without these privileges modern Windows versions will show the UAC prompt and start the tool in a privileged console which will open and close immediately without any further action.



- Type the name of the update tool which covers you CAN hardware (see Table 18) followed by one of the logical net numbers which is assigned to the CAN board you want to update. Below is an example for a CAN-USB/2 which is assigned the logical net number 6 in the system (1).

```
E:\TEMP\driver\usb2292>updusb2292 6 1

updusb2292 - firmware updater (Version: 2.2.7 - built Sep  5 2008 17:07:44)
(C) esd electronic system design gmbh

Available image:
  card-id       = "USB2"
  firmware-version = 1.0.05 2
  firmware-length = 59392(0xE800)

Current image on board:
  card-id       = "USB2"
  firmware-version = 1.1.00 3

Current Host-Driver:
  driver-version = 2.6.03

||||| WARNING!!! Do not turn off power or reset your machine before the firmware
||||| download is completed! Removing the power before the firmware
||||| update is completed could cause the firmware to be inadvertently
||||| deleted and will render the unit inoperable. If this occurs, the
||||| device will not start up and will require factory service to
||||| reload the firmware.
||||| Update only if you're having problems with the current firmware.
||||| "If it ain't broke, don't fix it."
|||||

The firmware on board is newer! 4
Do you want to continue(y/n)?
```

- The tool will display the version of the firmware the CAN board can be updated to (2) and the active firmware version (3). If the tool can not find a suitable CAN hardware for the given logical net number it just shows the built-in firmware image.
- You will also see a warning message about the dangers of a firmware update and in case you do a downgrade an information that you change to an older firmware version (4).
- If you answer the question to continue with 'y' the firmware update process is started and you will be informed about its progress. If you want to cancel now and keep the active firmware answer with 'n'.



#### Attention!

Once started do not interrupt the update process by aborting the program or resetting the system before the firmware update tool reports that it has updated the firmware successfully.

- After the new firmware is successfully updated an USB based device (with the exception of the CAN-USB/400) will automatically do a reset, the device is re-enumerated by the host system and can be used immediately with the new firmware. CAN boards for other buses require an explicit hardware reset before the new firmware becomes active which can either be performed by rebooting the host system or by loading/unloading the device driver (if this is supported by the host OS).



### Attention!

For devices of the Classical CAN C402 family (**CAN-PCI/402**, **CAN-PCIe/402**, **CAN-PCIMini/402**, **CPCIs serial-CAN/402**, **PMC-CAN/402**, **XMC-CAN/402**), their CAN FD enabled derivatives and the **CAN-USB/400** the method described above for the hardware reset is not sufficient to activate the new firmware. The hardware requires a real power cycle where it is disconnected from the power supply for a second.

## 5.2 Switch between CAN 2.0A and CAN 2.0B Mode

Some **esd** CAN boards (marked in Table 18 with an asterisk) support two different firmware operation modes. One mode (according to CAN 2.0A) can transmit and receive only 11-bit CAN-IDs (passive support of 29-bit CAN-IDs), the other mode (according to CAN 2.0B) can transmit and receive 29-bit as well as 11-bit CAN-IDs. The first version is the factory default, and this chapter describes how you can switch to the other version.



### Note:

Only the CAN boards marked in Table 18 with an asterisk come with two different firmware versions. All other **esd** CAN boards always support the universal CAN 2.0B mode and you can ignore this chapter if you have one of these other boards.

To switch between the two firmware modes the update application offers in addition to the real firmware update capability described in the previous chapter the options '-ta' to switch into the (factory default) 11-bit mode and '-tb' to switch into the 29-/11-bit mode.

Follow the steps below to switch to another firmware version:

- Open a console window and change into the directory with the firmware update application for your CAN board.
- Type the name of the update application which covers your CAN hardware (see Table 18) followed by the option '-ta' to switch into the CAN 2.0A mode or '-tb' to switch into the CAN 2.0B mode followed by one of the logical net numbers which is assigned to the CAN board.
- The CAN board requires an explicit hardware reset before the new firmware mode becomes active which can either be performed by rebooting the host system or by loading/unloading the device driver (if this is supported by the host OS).



### Note:

In comparison to the real firmware update described in the previous chapter just changing the firmware operation mode does not contain any risks to make the CAN board unusable.