

System Manual
PLCmodule-C32

User Manual
Version 1.0

Edition September 2009

Document No.: L-1072e_1

SYS TEC electronic GmbH August-Bebel-Straße 29 D-07973 Greiz
Telefon: +49 (3661) 6279-0 Telefax: +49 (3661) 6279-99
Web: <http://www.systec-electronic.com> Mail: info@systec-electronic.com

Status/Changes

Status: released

Date/Version	Section	Changes	Editor
2009/09/24 1.0	All	Creation	R. Sieber

This manual includes descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (©) symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, SYS TEC electronic GmbH assumes no responsibility for any inaccuracies. SYS TEC electronic GmbH neither guarantees nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. SYS TEC electronic GmbH reserves the right to alter the information contained herein without prior notification and does not accept responsibility for any damages which might result.

Additionally, SYS TEC electronic GmbH neither guarantees nor assumes any liability for damages arising from the improper usage or improper installation of the hardware or software. SYS TEC electronic GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2009 SYS TEC electronic GmbH. All rights – including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part – are reserved. No reproduction may occur without the express written consent from SYS TEC electronic GmbH.

Inform yourselves:

Contact	Direct	Your local distributor
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	Please find a list of our distributors under http://www.systemec-electronic.com/distributors
Ordering Information:	+49 (0) 36 61 / 62 79-0 info@systemec-electronic.com	
Technical Support:	+49 (0) 36 61 / 62 79-0 support@systemec-electronic.com	
Fax:	+49 (0) 36 61 / 6 79 99	
Web Site:	http://www.systemec-electronic.com	

1st Edition September 2009

Table of Contents

1	Introduction	5
2	Overview / Where to find what?	6
3	Product Description	8
4	Optional Accessory	10
4.1	IO-Box.....	10
4.2	USB-RS232 Adapter Cable	10
4.3	Driver Development Kit.....	11
5	Pinout of the PLCmodule-C32.....	12
5.1	Pin assignment.....	12
5.2	Power supply (24VDC).....	14
5.3	In- and outputs for industrial standard signals.....	15
5.3.1	Digital inputs DI0 ... DI23 (24VDC).....	15
5.3.2	Counter inputs C0 ... C2 (24VDC).....	15
5.3.3	Digital outputs DO0 ... DO15 (24VDC / 0.5A, short-circuit-proof)	16
5.3.4	Relay outputs REL0 ... REL3 (250VAC / 3A)	16
5.3.5	Pulse outputs P0 ... P1 (24VDC / 0.5A, short-circuit-proof).....	17
5.3.6	Analog inputs AI0 ... AI3 (0 ... +10V / 0 ... 20mA).....	17
5.3.7	Analog outputs AO0 ... AO1 (0 ... +10V).....	18
5.4	Communication interfaces	18
5.4.1	Serial interfaces ASC0 ... ASC2	18
5.4.2	CAN interfaces CAN0 ... CAN1.....	19
5.4.3	Ethernet interface ETH0	19
6	PLC functionality of the PLCmodule-C32	20
6.1	Overview.....	20
6.2	System start of the PLCmodule-C32	20
6.3	Programming the PLCmodule-C32	21
6.4	Process image of the PLCmodule-C32	22
6.4.1	Local In- and Outputs	22
6.4.2	Network variables for CAN1	23
6.5	Communication interfaces	23
6.5.1	Serial interfaces	23
6.5.2	CAN interfaces.....	24
6.5.3	Ethernet interface	24
6.6	Specific peripheral interfaces	24
6.6.1	Counter inputs	24
6.6.2	Pulse outputs.....	25
6.7	Control and display elements	25
6.7.1	Run/Stop switch.....	25
6.7.2	Run-LED (green)	25
6.7.3	Error-LED (red).....	26
6.8	Local deletion of a PLC program.....	27
6.9	Using CANopen for CAN interfaces	27
6.9.1	CAN interface CAN0.....	28
6.9.2	CAN interface CAN1.....	29
7	Configuration and Administration of the PLCmodule-C32.....	32
7.1	System requirements and necessary software tools.....	32
7.2	Activation/Deactivation of Linux Autostart	33
7.3	Ethernet configuration of the PLCmodule-C32.....	34
7.4	PLC configuration of the PLCmodule-C32	36
7.4.1	PLC configuration via WEB-Frontend.....	36

7.4.2	PLC configuration via control elements of the PLCmodule-C32	38
7.4.3	Setup of the configuration file "plcmodule-c32.cfg"	39
7.5	Boot configuration of the PLCmodule-C32	41
7.6	Selecting the appropriate firmware version	41
7.7	Predefined user accounts	43
7.8	Login to the PLCmodule-C32	44
7.8.1	Login to the command shell	44
7.8.2	Login to the FTP server	45
7.9	Adding and deleting user accounts	47
7.10	How to change the password for user accounts	47
7.11	Setting the system time	48
7.12	File system of the PLCmodule-C32	49
7.13	Software update of the PLCmodule-C32	50
7.13.1	Updating the PLC firmware	50
7.13.2	How to update the Linux-Image	52
8	Data exchange via shared process image	55
8.1	Overview of the shared process image	55
8.2	API of the shared process image client	59
8.3	Creating a user-specific client application	62
8.4	Example for using the shared process image	65
Appendix A: Firmware function scope of PLCmodule-C32		69
Appendix B: Technical Specification		72
Appendix C: GNU GENERAL PUBLIC LICENSE		74
Index		79

1 Introduction

Thank you that you have decided for the SYS TEC PLCmodule-C32. This product provides to you an innovative, Linux-based and high-capacity compact controller to process standard industrial signals. Due to its numerous in- and outputs and communication interfaces it is well-suitable as central control in distributed automation appliances.

Please take some time to read through this manual carefully. It contains important information about the commissioning, configuration and programming of PLCmodule-C32. It will assist you in getting familiar with the functional range and usage of the PLCmodule-C32. This document is complemented by other manuals, e.g. for the *OpenPCS* IEC 61131 programming system and the CANopen extension for IEC 61131-3. Table 1 in section 2 lists relevant manuals for PLCmodule-C32. Please also refer to those complementary documents.

For more information, optional products, updates et cetera, we recommend you to visit our website: <http://www.systemec-electronic.com>. The content of this website is updated periodically and provides to you downloads of latest software releases and manual versions.

Declaration of Electro Magnetic Conformity for PLCmodule-C32 (EMC law)



The PLCmodule-C32 is a ready-to-use and tested device and must only be used in this condition.

During its operation, no cables longer than 3m length are allowed to be connected to the module without further protective circuit and inspection. In particular, only screened CAN and Ethernet cables must be used.

The CE-conformity is only valid for the application area described in this document and only under compliance with the following commissioning instructions!

2 Overview / Where to find what?

Table 1 lists all manuals relevant for PLCmodule-C32. To program the PLCmodule-C32 as PLC according to IEC 61131-3, the programming environment *OpenPCS* is used. There are also some manuals for *OpenPCS* that describe the usage and SYS TEC-specific extensions. Those are part of the software package "*OpenPCS*".

Table 1: Overview of relevant manuals for the PLCmodule-C32

Information about...	In which manual?
Basic information about the PLCmodule-C32 (connections, configuration, administration, process image, firmware update etc.)	In this manual
Development of user-specific C/C++ applications for PLCmodule-C32 (the PLCmodule-C32 is based on the ECUcore-5484 / PLCcore-5484), VMware image of the Linux development system	System Manual ECUcore-5484 (Manual no.: L-1102)
Basics about the <i>OpenPCS</i> IEC 61131 programming system	Brief instructions for the programming system (Entry " <i>OpenPCS Dokumentation</i> " in the <i>OpenPCS</i> program group of the start menu) (Manual no.: L-1005)
Complete description about the <i>OpenPCS</i> IEC 61131 programming system, basics of PLC programming according to IEC 61131-3	Online help about the <i>OpenPCS</i> programming system
Command overview and description of standard function blocks according to IEC 61131-3	Online help about the <i>OpenPCS</i> programming system
SYS TEC extension for IEC 61131-3: <ul style="list-style-type: none"> - String functions - UDP function blocks - SIO function blocks - FB for RTC, Counter, EEPROM, PWM/PTO 	User Manual " <i>SYS TEC-specific extensions for OpenPCS / IEC 61131-3</i> " (Manual no.: L-1054)
CANopen extension for IEC 61131-3 (network variables, CANopen function blocks)	User Manual " <i>CANopen extension for IEC 61131-3</i> " (Manual no.: L-1008)
Textbook about PLC programming according to IEC 61131-3	IEC 61131-3: Programming Industrial Automation Systems John/Tiegelkamp Springer-Verlag ISBN: 3-540-67752-6 (a short version is available as PDF on the <i>OpenPCS</i> installation CD)

- Section 5** this manual describes **electric connections** of PLCmodule-C32 and their application; moreover, it documents their **internal structure**.
- Section 6** includes details about the **usage of PLCmodule-C32**, e.g. the **setup of the process image**, the **meaning of control elements** and this section provides basic information about programming the module. Furthermore, it gives information about the usage of CAN interfaces in combination with **CANopen**.
- Section 7** makes available **details about the configuration of PLCmodule-C32**, e.g. the configuration of Ethernet and CAN interfaces, the procedure of Linux Autostart and the selection of the firmware version. In addition, the **administration of PLCmodule-C32** is explained, e.g. the login to the system, the user administration and the execution of software updates.
- Section 8** covers information about data exchange between a PLC program and a user-specific C/C++ application via **shared process image**.

3 Product Description

The PLCmodule-C32 extends the SYS TEC electronic GmbH product range within the field of control applications. It is an innovative, Linux-based compact controller for universal processing purposes of standard industrial signals. The controller module provides to the user numerous local in- and outputs as well as versatile communication interfaces. Due to CAN and Ethernet interfaces, the PLCmodule-C32 is suited for realizing decentral control tasks in distributed fieldbus systems of automation technology.



Figure 1: Top view of PLCmodule-C32

These are some significant features of PLCmodule-C32:

- Linux-based compact PLC for industrial controls
- High-capacity CPU kernel (Freescale 32-Bit MCF5484 ColdFire, 200 MHz CPU clock, 300 MIPS)
- 64 MByte SDRAM Memory, 16 MByte FLASH Memory
- 1x 10/100 Mbps Ethernet LAN interface
- 2x CAN 2.0B interface, usable as CANopen Manager (CiA 302-conform)
- 3x asynchronous serial ports (UART)
- 24 digital inputs 24VDC, galvanic isolated
- 16 digital outputs 24VDC/500mA, galvanic isolated, short-circuit-proof
- 4 Relay outputs 250VAC/3A (3x closed contact, 1x two-way contact)
- 4 analog inputs 0-10VDC (0-20mA) with 10-Bit resolution
- 2 analog outputs 0-10VDC, short-circuit-proof
- 3 high-speed counter inputs 24VDC(50kHz), galvanic isolated
- 2 PWM/PTO outputs 24VDC/500mA
- RTC (battery-buffered)
- Temperature sensor
- On-board software: Linux, PLC firmware, CANopen Master, HTTP and FTP server
- Programmable according to IEC 61131-3 and in C/C++
- Function block libraries for communication (CANopen, Ethernet and UART)
- Function block libraries for hardware components (RTC, Counter, PWM/PTO)
- Linux-based (other user programs are executable in parallel)
- Easy, HTML-based configuration via WEB Browser
- Remote login via Telnet
- Dimensions: 160 x 90 x 75mm
- Transparent front cover
- Suitable for DIN top hat rail mounting

There are different types of firmware available for PLCmodule-C32. They differ regarding the protocol used for communication between the Programming PC and the PLCmodule-C32:

Order no.: 3090001: PLCmodule-C32/Z4 (CANopen)
Communication with Programming PC via CANopen Protocol
(Interface CAN0)

Order no.: 3090002: PLCmodule-C32/Z5 (Ethernet)
Communication with Programming PC via UDP Protocol
(Interface ETH0)

The PLCmodule-C32 is an all-round PLC for complex industrial control tasks. As Linux-based compact controller, the module is programmable in C/C++ and in IEC 61131-3. It allows for highly efficient software development for this module. The on-board firmware of PLCmodule-C32 contains the entire PLC runtime environment including CANopen connection with CANopen Master functionality. Thus, the module is able to operate control tasks such as linking in- and outputs or converting rule algorithms. Data and occurrences can be exchanged with other nodes (e.g. superior main controller, I/O slaves and so forth) via CANopen network, Ethernet (UDP protocol) and serial interfaces (UART). The numerous in- and outputs that the module provides can be decentrally extended by CANopen devices. CANopen IO modules of *sysWORXX Automation Series* are well-suited for this. Those modules are also designed for processing industrial standard signals (24VDC, 0-10VDC, 0-20mA etc.).

Programming the PLCmodule-C32 takes place according to IEC 61131-3 using the programming system *OpenPCS* of the company infoteam Software GmbH (<http://www.infoteam.de>). This programming system has been extended and adjusted for the PLCmodule-C32 by the company SYS TEC electronic GmbH. Hence, it is possible to program the PLCmodule-C32 graphically in KOP/FUB, AS and CFC as well as textually in AWL or ST. Downloading the PLC program onto the module takes place via Ethernet or CANopen – depending on the firmware that is used. Addressing in- and outputs and creating a process image follows the SYS TEC scheme for compact control units. Hence, PLC programs developed by the user can be operated on different SYS TEC control modules without adjustments. Like all other SYS TEC controls, the PLCmodule-C32 supports backward documentation of the PLC program as well as the debug functionality including watching and setting variables, single cycles, breakpoints and single steps.

The PLCmodule-C32 is based on the SYS TEC *PLCcore-5484* which is an insert-ready core module that realizes a complete compact PLC. It is also separately available as core module. The PLCmodule-C32 adapts all in- and outputs of PLCcore-5484 to industrial standard signals (24VDC, 0-10VDC, 0-20mA etc.) and complements the core module by a housing which is industry-proven and suitable for DIN rail mounting. The configuration and programming of PLCmodule-C32 are almost identical to the procedures for PLCcore-5484.

The PLCmodule-C32 uses Embedded Linux as operating system. This allows for an execution of other user-specific programs while PLC firmware is running. If necessary, those other user-specific programs may interchange data with the PLC program via the process image. More information about this is provided in section 8.

The Embedded Linux applied to PLCmodule-C32 is licensed under GNU General Public License, version 2. Appendix C contains the license text. All sources of LinuxBSP are included in the software package **SO-1095** ("VMware-Image of the Linux development system for ECUcore-5484"). If you require the LinuxBSP sources independently from the VMware-Image of the Linux development system, please contact our support:

support@systec-electronic.com

The PLC system and the PLC- and C/C++ programs developed by the user are **not** subject to GNU General Public License!

4 Optional Accessory

4.1 IO-Box

The SYS TEC IO-Box (order number phyPS-451) is a universal in- and output device in robust aluminum housing. It allows for easy simulation of analog and digital in- and outputs. Hence, the IO-Box makes easy the commissioning of PLCmodule-C32 and supports the software development and testing - independently from the actual process.

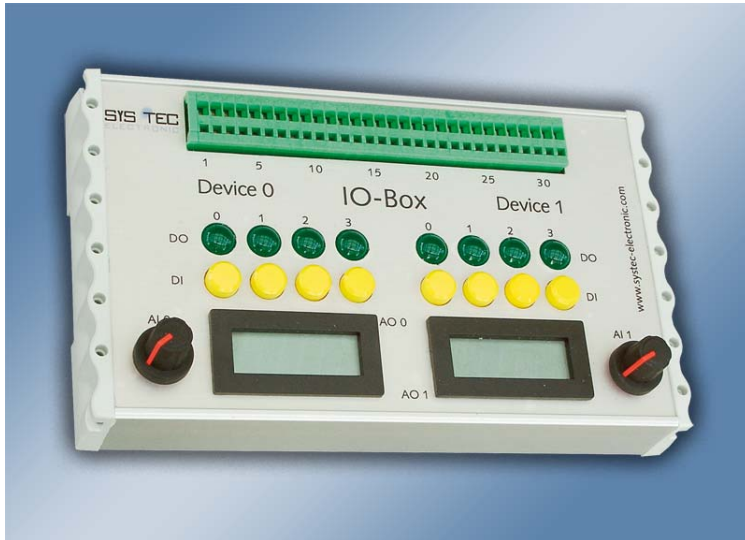


Figure 2: SYS TEC IO-Box

4.2 USB-RS232 Adapter Cable

The SYS TEC USB-RS232 Adapter Cable (order number 3234000) provides a RS232 interface via an USB-Port of the PC. Together with a terminal program, it enables the configuration of the PLCmodule-C32 from PCs, e.g. laptop computers which do not have RS232 interfaces any more (see section 7.1).



Figure 3: SYS TEC USB-RS232 Adapter Cable

4.3 Driver Development Kit

The Driver Development Kit for ECUcore-5484 (order number SO-1098) allows the user to independently adjust the I/O level to own specific application requirements. The Embedded Linux used for PLCmodule-C32 (resp. for the PLCcore-5484 that it contains) supports dynamic loading of drivers during runtime and consequently allows for a separation of PLC runtime system and I/O driver. Hence, the user is able to adjust the I/O driver to own requirements - without having to modify the PLC runtime system itself.

5 Pinout of the PLCmodule-C32

5.1 Pin assignment

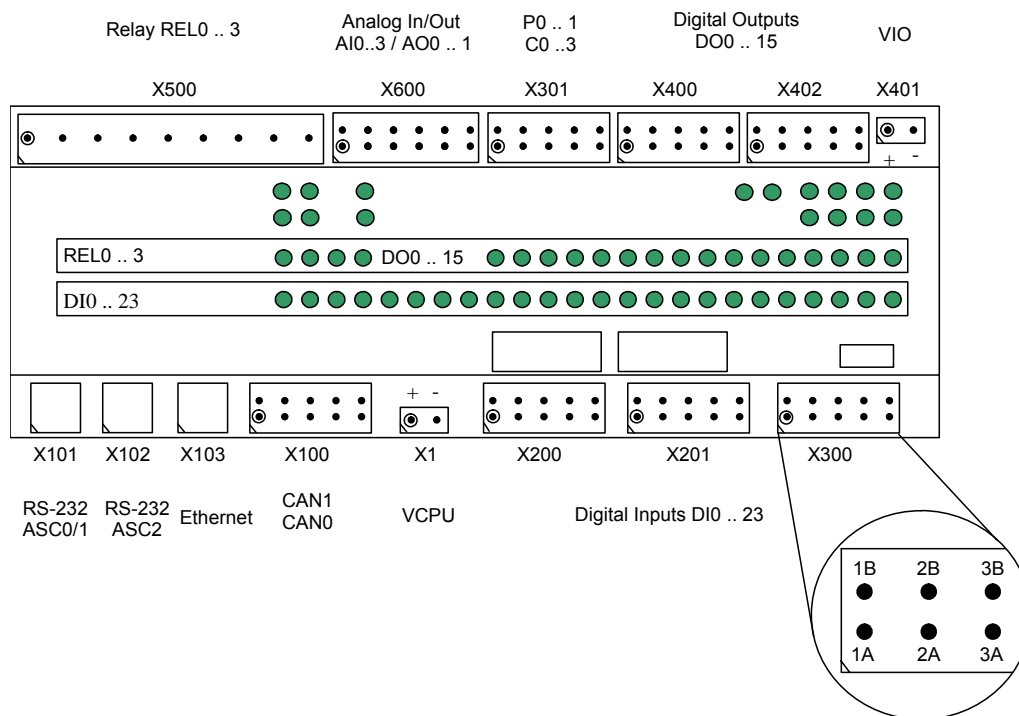


Figure 4: Pinout of PLCmodule-C32

Figure 4 shows the positioning of connectors on the PLCmodule-C32 as an overview. Table 2 lists all connectors in detail.

Table 2: Pin assignment of PLCmodule-C32

Interface	Assignment	Pin	Remark
RS-232 ASC0	RxD	X101.4	RJ-11 connector
	TxD	X101.2	
	GND	X101.3	
RS-232 ASC1	RxD	X101.6	RJ-11 connector
	TxD	X101.1	
	GND	X101.5	
RS-232 ASC2	DSR	X102.1	RJ-45 connector
	CTS	X102.2	
	DCD	X102.3	
	RxD	X102.4	
	GND	X102.5	
	TxD	X102.6	
	RTS	X102.7	
	DTR	X102.8	
Ethernet	TxD+	X103.1	RJ-45 connector
	TxD-	X103.2	
	RxD+	X103.3	
	RxD-	X103.6	

CAN0, 1 X100	CAN5V	X100.5A X100.5B	CAN0 and CAN1 are galvanically isolated to on another and to the CPU.
CAN0, 1 X100	CAN0_HIGH	X100.4A	
	CAN1_HIGH	X100.4B	
	N.C.	X100.3A X100.3B	
	CAN0_LOW	X100.2A	
	CAN1_LOW	X100.2B	
Supply voltage VCPU	CAN_GND	X100.1A X100.1B	
	+24VDC	X1.1	
Digital inputs DI0 .. 7 X200	0G	X1.2	Ground CPU
	1G	X200.1A	Ground DI0-3
	DI0	X200.2A	
	DI1	X200.3A	
	DI2	X200.4A	
	DI3	X200.5A	
	2G	X200.1B	Ground DI4-7
	DI4	X200.2B	
	DI5	X200.3B	
	DI6	X200.4B	
Digital inputs DI8 .. 15 X201	DI7	X200.5B	
	3G	X201.1A	Ground DI8-11
	DI8	X201.2A	
	DI9	X201.3A	
	DI10	X201.4A	
	DI11	X201.5A	
	4G	X201.1B	Ground DI12-15
	DI12	X201.2B	
	DI13	X201.3B	
	DI14	X201.4B	
Digital inputs DI16 .. 23 X300	DI15	X201.5B	
	5G	X300.1A	Ground DI16-19
	DI16	X300.2A	
	DI17	X300.3A	
	DI18	X300.4A	
	DI19	X300.5A	
	6G	X300.1B	Ground DI20-23
	DI20	X300.2B	
	DI21	X300.3B	Counter input C3
	DI22	X300.4B	Up/Down control C2
Digital outputs DO0 .. 7 X400	DI23	X300.5B	Up/Down control C0
	7G	X400.1A	Ground VIO
	DO0	X400.2A	
	DO1	X400.3A	
	DO2	X400.4A	
	DO3	X400.5A	
	7G	X400.1B	Ground VIO
	DO4	X400.2B	
	DO5	X400.3B	
DO6	X400.4B		
DO7	X400.5B		

Digital outputs DO8 .. 15 X402	7G	X402.1A	Ground VIO
	DO8	X402.2A	
	DO9	X402.3A	
	DO10	X402.4A	
	DO11	X402.5A	
	7G	X402.1B	Ground VIO
	DO12	X402.2B	
	DO13	X402.3B	
	DO14	X402.4B	
	DO15	X402.5B	
Supply voltage VIO X401	+24VDC	X401.1	
	7G	X401.2	Ground VIO
PWM outputs P0 .. 1 X301	P0	X301.1A	
	VIO	X301.1B	
	P1	X301.2A	
	VIO	X301.2B	
Counter inputs C0 .. C2 X301	C0	X301.3A	
	8G	X301.3B	Ground C0
	C1	X301.4A	
	9G	X301.4B	Ground C1
	C2	X301.5A	
	10G	X301.5B	Ground C2
Analog inputs AI0 .. 3 X600	AI0	X600.1A	
	AGND	X600.1B	analog Ground
	AI1	X600.2A	
	AGND	X600.2B	analog Ground
	AI2	X600.3A	
	AGND	X600.3B	analog Ground
	AI3	X600.4A	
	AGND	X600.4B	analog Ground
Analog outputs AO0 .. 1 X600	AO0	X600.5A	
	AGND	X600.5B	analog Ground
	AO1	X600.6A	
	AGND	X600.6B	analog Ground
Relay contacts REL0 .. 3 X500	REL0_1	X500.1	Closed contact
	REL0_2	X500.2	
	REL1_1	X500.3	Closed contact
	REL1_2	X500.4	
	REL2_1	X500.5	Closed contact
	REL2_2	X500.6	
	REL3_1	X500.7	Closed contact
	REL3_2	X500.8	
	REL3_3 (Closed contact)	X500.9	Open contact

5.2 Power supply (24VDC)

The PLCmodule-C32 features two separate power supply inputs (each 24VDC \pm 20%) for CPU unit ("VCPU") and periphery ("VIO"). Within the PLCmodule-C32, both components are galvanically isolated from one another.

Connector **VCPU** supplies the CPU unit, the analog in- and outputs and activates the Relays. This input has reverse polarity protection.

Connector **VIO** is in charge of supplying Transistor and PWM outputs. This input has reverse polarity protection up to a maximum of I_{VIOmax} .

5.3 In- and outputs for industrial standard signals

5.3.1 Digital inputs DI0 ... DI23 (24VDC)

The PLCmodule-C32 features 24 digital inputs (DI0 ... DI23). The inputs are galvanically isolated in groups of four inputs. Each four inputs have the same supply potential (DI0..3, DI4..7, DI8..11, ...). The inputs are highly active with the following selector shaft:

- Input voltage > 13 VDC: is shown as '1' in the process image
- Input voltage < 5 VDC: is shown as '0' in the process image

Digital inputs DI0 ... DI23 have the internal structure as shown in Figure 5.

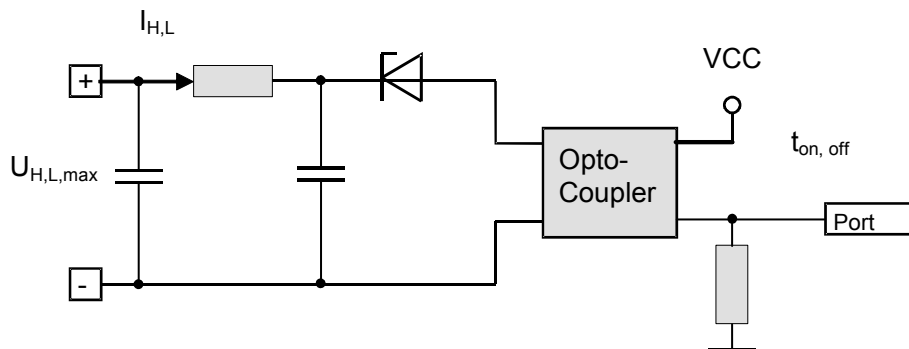


Figure 5: Setup of digital inputs DI0 ... DI23

The digital inputs in a PLC program are accessible via the process image (see Table 4 in section 6.4.1).

5.3.2 Counter inputs C0 ... C2 (24VDC)

The PLCmodule-C32 features 3 high-speed counter inputs (C0 ... C2) that are galvanically isolated from one another and from the CPU kernel. The inputs are highly active with the following selector shafts:

- Input voltage > 13 VDC: is shown as '1' in the process image
- Input voltage < 5 VDC: is shown as '0' in the process image

Counter inputs C0 ... C2 have the same internal structure as the digital inputs DI0 ... DI23 shown in Figure 5.

The counter inputs in a PLC program are accessible via process image (see Table 4 in section 6.4.1) as well as via function block "CNT_FUD" (see section 6.6.1 and manual "SYS TEC-specific extensions for OpenPCS / IEC 61131 3", manual no.: L-1054).

5.3.3 Digital outputs DO0 ... DO15 (24VDC / 0.5A, short-circuit-proof)

The PLCmodule-C32 features 16 digital transistor outputs (DO0 ... DO15). The outputs each connect the supply voltage V_{CC} of the appliance (switching positively). The maximum load current for each 24V output is 0.5A for ohmic, inductive or capacitive load. The outputs are short-circuit-proof and galvanically isolated from the CPU unit. They are supplied with voltage via connector "VIO" (see section 5.2). The performance drivers used are protected against excess voltage, reverse polarity and excess temperature. The transistor outputs are accessed high-actively:

- '1' in process image: output transistor active, appliance connected with V_{CC}
- '0' in process image: output transistor inactive, appliance disconnected from V_{CC}

The digital transistor outputs DO0 ... DO15 have the internal structure as shown in Figure 6.

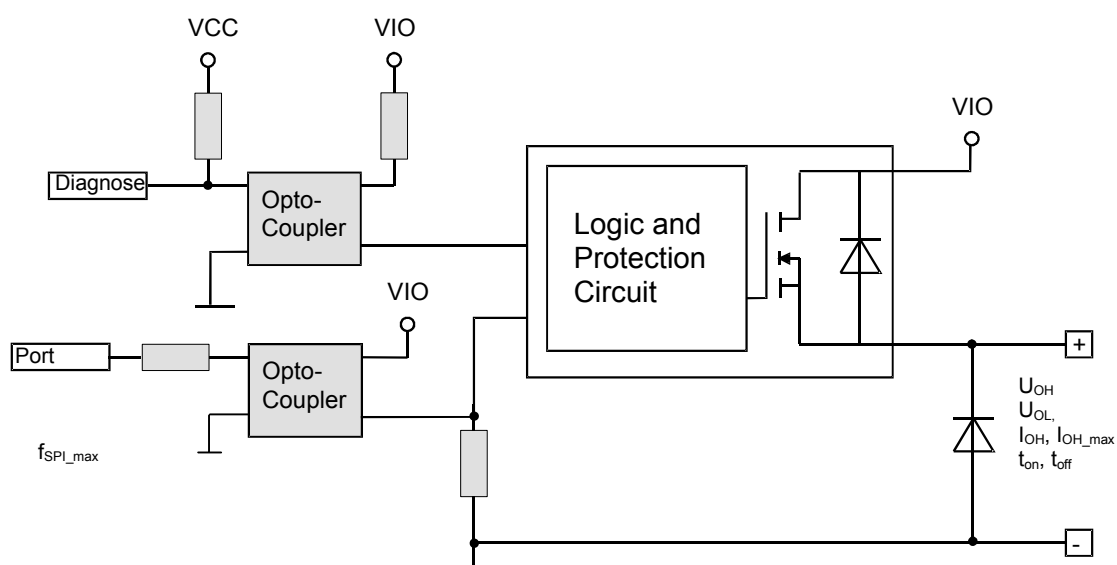


Figure 6: Setup of digital outputs DO0 ... DO15

The digital outputs in a PLC program are accessible via the process image (see Table 4 in section 6.4.1).

5.3.4 Relay outputs REL0 ... REL3 (250VAC / 3A)

The PLCmodule-C32 features 4 Relays outputs. Outputs REL0 ... REL2 are closed contacts and REL3 is a two-way contact. The Relays are activated high-actively:

- '1' in process image: contact is closed
- '0' in process image: contact is open / two-way contact is closed

Relay outputs REL0 ... REL2 have the internal structure as shown in Figure 7.

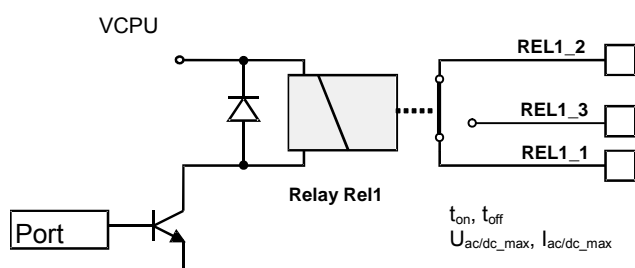


Figure 7: Setup of Relay outputs REL0 ... REL3

Attention! Country-typical technical standards for the usage of power supply voltage must be taken into consideration.

The Relay outputs are in a PLC program are accessible via the process image (see Table 4 in section 6.4.1).

5.3.5 Pulse outputs P0 ... P1 (24VDC / 0.5A, short-circuit-proof)

The PLCmodule-C32 features 2 Pulse outputs (P0 ... P1) to output PWM and PTO signal sequences. The outputs each activate the ground signal (GND) of the connected appliance (switching negatively). The maximum load current for each 24V output is 0.5 A for ohmic, inductive or capacitive load. The outputs are short-circuit-proof and galvanically isolated from the CPU unit. They are supplied with voltage via connector "VIO" (see section 5.2). The outputs have the same ground potential as connection "VIO". The performance drivers used are protected against reverse polarity. The transistor outputs are activated low-actively:

- '1' in process image: output transistor active, appliance connected with GND
- '0' in process image: output transistor inactive, appliance disconnected from GND

Pulse outputs P0 ... P1 have an internal structure as shown in Figure 8.

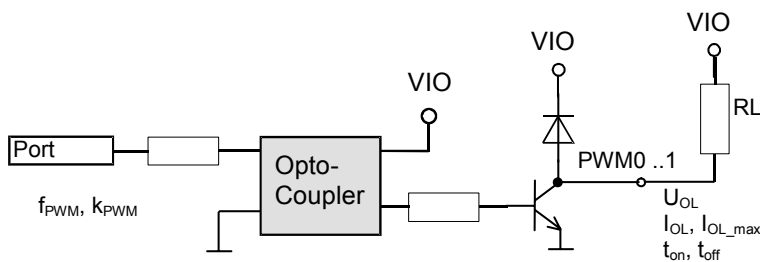


Figure 8: Setup of Pulse outputs P0 ... P1

In a PLC program, the PWM/PTO functionality of Pulse outputs is accessible via function block "PTO_PWM" (see section 6.6.2 as well as manual "SYS TEC-specific extensions for OpenPCS / IEC 61131-3", manual no.: L-1054)

5.3.6 Analog inputs AI0 ... AI3 (0 ... +10V / 0 ... 20mA)

In its standard configuration, the PLCmodule-C32 features 4 analog inputs for a voltage range of 0 ... +10 V and a resolution of 10Bit. Alternatively, voltage inputs can be substituted with current inputs of 0 ... 20 mA. This must be indicated when ordering the module.

Analog inputs AI0 ... AI3 have the same internal structure as shown in Figure 9.

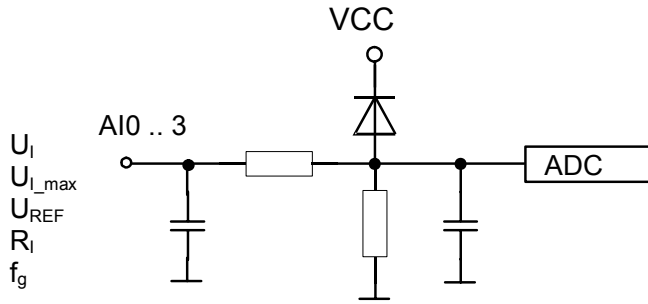


Figure 9: Setup of analog inputs AI0 ... AI3

The analog inputs in a PLC program are accessible via the process image (see Table 4 in section 6.4.1).

5.3.7 Analog outputs AO0 ... AO1 (0 ... +10V)

The PLCmodule-C32 features 2 analog outputs (AO0 ... AO1) for a voltage range of 0 ... +10 V with a resolution of 10Bit. Internally, the analog outputs are realized via PWM channels; each in connection with an active low-pass filter. This allows for a voltage accuracy of $\pm 1\%$.

Analog outputs AO0 ... AO1 have an internal structure as shown in Figure 10.

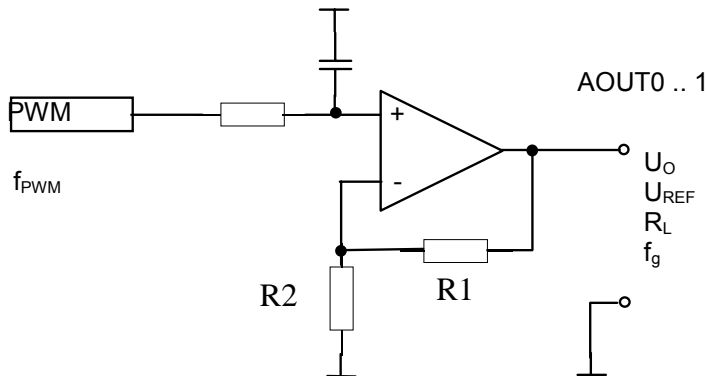


Figure 10: Setup of analog outputs AO0 ... AO1

The analog outputs in a PLC program are accessible via the process image (see Table 4 in section 6.4.1).

5.4 Communication interfaces

5.4.1 Serial interfaces ASC0 ... ASC2

The PLCmodule-C32 features 3 serial interfaces (ASC0 ... ASC2) which are realized as RS-232.

ASC0: Interface ASC0 (also called COM0) primarily serves as service interface to administer the PLCmodule-C32. RS-232 signals RxD, TxD and GND are available at a RJ11 connector. The delivery of the PLCmodule-C32 includes an adapter cable that leads through the signals on a SUB-D9 connector with standard circuit (RS-232 Y cable ASC0+ASC1, order number

162049). If required, it is possible to use a screened RS232 extension cable (plug/plug, 1:1) to connect the PLCmodule-C32 with the Host PC.

ASC1: Interface ASC1 (also called COM1) is disposable and usually used for data exchange between the PLCmodule-C32 and other field devices under control of the PLC program. Signals RxD, TxD and GND are available at a RJ11 connector. The delivery of the PLCmodule-C32 includes an adapter cable which leads through the signals on a SUB-D9 connector with standard circuit (RS-232 Y cable ASC0+ASC1, order number 162049). If required, it is possible to use a shielded null modem cable (plug/connector) to connect the PLCmodule-C32 with the Host PC.

ASC2: Interface ASC2 (also called COM2) is disposable and **modem-compliant**. It is normally used for data exchange between the PLCmodule-C32 and other field devices under control of the PLC program. Signals RxD, TxD, DTR, DCD, RTS, CTS, DSR and GND are available at a RJ45 connector. The delivery of the PLCmodule-C32 includes an adapter cable which leads through signals on a SUB-D9 connector with standard circuit (RS-232 modem cable, order number 162052). If required, it is possible to use a screened null modem cable (plug/connector) to connect the PLCmodule-C32 with the Host PC.

5.4.2 CAN interfaces CAN0 ... CAN1

The PLCmodule-C32 features 2 CAN interfaces (CAN0 ... CAN1). Those two CAN-Bus-Transceivers (PHILIPS PCA82C251) are galvanically isolated to one another and to the CPU. The transceivers are supplied via two on-board DC/DC converter. CAN-Bus signals CAN0_HIGH, CAN0_LOW, CAN1_HIGH, CAN1_LOW and CAN_GND are available from withdrawable screw-clamping connectors.

Section 6.9 provides detailed information about the usage of both CAN interfaces in connection with CANopen.

CAN cable: The CAN-Bus usually is a twisted pair line. At both ends of the cable, a termination resistor of **120 Ohm termination** is necessary **between CAN_H and CAN_L**. CiA (CAN in Automation) suggests using CAN-GND in CiA DRP 303-1. For more information please refer to the appropriate CiA standards.

5.4.3 Ethernet interface ETH0

The PLCmodule-C32 features one Ethernet interface (ETH0, also called ETHERNET) which is designed as 10Base-T/100Base-TX. The connection takes place via a standard RJ-45 connector.

The Ethernet interface serves as service interface to administer the PLCmodule-C32 and it can be used for data exchange with any other devices.

6 PLC functionality of the PLCmodule-C32

6.1 Overview

The PLCmodule-C32 is provided with the SYS TEC *PLCcore-5484*, a complete Linux based compact PLC as plug-in "core"-module. The PLCmodule-C32 adapt all inputs and outputs of the PLCcore-5484 to industrial standard level (24VDC, 0-10VDC, 0-20mA etc.) and completes the core-module with a industrial suited DIN rail enclosure.

Both modules, PLCmodule-C32 and PLCcore-5484, are using the same Embedded Linux as operating system. So configuration and programming of PLCmodule-C32 is nearly identical to PLCcore-5484.

6.2 System start of the PLCmodule-C32

By default, the PLCmodule-C32 loads all necessary firmware components upon Power-on or Reset and starts running the PLC program afterwards. Hence, the PLCmodule-C32 is suitable for the usage in autarchic control systems. In case of power breakdown, such systems resume the execution of the PLC program independently and without user intervention. Figure 11 shows the system start in detail:

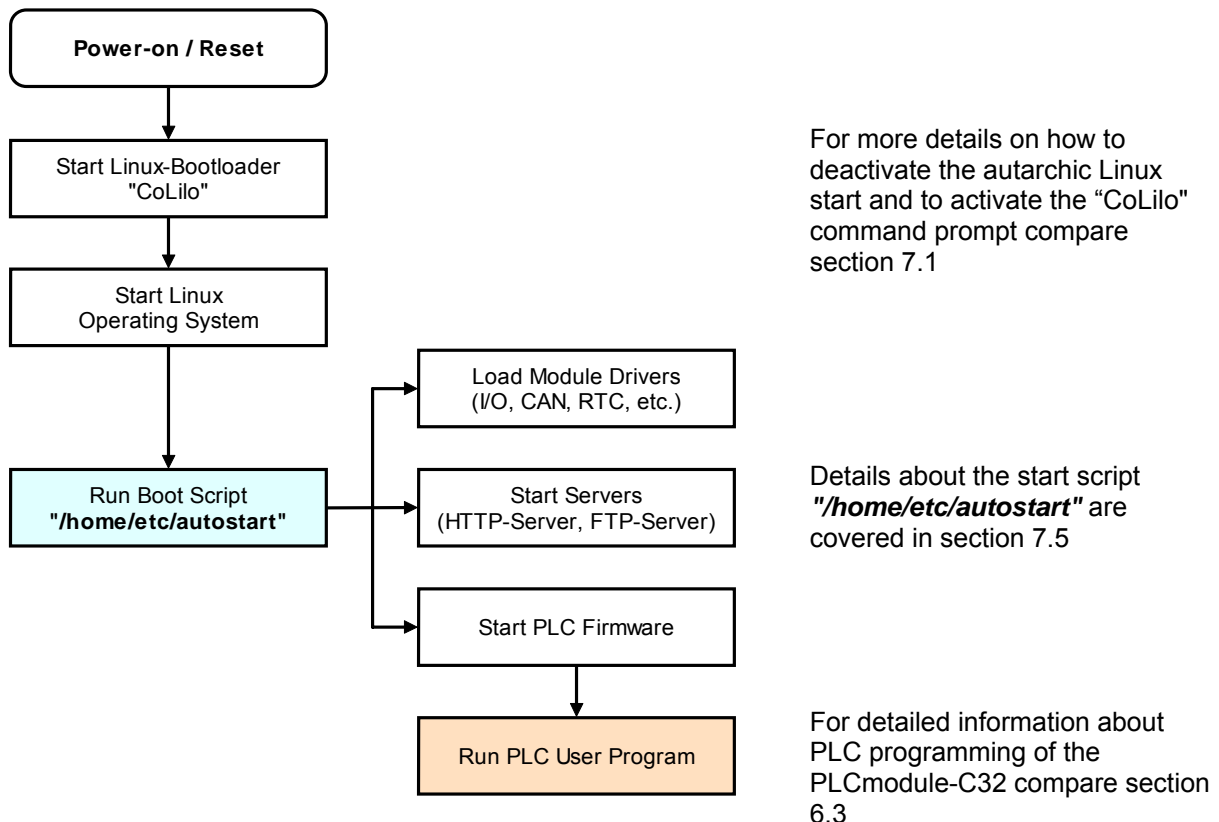


Figure 11: System start of the PLCmodule-C32

6.3 Programming the PLCmodule-C32

The PLCmodule-C32 is programmed with IEC 61131-3-conform *OpenPCS* programming environment. There exist additional manuals about *OpenPCS* that describe the handling of this programming tool. Those are part of the software package "*OpenPCS*". All manuals relevant for the PLCmodule-C32 are listed in Table 1.

PLCmodule-C32 firmware is based on standard firmware for SYS TEC's compact control units. Consequently, it shows identical properties like other SYS TEC control systems. This affects especially the process image setup (see section 6.4) as well as the functionality of control elements (Hex-Encoding switch, DIP-Switch, Run/Stop switch, Run-LED, Error-LED).

Depending on the firmware version used, PLCmodule-C32 firmware provides numerous function blocks to the user to access communication interfaces. Table 3 specifies the availability of FB communication classes (SIO, CAN, UDP) for different PLCmodule-C32 firmware versions. Section 7.6 describes the selection of the appropriate firmware version.

Table 3: Support of FB communication classes for different types of the PLCmodule

Type of Interface	PLCmodule-C32/Z3 Art. no.: 3090000	PLCmodule-C32/Z4 Art. no.: 3090001	PLCmodule-C32/Z5 Art. no.: 3090002	Remark
CAN	-	x	x	FB description see manual L-1008
UDP	-	x	x	FB description see manual L-1054
SIO	x	x	x	FB description see manual L-1054

Table 20 in Appendix A contains a complete listing of firmware functions and function blocks that are supported by the PLCmodule-C32.

Detailed information about using the CAN interfaces in connection with CANopen is provided in section 6.9.

6.4 Process image of the PLCmodule-C32

6.4.1 Local In- and Outputs

Compared to other SYS TEC compact control systems, the PLCmodule-C32 obtains a process image with identical addresses. All in- and outputs listed in Table 4 are supported by the PLCmodule-C32.

Table 4: Assignment of in- and outputs to the process image of the PLCmodule-C32

I/O of the PLCmodule-C32	Address and Data type in the Process Image
DI0 ... DI7	%IB0.0 as Byte with DI0 ... DI7 %IX0.0 ... %IX0.7 as single Bit for each input
DI8 ... DI15	%IB1.0 as Byte with DI8 ... DI15 %IX1.0 ... %IX1.7 as single Bit for each input
DI16 ... DI23	%IB2.0 as Byte with DI16 ... DI23 %IX2.0 ... %IX2.7 as single Bit for each input
AI0	%IW8.0 15Bit + sign (0 ... +32767)
AI1	%IW10.0 15Bit + sign (0 ... +32767)
AI2	%IW12.0 15Bit + sign (0 ... +32767)
AI3	%IW14.0 15Bit + sign (0 ... +32767)
C0	%ID40.0 31Bit + sign ($-2^{31} - 2^{31} - 1$) counter input: DI24 (%IX3.0), direction: DI21 (%IX2.5), see section 6.6.1
C1	%ID44.0 31Bit + sign ($-2^{31} - 2^{31} - 1$) counter input: DI25 (%IX3.1), direction: DI22 (%IX2.6), see section 6.6.1
C2	%ID48.0 31Bit + sign ($-2^{31} - 2^{31} - 1$) counter input: DI26 (%IX3.2), direction: DI23 (%IX2.7), see section 6.6.1
On-board Temperature Sensor, see ⁽¹⁾	%ID72.0 31Bit + sign as 1/10000 °C
DO0 ... DO7	%QB0.0 as Byte with DO0 ... DO7 %QX0.0 ... %QX0.7 as single Bit for each output
DO8 ... DO15	%QB1.0 as Byte with DO8 ... DO15 %QX1.0 ... %QX1.7 as single Bit for each output
REL0 ... REL3 (corresponds to DO16 ... DO19)	%QB2.0 as Byte with REL0 ... REL3 %QX2.0 ... %QX2.3 as single Bit for each Relay
P0	%QX2.4 (default value for inactive generator) Impulse output: DO20, see section 6.6.2
P1	%QX2.5 (default value for inactive generator) Impulse output: DO21, see section 6.6.2
AO0	%QW8.0 15Bit + sign (0 ... +32767)
AO1	%QW10.0 15Bit + sign (0 ... +32767)

⁽¹⁾ This marked components are only available in the process image, if the **Option "Enable extended I/O's"** is activated within the PLC configuration (see section 7.4.1).
Alternatively, entry "EnableExtIo=" can directly be set within section "[Proclmg]" of the

configuration file *"/home/plc/plcmodule-c32.cfg"* (see section 7.4.3). The appropriate configuration setting is evaluated upon start of the PLC firmware.

Advice: The PLCmodule-C32 works with Big-Endian format ("Motorola-Notation"). Consequently and on the contrary to controls using Little-Endian format ("Intel-Notation"), it is **not possible** to sum up several BYTE variables of the process image to one WORD or DWORD and to access Bits above Bit7. The following example shows issue described:

```
bInByte0 AT %IB0.0 : BYTE;
bInByte1 AT %IB1.0 : BYTE;
wInWord  AT %IW0.0 : BYTE;

wInWord.0 <> bInByte0.0      due to Big-Endian: wInWord.0 == bInByte1.0
wInWord.8 <> bInByte1.0      due to Big-Endian: wInWord.8 == bInByte0.0
```

In- and outputs of the PLCmodule-C32 are not negated in the process image. Hence, the H-level at one input leads to value "1" at the corresponding address in the process image. Contrariwise, value "1" in the process image leads to an H-level at the appropriate output.

6.4.2 Network variables for CAN1

Contrary to interface CAN0, interface CAN1 of the PLCmodule-C32 is designed as static object dictionary. Thus, at interface CAN1 the PLCmodule-C32 acts as a CANopen I/O device. All static network variables for CAN1 are accessible via the marker section of the process image.

Section 6.9.2 includes more detailed information about CAN interface CAN1 and the network variables that are provided by it in the marker section.

6.5 Communication interfaces

6.5.1 Serial interfaces

The PLCmodule-C32 features 3 serial interfaces (ASC0 ... ASC2) that function as RS-232. Details about hardware activation are included in section 5.4.1.

ASC0: Interface ASC0 (also termed as COM0) primarily serves as service interface to administer the PLCmodule-C32. By default, in boot script *"/etc/inittab"* it is assigned to the Linux process *"getty"* and is used as Linux console to administer the PLCmodule-C32. Even though interface ASC0 may be used from a PLC program via function blocks of type *"SIO_Xxx"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054), only signs should be output in this regard. The module tries to interpret and to execute signs that it receives as Linux commands.

To freely use an interface from a PLC program, boot script *"/etc/inittab"* must be adjusted appropriately which is only possible by modifying the Linux image. This requires software package SO-1095 ("VMware-Image of the Linux Development System for the ECUcore-5484").

ASC1: Interface ASC1 (also termed as COM1) is disposable and support data exchange between the PLCmodule-C32 and other field devices kept under control of the PLC program.

Interface ASC1 may be used from a PLC program via function blocks of type *"SIO_Xxx"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054).

ASC2: Interface ASC2 (also termed as COM2) is disposable and **modem-compatible**. It supports data exchange between the PLCmodule-C32 and other field devices kept under control of the PLC program.

Interface ASC2 may be used from a PLC program via function blocks of type "SIO_Xxx" (see manual "SYS TEC-specific Extensions for OpenPCS / IEC 61131-3", Manual no.: L-1054).

6.5.2 CAN interfaces

The PLCmodule-C32 features 2 CAN interfaces (CAN0 ... CAN1). Details about the hardware activation are included in section 5.4.2.

Both CAN interfaces allow for data exchange with other devices via network variables and they are accessible from a PLC program via function blocks of type "CAN_Xxx" (see section 6.9 and "User Manual CANopen Extension for IEC 61131-3", Manual no.: L-1008).

Section 6.9 provides detailed information about the usage of the CAN interfaces in connection with CANopen.

6.5.3 Ethernet interface

The PLCmodule-C32 features 1 Ethernet interface (ETH0, also termed as ETHERNET). Details about the hardware activation are included in section 5.4.3

The Ethernet interface serves as service interface to administer the PLCmodule-C32 and it enables data exchange with other devices. The interface is accessible from a PLC program via function blocks of type "LAN_Xxx" (see manual "SYS TEC-specific Extensions for OpenPCS / IEC 61131-3", Manual no.: L-1054).

The exemplary PLC program "UdpRemoteCtrl" illustrates the usage of function blocks of type "LAN_Xxx" within a PLC program.

6.6 Specific peripheral interfaces

6.6.1 Counter inputs

The PLCmodule-C32 features 3 fast counter input (C0 ... C2). Prior to its usage, all counter inputs must be parameterized via function block "CNT_FUD" (see manual "SYS TEC-specific Extensions for OpenPCS / IEC 61131 3", Manual no.: L 1054). Afterwards, in a PLC program the current counter value is accessible via the process image (see Table 4 in section 6.4.1) or via function block "CNT_FUD". Table 5 lists the allocation between counter channels and inputs.

Table 5: Allocation between counter channels and inputs

Counter channel	Counter input	Optional direction input	Counter value in process image
C0	C0 (DI24) %IX3.0	DI21 %IX2.5	%ID40.0
C1	C1 (DI25) %IX3.1	DI22 %IX2.6	%ID44.0
C2	C2 (DI26) %IX3.2	DI23 %IX2.7	%ID48.0

6.6.2 Pulse outputs

To release PWM and PTO signal sequences, the PLCmodule-C32 features 2 pulse outputs (P0 ... P1). Prior to its usage, all pulse outputs must be parameterized using function block "PTO_PWM" (see manual "SYS TEC-specific Extensions for OpenPCS / IEC 61131 3", Manual no.: L 1054). After the impulse generator is started, it takes over the control of respective outputs. After the impulse generator is deactivated, the respective output adopts the corresponding value that is filed in the process image for this output (see Table 4 in section 6.4.1). Table 6 lists the allocations between impulse channels and outputs.

Table 6: Allocation between impulse channels and outputs

Impulse channel	Impulse output
P0	P0 (DO21) %QX2.4
P1	P1 (DO22) %QX2.5

6.7 Control and display elements

6.7.1 Run/Stop switch

The Run/Stop switch makes it possible to start and interrupt the execution of the PLC program. Together with start and stop pushbuttons of the *OpenPCS* programming environment, the Run/Stop switch represents a "logical" AND-relation. This means that the PLC program will not start the execution until the local Run/Stop switch is positioned to "Run" **AND** additionally the start command (cold, warm or hot start) is given by the *OpenPCS* user interface. The order hereby is not relevant. A run command given by *OpenPCS* while at the same time the Run/Stop switch is positioned to "Stop" is visible through quick flashing of the Run-LED (green).

Positioned to "MRes" ("Modul Reset"), the Run/Stop switch allows for local deletion of a PLC program from the PLCmodule-C32. This might for example be necessary if an error occurs and the PLC program is running an infinite loop and consequently, accessing the *OpenPCS* programming environment is no longer possible. The procedure for deleting a PLC program is described in section 6.8.

6.7.2 Run-LED (green)

The Run-LED provides information about the activity state of the control system. The activity state is shown through different modes:

Table 7: Display status of the Run-LED

LED Mode	PLC Activity State
Off	The PLC is in state "Stop": <ul style="list-style-type: none"> the PLC does not have a valid program, the PLC has received a stop command from the <i>OpenPCS</i> programming environment or the execution of the program has been canceled due to an internal error
Quick flashing in relation 1:8 to pulse	The PLC is on standby but is not yet executing: <ul style="list-style-type: none"> The PLC has received a start command from the <i>OpenPCS</i> programming environment but the local Run/Stop switch is still positioned to "Stop"
Slow flashing in relation 1:1 to pulse	The PLC is in state "Run" and executes the PLC program.
Quick flashing in relation 1:1 to pulse	The PLC is in mode "Reset", compare section 6.8

6.7.3 Error-LED (red)

The Error-LED provides information about the error state of the control system. The error state is represented through different modes:

Table 8: Display status of the Error-LED

LED Mode	PLC Error State
Off	No error has occurred; the PLC is in normal state.
Permanent light	A severe error has occurred: <ul style="list-style-type: none"> The PLC was started using an invalid configuration (e.g. CAN node address 0x00) and had to be stopped or A severe error occurred during the execution of the program and caused the PLC to independently stop its state "Run" (division by zero, invalid Array access, ...), see below
Slow flashing in relation 1:1 to pulse	A network error occurred during communication to the programming system; the execution of a running program is continued. This error state will be reset independently by the PLC as soon as further communication to the programming system is successful.
Quick flashing in relation 1:1 to pulse	The PLC is in mode "Reset", compare section 6.8
Quick flashing in relation 1:8 to pulse	The PLC is on standby, but is not yet running: <ul style="list-style-type: none"> The PLC has received a start command from the <i>OpenPCS</i> programming environment but the local Run/Stop switch is positioned to "Stop"

In case of severe system errors such as division by zero or invalid Array access, the control system passes itself from state "Run" into state "Stop". This is recognizable by the permanent light of the

Error-LED (red). In this case, the error cause is saved by the PLC and is transferred to the computer and shown upon next power-on.

6.8 Local deletion of a PLC program

If the Run/Stop switch is positioned to "*MRes*" ("*Modul Reset*") (see section 6.7.1), it is possible to delete a program from the PLCmodule-C32. This might for example be necessary if an error occurs and the PLC program is running an infinite loop and consequently, accessing the *OpenPCS* programming environment is no longer possible. To prevent deleting a PLC program by mistake, it is necessary to keep to the following order:

- (1) Position the Run/Stop switch to "*MRes*"
- (2) Reset the PLCmodule-C32 (by pressing the reset pushbutton on the top of the module or through temporary power interrupt)
 - ⇒ Run-LED (green) is flashing quickly in relation 1:1 to the pulse
- (3) Position the Run/Stop switch to "*Run*"
 - ⇒ Error-LED (red) is flashing quickly in relation 1:1 to the pulse
- (4) Reposition Run/Stop switch back to "*MRes*" **within 2 seconds**
 - ⇒ PLCmodule-C32 is deleting PLC program
 - ⇒ Run-LED (green) and Error-LED (red) are both flashing alternately
- (5) Reposition Run/Stop switch to "*Stop*" or "*Run*" and reset again to start the PLCmodule-C32 and bring it into normal working state

If Reset of the PLCmodule-C32 is activated (e.g. through temporary power interrupt) while at the same time the Run/Stop switch is positioned to "*MRes*", the module recognizes a reset requirement. This is visible through quick flashing of the Run-LED (green). This mode can be stopped without risk. Therefore, the Run/Stop switch must be positioned to "*Run*" or "*Stop*" (Error-LED is flashing) and it must be waited for 2 seconds. The PLCmodule-C32 independently stops the reset process after 2 seconds and starts a normal working state with the PLC program which was saved last.

6.9 Using CANopen for CAN interfaces

The PLCmodule-C32 features 2 CAN interfaces (CAN0 ... CAN1), both are usable as CANopen Manager (conform to CiA Draft Standard 302). The configuration of both interfaces (active/inactive, node number, Bittate, Master on/off) is described in section 7.4.

Both CAN interfaces allow for data exchange with other devices via network variables and they are usable from a PLC program via function blocks of type "*CAN_Xxx*". More details are included in "*User Manual CANopen Extension for IEC 61131-3*", Manual no.: L-1008.

The CANopen services **PDO** (**P**rocess **D**ata **O**bjects) and **SDO** (**S**ervice **D**ata **O**bjects) are two separate mechanisms for data exchange between single field bus devices. Process data sent from a node (**PDO**) are available as broadcast to interested receivers. PDOs are limited to 1 CAN telegram and therewith to 8 Byte user data maximum because PDOs are executed as non-receipt broadcast messages. On the contrary, **SDO** transfers are based on logical point-to-point connections ("Peer to Peer") between two nodes and allow the receipted exchange of data packages that may be larger than 8 Bytes. Those data packages are transferred internally via an appropriate amount of CAN telegrams. Both services are applicable for interface CAN0 as well as for CAN1 of the PLCmodule-C32.

SDO communication basically takes place via function blocks of type "CAN_SDO_Xxx" (see "User Manual CANopen Extension for IEC 61131-3", Manual no.: L-1008). Function blocks are also available for PDOs ("CAN_PDO_Xxx"). Those should only be used for particular cases in order to also activate non-CANopen-conform devices. For the application of PDO function blocks, the CANopen configuration must be known in detail. The reason for this is that the PDO function blocks only use 8 Bytes as input/output parameter, but the assignment of those Bytes to process data is subject to the user.

Instead of PDO function blocks, network variables should mainly be used for PDO-based data exchange. Network variables represent the easiest way of data exchange with other CANopen nodes. Accessing network variables within a PLC program takes place in the same way as accessing internal, local variables of the PLC. Hence, for PLC programmers it is not of importance if e.g. an input variable is allocated to a local input of the control or if it represents the input of a decentralized extension module. The application of network variables is based on the integration of DCF files that are generated by an appropriate CANopen configurator. On the one hand, DCF files describe communication parameters of any device (CAN Identifier, etc.) and on the other hand, they allocate network variables to the Bytes of a CAN telegram (mapping). The application of network variables only requires basic knowledge about CANopen.

For the PLCmodule-C32, the usage of PDO-based network variables is different for each CAN interface CAN0 and CAN1. Sections 6.9.1 and 6.9.2 provide more detail on this.

In a CANopen network, exchanging PDOs only takes place in status "OPERATIONAL". If the PLCmodule-C32 is not in this status, it does not process PDOs (neither for send-site nor for receive-site) and consequently, it does not update the content of network variables. The CANopen Manager is in charge of setting the operational status "OPERATIONAL", "PRE-OPERATIONAL" etc. (mostly also called "CANopen Master"). In typical CANopen networks, a programmable node in the form of a PLC is used as CANopen-Manager. The PLCmodule-C32 is able to take over tasks of the CANopen Manager at both CAN interfaces CAN0 and CAN1. How the Manager is activated is described in section 7.4.

As CANopen Manager, the PLCmodule-C32 is able to parameterize the CANopen I/O devices ("CANopen-Slaves") that are connected to the CAN bus. Therefore, upon system start via SDO it transfers DCF files generated by the CANopen configurator to the respective nodes.

6.9.1 CAN interface CAN0

Interface CAN0 features a dynamic object dictionary. This implicates that after activating the PLC, the interface does not provide communication objects for data exchange with other devices. After downloading a PLC program (or its reload from the non-volatile storage after power-on), the required communication objects are dynamically generated according to the DCF file which is integrated in the PLC project. Thus, CAN interface CAN0 is extremely flexible and also applicable for larger amount of data.

For the PLC program, all network variables are declared as "VAR_EXTERNAL" according to IEC61131-3. Hence, they are marked as „outside of the control“, e.g.:

```
VAR_EXTERNAL
  NetVar1 : BYTE ;
  NetVar2 : UINT ;
END_VAR
```

A detailed procedure about the integration of DCF files into the PLC project and about the declaration of network variables is provided in manual "User Manual CANopen Extension for IEC 61131-3" (Manual no.: L-1008).

When using CAN interface CAN0 it must be paid attention that the generation of required objects takes place upon each system start. This is due to the dynamic object directory. "Design instructions"

are included in the DCF file that is integrated in the PLC project. **Hence, changes to the configuration can only be made by modifying the DCF file.** This implies that after the network configuration is changed (modification of DCF file), the PLC project must again be translated and loaded onto the PLCmodule-C32.

6.9.2 CAN interface CAN1

On the contrary to interface CAN0, interface CAN1 is provided as static object dictionary. This means that the amount of network variables (communication objects) and the amount of PDOs available are both strongly specified. During runtime, the configuration of PDOs is modifiable. This implies that communication parameters used (CAN Identifier, etc.) and the allocation of network variables to each Byte of a CAN telegram (mapping), can be set and modified by the user. Thus, only the amount of objects (amount of network variables and PDOs) is strongly specified in the static object dictionary. Consequently, application and characteristics of objects can be modified during runtime. For this reason, at interface CAN1 the PLCmodule-C32 acts as a CANopen I/O device.

All network variables of the PLC program are available through the marker section of the process image. Therefore, 252 Bytes are usable as input variables and also 252 Bytes as output variables. To enable any data exchange with other CANopen I/O devices, the section of static network variables is mapped to different data types in the object dictionary (BYTE, SINT, WORD, INT, DWORD, DINT). Variables of the different data types are located within the same memory area which means that all variables represent the same physical storage location. Hence, a WORD variable interferes with 2 BYTE variables, a DWORD variable with 2 WORD or 4 BYTE variables. Figure 12 the positioning of network variables for CAN1 within the marker section.

		CAN1 Input Variables																
		CAN1 IN0	CAN1 IN1	CAN1 IN2	CAN1 IN3	CAN1 IN4	CAN1 IN5	CAN1 IN6	CAN1 IN7	...	CAN1 IN244	CAN1 IN245	CAN1 IN246	CAN1 IN247	CAN1 IN248	CAN1 IN249	CAN1 IN250	CAN1 IN251
BYTE / SINT, USINT		%MB 0.0 (Byte0)	%MB 1.0 (Byte1)	%MB 2.0 (Byte2)	%MB 3.0 (Byte3)	%MB 4.0 (Byte4)	%MB 5.0 (Byte5)	%MB 6.0 (Byte6)	%MB 7.0 (Byte7)	...	%MB 244.0 (Byte244)	%MB 245.0 (Byte245)	%MB 246.0 (Byte246)	%MB 247.0 (Byte247)	%MB 248.0 (Byte248)	%MB 249.0 (Byte249)	%MB 250.0 (Byte250)	%MB 251.0 (Byte251)
WORD / INT, UINT		%MW 0.0 (Word0)		%MW 2.0 (Word1)		%MW 4.0 (Word2)		%MW 6.0 (Word3)		...	%MW 244.0 (Word122)		%MW 246.0 (Word123)		%MW 248.0 (Word124)		%MW 250.0 (Word125)	
DWORD / DINT, UDINT		%MD 0.0 (Dw ord0)				%MD 4.0 (Dw ord1)				...	%MD 244.0 (Dw ord61)				%MD 248.0 (Dw ord62)			

		CAN1 Output Variables																
		CAN1 OUT0	CAN1 OUT1	CAN1 OUT2	CAN1 OUT3	CAN1 OUT4	CAN1 OUT5	CAN1 OUT6	CAN1 OUT7	...	CAN1 OUT244	CAN1 OUT245	CAN1 OUT246	CAN1 OUT247	CAN1 OUT248	CAN1 OUT249	CAN1 OUT250	CAN1 OUT251
BYTE / SINT, USINT		%MB 256.0 (Byte0)	%MB 257.0 (Byte1)	%MB 258.0 (Byte2)	%MB 259.0 (Byte3)	%MB 260.0 (Byte4)	%MB 261.0 (Byte5)	%MB 262.0 (Byte6)	%MB 263.0 (Byte7)	...	%MB 500.0 (Byte244)	%MB 501.0 (Byte245)	%MB 502.0 (Byte246)	%MB 503.0 (Byte247)	%MB 504.0 (Byte248)	%MB 505.0 (Byte249)	%MB 506.0 (Byte250)	%MB 507.0 (Byte251)
WORD / INT, UINT		%MW 256.0 (Word0)		%MW 258.0 (Word1)		%MW 260.0 (Word2)		%MW 262.0 (Word3)		...	%MW 500.0 (Word122)		%MW 502.0 (Word123)		%MW 504.0 (Word124)		%MW 506.0 (Word125)	
DWORD / DINT, UDINT		%MD 265.0 (Dw ord0)				%MD 260.0 (Dw ord1)				...	%MD 500.0 (Dw ord61)				%MD 504.0 (Dw ord62)			

Figure 12: Positioning of network variables for CAN1 within the marker section

Table 9 shows the representation of network variables through appropriate inputs in the object dictionary of interface CAN0.

Table 9: Repräsentation der Netzwerkvariablen für CAN1 durch Einträge im Object-Dictionary

OD section	OD variable / EDS input	Data type CANopen	Data type IEC 61131-3
<i>Inputs (inputs for the PLCmodule-C32)</i>			
Index 2000H Sub 1 ... 252	CAN1InByte0 ... CAN1InByte251	Unsigned8	BYTE, USINT
Index 2001H Sub 1 ... 252	CAN1InSint0 ... CAN1InSint251	Integer8	SINT
Index 2010H Sub 1 ... 126	CAN1InWord0 ... CAN1InWord125	Unsigned16	WORD, UINT
Index 2011H Sub 1 ... 126	CAN1InInt0 ... CAN1InInt125	Integer16	INT
Index 2020H Sub 1 ... 63	CAN1InDword0 ... CAN1InDword62	Unsigned32	DWORD, UDINT
Index 2021H Sub 1 ... 63	CAN1InDint0 ... CAN1InDint62	Integer32	DINT
<i>Outputs (outputs for the PLCmodule-C32)</i>			
Index 2030H Sub 1 ... 252	CAN1OutByte0 ... CAN1OutByte251	Unsigned8	BYTE, USINT
Index 2031H Sub 1 ... 252	CAN1OutSint0 ... CAN1OutSint251	Integer8	SINT
Index 2040H Sub 1 ... 126	CAN1OutWord0 ... CAN1OutWord125	Unsigned16	WORD, UINT
Index 2041H Sub 1 ... 126	CAN1OutInt0 ... CAN1OutInt125	Integer16	INT
Index 2050H Sub 1 ... 63	CAN1OutDword0 ... CAN1OutDword62	Unsigned32	DWORD, UDINT
Index 2051H Sub 1 ... 63	CAN1OutDint0 ... CAN1OutDint62	Integer32	DINT

The object dictionary of interface CAN0 in total has available 16 TPDO and 16 RPDO. The first 4 TPDO and RPDO are preconfigured and activated according to the Predefined Connection Set. The first 32 Byte of input and output variables are mapped to those PDOs. Table 10 in detail lists all preconfigured PDOs for interface CAN1.

Table 10: Preconfigured PDOs for interface CAN1

PDO	CAN-ID	Data
1. RPDO	0x200 + NodeID	%MB0.0 ... %MB7.0
2. RPDO	0x300 + NodeID	%MB8.0 ... %MB15.0
3. RPDO	0x400 + NodeID	%MB16.0 ... %MB23.0
4. RPDO	0x500 + NodeID	%MB24.0 ... %MB31.0
1. TPDO	0x180 + NodeID	%MB256.0 ... %MB263.0
2. TPDO	0x280 + NodeID	%MB264.0 ... %MB271.0
3. TPDO	0x380 + NodeID	%MB272.0 ... %MB279.0
4. TPDO	0x480 + NodeID	%MB280.0 ... %MB287.0

Due to limitation to 16 TPDO and 16 RPDO, only 256 Bytes (2 * 16PDO * 8Byte/PDO) of total 504 Bytes for network variables in the marker section (2 252Bytes) can be transferred via PDO. Irrespective of that it is possible to access all variables via SDO.

The configuration (mapping, CAN Identifier etc.) of interface CAN1 typically takes place via an external Configuration Manager that parameterizes the object dictionary on the basis of a DCF file created by the CANopen configurator. By using default object inputs 1010H und 1011H, the PLCmodule-C32 supports the persistent storage and reload of a backed configuration.

Alternatively, the configuration (mapping, CAN Identifier etc.) of the static object dictionary for interface CAN1 can take place from the PLC program by using SDO function blocks. Therefore, inputs *NETNUMBER* and *DEVICE* must be used as follows:

```
NETNUMBER := 1;          (* Interface CAN1 *)
DEVICE    := 0;          (* local Node   *)
```

The PLC program example "*ConfigCAN1*" exemplifies the configuration of interface CAN0 through a PLC program by using function blocks of type "*CAN_SDO_Xxx*".

7 Configuration and Administration of the PLCmodule-C32

7.1 System requirements and necessary software tools

The administration of the PLCmodule-C32 requires any Windows or Linux computer that has available an Ethernet interface and a serial interface (RS232). As alternative solution to the on-board serial interface, SYS TEC offers a USB-RS232 Adapter Cable (order number 3234000, see section 4.2) that provides an appropriate RS232 interface via USB port.

All examples referred to in this manual are based on an administration of the PLCmodule-C32 using a Windows computer. Procedures using a Linux computer would be analogous.

To administrate the PLCmodule-C32 the following software tools are necessary:

Terminal program A Terminal program allows the communication with the **command shell** of the PLCmodule-C32 via a **serial RS232 connection to ASC0 (resp. COM0) of the PLCmodule-C32**. This is required for the Ethernet configuration of the PLCmodule-C32 as described in section 7.3. After completing the Ethernet configuration, all further commands can either be entered in the Terminal program or alternatively in a Telnet client (see below).

Suitable as Terminal program would be "*HyperTerminal*" which is included in the Windows delivery or "*TeraTerm*" which is available as Open Source and meets higher demands (downloadable from: <http://tssh2.sourceforge.jp>).

Telnet client Telnet-Client allows the communication with **command shell** of the PLCmodule-C32 via **Ethernet connection to ETH0 of the PLCmodule-C32**. Using Telnet clients requires a completed Ethernet configuration of the PLCmodule-C32 according to section 7.3. As alternative solution to Telnet client, all commands can be edited via a Terminal program (to ASC0 resp. COM0 of the PLCmodule-C32).

Suitable as Telnet client would be "*Telnet*" which is included in the Windows delivery or "*TeraTerm*" which can also be used as Terminal program (see above).

FTP client An FTP client allows for file exchange between the PLCmodule-C32 (ETH0) and the computer. This allows for example **editing configuration files** by transferring those from the PLCmodule-C32 onto the computer where they can be edited and get transferred back to the PLCmodule-C32. Downloading files onto the PLCmodule-C32 is also necessary to **update the PLC firmware**. (Advice: The update of *PLC firmware* is not identical with the update of the *PLC user program*. The PLC program is directly transferred to the module from the *OpenPCS* programming environment. No additional software is needed for that.)

Suitable as FTP client would be "*WinSCP*" which is available as Open Source (download from: <http://winscp.net>). It only consists of one EXE file that needs no installation and can be booted immediately. Furthermore, freeware "*Core FTP LE*" (downloadable from: <http://www.coreftp.com>) or "*Total Commander*" (integrated in the file manager) are suitable as FTP client.

TFTP server

The TFTP server is necessary to update the Linux-Image on the PLCmodule-C32. Freeware "TFTPD32" (download from: <http://tftpd32.jounin.net>) is suitable as TFTP server. It only consists of one EXE file that needs no installation and can be booted immediately.

For programs that communicate via Ethernet interface, such as FTP client or TFTP server, it must be paid attention to that rights in the Windows-Firewall are released. Usually Firewalls signal when a program seeks access to the network and asks if this access should be permitted or denied. In this case access is to be permitted.

7.2 Activation/Deactivation of Linux Autostart

During standard operation mode, the bootloader "CoLilo" automatically starts the Linux operating system of the module after Reset (or Power-on). Afterwards, the operating system loads all further software components and controls the PLC program execution (see section 6.2). For service purposes, such as configuring the Ethernet interface (see section 7.3) or updating the Linux-Image (see section 7.13.2), it is necessary to disable this Autostart mode and to switch to "CoLilo" command prompt instead (configuration mode).

The automatic boot of Linux operating system is connected with the **simultaneous compliance** with various conditions ("AND relation"). Consequently, for disabling Linux Autostart, it is sufficient to simply **not comply** with one of the conditions.

Table 11 lists up all conditions that are verified by the bootloader "CoLilo". All of them must be complied with to start an Autostart for the Linux-Image.

Table 11: Conditions for booting Linux

No.	Condition	Remark
1	"kfl=1" und "auto=1"	The Linux-Image stored in the Flash must be declared as valid during CoLilo configuration and the Autostart of the image must be activated after Reset (for corresponding CoLilo commands see section 7.13.2)
2	don't push "Boot" Button while "Reset" Button is pushed !	If at the moment of releasing Reset the "Boot"-button is pushed, the CoLilo don't start Linux booting. In this case the CoLilo start to it's own command prompt.
3	No interruption of Autostart via COM0 of the PLCmodule-C32	If all conditions are met, CoLilo verifies the serial interface COM0 of the PLCmodule-C32 for about 1 second after Reset regarding the reception of SPACE signals (ASCII 20H). If such a signal is received within that time, CoLilo will disable the Linux Autostart and will activate its own command prompt instead.

According to Table 11 , the Linux boot is disabled after Reset and the CoLilo command prompt is activated instead if the following conditions occur:

- (1) Button "Boot" while "Reset" is pushed
- (a) Button "Boot" and "Reset" pushed at same time
 (b) "Reset" released, "Boot" pushed
 (c) "Boot" released 2 seconds after "Reset"
- OR -
- (2) Reception of a SPACE signal (ASCII 20H) within 1 second after Reset

Communicating with the bootloader "CoLilo" only takes place via the serial interface ASC0 (resp. COM0) of the PLCmodule-C32. As receiver on the computer one of the terminal programs must be started (e.g. HyperTerminal or TeraTerm, see section 7.1) and must be configured as follows (see Figure 13):

- 19200 Baud
- 8 Data bit
- 1 Stop bit
- no parity
- no flow control

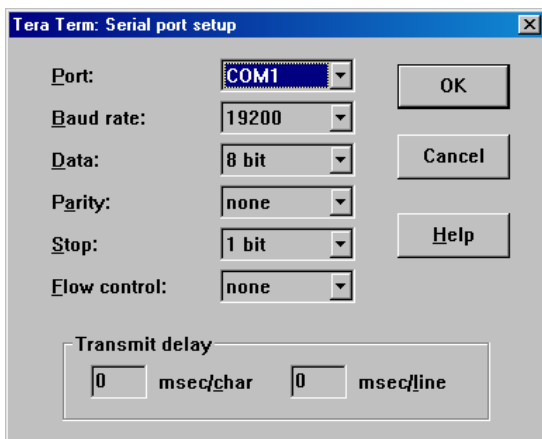


Figure 13: Terminal configuration using the example of "TeraTerm"

7.3 Ethernet configuration of the PLCmodule-C32

The main Ethernet configuration of the PLCmodule-C32 takes place within the bootloader "CoLilo" and is taken on for all software components (Linux, PLC firmware, HTTP server etc.). The Ethernet configuration is carried out via the serial interface ASC0 (resp. COM0). **Therefore, the CoLilo command prompt must be activated as described in section 7.1.** Table 12 lists up CoLilo commands necessary for the Ethernet configuration of the PLCmodule-C32.

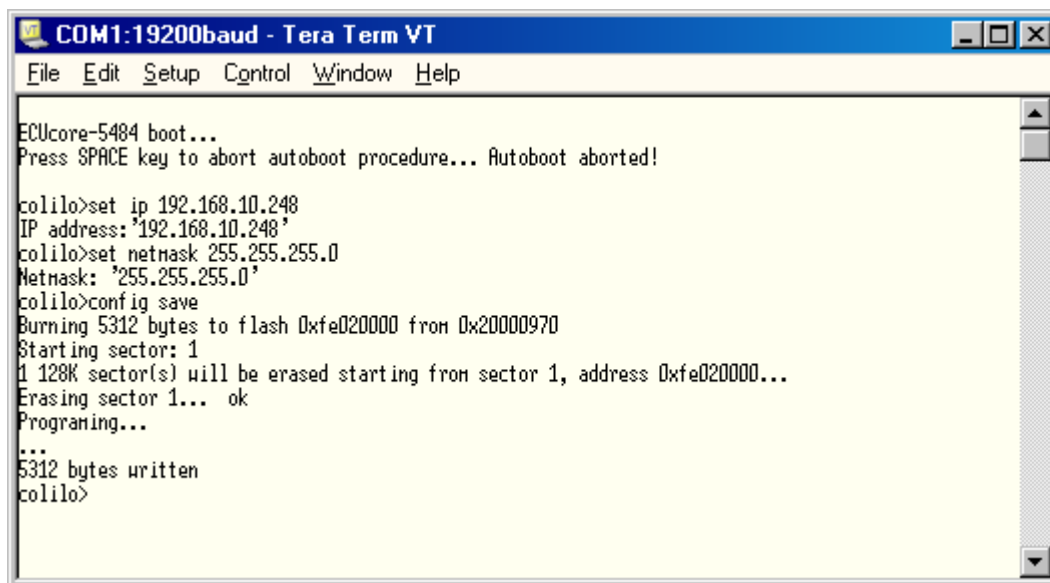
Table 12: "CoLilo" configuration commands of the PLCmodule-C32

Configuration	Command	Remark
MAC address	set mac <xx:xx:xx:xx:xx:xx>	The MAC address worldwide is a clear identification of the module and is assigned by the producer. It should not be modified by the user.
IP address	set ip <xxx.xxx.xxx.xxx>	This command sets the local IP address of the PLCmodule-C32. The IP address is to be defined by the network administrator.
Network mask	set netmask <xxx.xxx.xxx.xxx>	This command sets the network mask of the PLCmodule-C32. The network mask is to be defined by the network administrator.
Gateway address	set gw <xxx.xxx.xxx.xxx>	This command defines the IP address of the gateway which is to be used by the PLCmodule-C32. The gateway address is set by the network administrator. Advice: If PLCmodule-C32 and Programming PC are located within the same sub-net, defining the gateway address may be skipped and value "0.0.0.0" may be used instead.
Saving the configuration	config save	This command saves active configurations in the flash of the PLCmodule-C32.

Modified configurations may be verified again by entering "?" in the "CoLilo" command prompt. Active configurations are permanently saved in the Flash of the PLCmodule-C32 by command

config save

Modifications are adopted upon next Reset of the PLCmodule-C32.



```

COM1:19200baud - Tera Term VT
File Edit Setup Control Window Help
ECUcore-5484 boot...
Press SPACE key to abort autoboot procedure... Autoboot aborted!

colilo>set ip 192.168.10.248
IP address: '192.168.10.248'
colilo>set netmask 255.255.255.0
Netmask: '255.255.255.0'
colilo>config save
Burning 5312 bytes to flash 0xfe020000 from 0x20000970
Starting sector: 1
1 128K sector(s) will be erased starting from sector 1, address 0xfe020000...
Erasing sector 1... ok
Programing...
...
5312 bytes written
colilo>

```

Figure 14: Ethernet configuration of the PLCmodule-C32

After the configuration is finished and according to section 7.1, all conditions for a Linux Autostart must be re-established.

Upon Reset the module starts using the active configurations.

Advice: After the configuration is finished, the serial connection between PC and PLCmodule-C32 is no longer necessary.

7.4 PLC configuration of the PLCmodule-C32

7.4.1 PLC configuration via WEB-Frontend

After finishing the Ethernet configuration (see section 7.3), all further adjustments can take place via the integrated WEB-Frontend of the PLCmodule-C32. For the application of the PLCmodule-C32 using the Development Kit, basic configurations may also be set via local control elements (see section 7.4.2).

To configure the PLCmodule-C32 via WEB-Frontend it needs a WEB-Browser on the PC (e.g. Microsoft Internet Explorer, Mozilla Firefox etc.). To call the configuration page, prefix "*http://*" must be entered into the address bar of the WEB-Browser prior to entering the IP address of the PLCmodule-C32 as set in section 7.1, e.g. "*http://192.168.10.248*". Figure 15 exemplifies calling the PLCmodule-C32 configuration page in the WEB-Browser.

The standard setting (factory setting) requires a user login to configure the PLCmodule-C32 via WEB-Frontend. This is to prevent unauthorized access. Therefore, user name and password must be entered (see Figure 15). On delivery of the module, the following user account is preconfigured (see section 7.7):

User: PlcAdmin
Password: Plc123

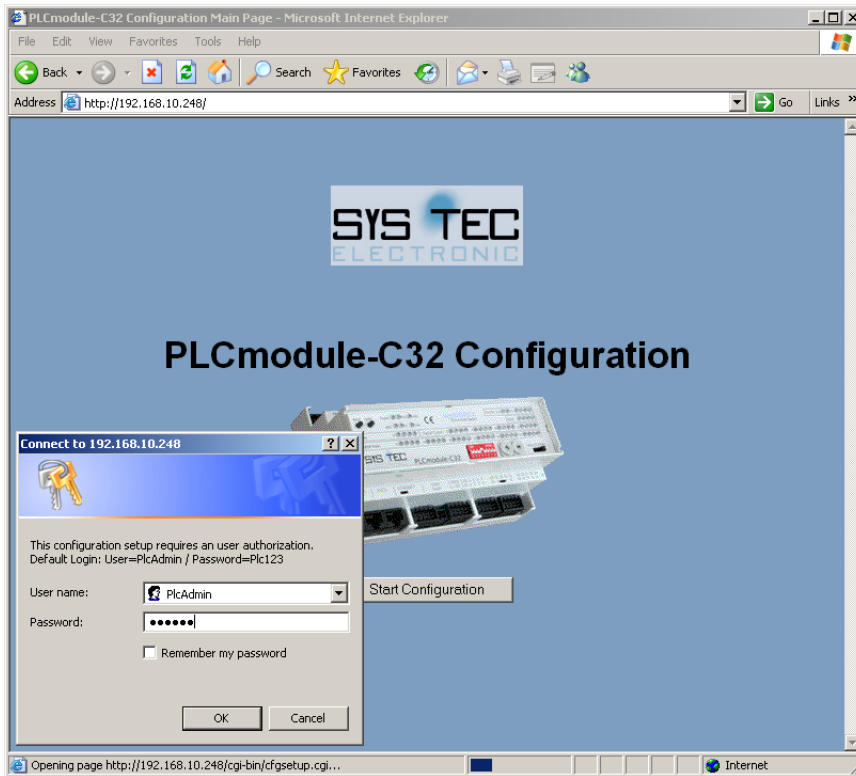


Figure 15: User login dialog of the WEB-Frontend

All configuration adjustments for the PLCmodule-C32 are based on dialogs. They are adopted into the file **"/home/plc/plcmodule-c32.cfg"** of the PLCmodule-C32 by activating the pushbutton "Save Configuration" (also compare section 7.4.3). After activating Reset the PLCmodule-C32 starts automatically using the active configuration. Figure 16 shows the configuration of the PLCmodule-C32 via WEB-Frontend.

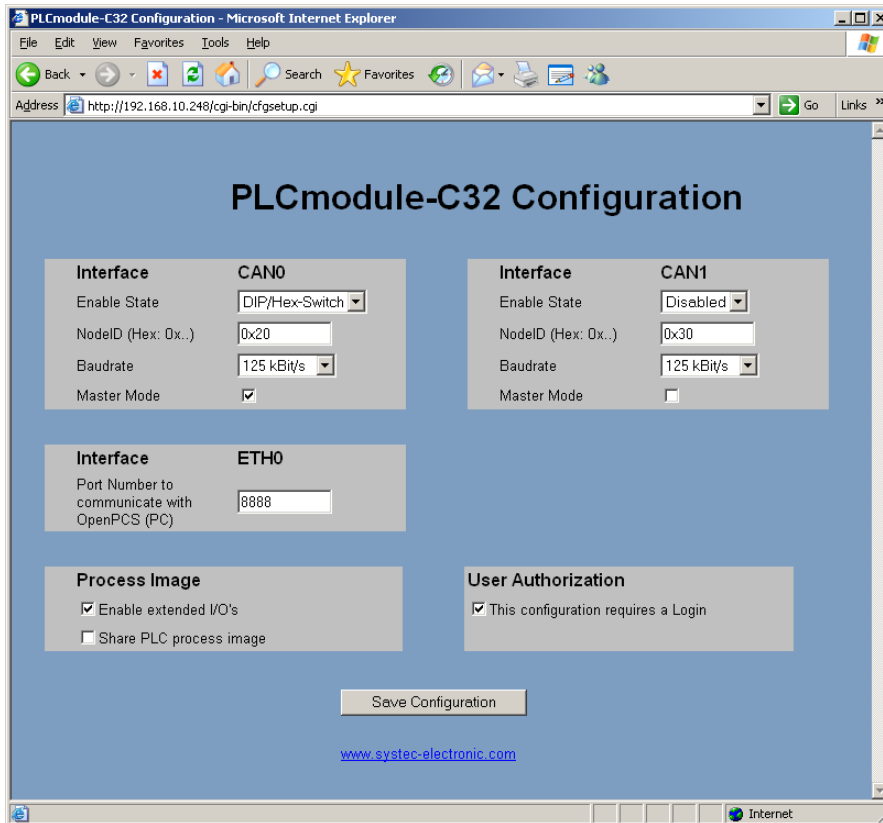


Figure 16: PLC configuration via WEB-Frontend

If "DIP/Hex-Switch" is chosen as Enable State of Interface CAN0, the configuration of this interface takes place via local control elements of the PLCmodule-C32 (see section 7.4.2).

The standard setting (factory setting) of the PLCmodule-C32 requires a user login to access the WEB-Frontend. Therefore, only the user name indicated in configuration file *"/home/plc/plcmodule-c32.cfg"* is valid (entry *"User="* in section *"[Login]"*, see section 7.4.3). Procedures to modify the user login password are described in section 7.10. To allow module configuration to another user, an appropriate user account is to be opened as described in section 7.9. Afterwards, the new user name must be entered into the configuration file *"/home/plc/plcmodule-c32.cfg"*. Limiting the user login to one user account is cancelled by deleting the entry *"User="* in section *"[Login]"* (see 7.4.3). Thus, any user account may be used to configure the module. By deactivating control box *"This configuration requires a Login"* in the field *"User Authorization"* of the configuration page (see Figure 16) free access to the module configuration is made available without previous user login.

7.4.2 PLC configuration via control elements of the PLCmodule-C32

The **configuration via control elements** of the PLCmodule-C32 is **preset upon delivery of the PLCmodule-C32**. This allows for an easy commissioning of the module by using CAN interface CAN0. Due to a limited number of switch elements, the initial setting of CAN0 is restricted. Using interface CAN1 requires a configuration via WEB-Frontend as described in section 7.4.1. This allows for other adjustments as well.

Advice: Configuring interface CAN0 is only possible via local control elements if "DIP/Hex-Switch" is activated as Enable State for CAN0 via WEB-Frontend (factory setting). Otherwise, configurations made via WEB-Frontend take priority over those via control elements.

Node address CAN0: The node address for interface CAN0 is set via Hex-Encoding switches of the PLCmodule-C32:

left (towards to DIP-Switch): High part of the node address
 right (towards to Run/Stop switch): Low part of the node address

Example: left=2 / right=0 → resulting node address = 20 Hex.

Bitrate CAN0: The bitrate for interface CAN0 is adjusted via bit positions 1-3 of DIP-Switch. Table 13 lists the coding of bitrates supported.

Table 13: Setting the bitrate for CAN0 via DIP-Switch

Bitrate [kBit/s]	DIP1	DIP2	DIP3
10	OFF	OFF	ON
20	ON	OFF	OFF
50	ON	OFF	ON
125	OFF	OFF	OFF
250	OFF	ON	ON
500	OFF	ON	OFF
800	ON	ON	ON
1000	ON	ON	OFF

Master mode CAN0: The Master mode is activated via Bit position 4 of DIP-Switch:

DIP4 = OFF: PLC is NMT-Slave
 DIP4 = ON: PLC is NMT-Master

7.4.3 Setup of the configuration file "plcmodule-c32.cfg"

The configuration file ***"/home/plc/plcmodule-c32.cfg"*** allows for comprehensive configuration of the PLCmodule-C32. Although, working in it manually does not always make sense, because most of the adjustments may easily be edited via WEB-Frontend (compare section 7.4.1). The setup of the configuration file is similar to the file format "Windows INI-File". It is divided into "[Sections]" which include different entries "Entry=". Table 14 shows all configuration entries. Entries of section "[CAN0]" take priority over settings via control elements (see section 7.4.2).

Table 14: Configuration entries of the CFG file

Section	Entry	Value	Meaning
[CAN0]	Enabled	-1, 0, 1	-1: Interface CAN0 is activated, configuration takes place via control elements of the PLCmodule-C32 (factory setting, see section 7.4.2) 0: Interface CAN0 is deactivated 1: Interface CAN0 is activated, configuration takes place via entries of the configuration file below
	NodeID	1 ... 127 or 0x01 ... 0x7F	Node number for interface CAN0 (decimal or hexadecimal with prefix "0x")
	Baudrate	10, 20, 50, 125, 250, 500, 800, 1000	Bitrate for interface CAN0
	MasterMode	0, 1	1: Master mode is activated 0: Master mode is deactivated
[CAN1]	Enabled	0, 1	0: Interface CAN1 is deactivated 1: Interface CAN1 is activated, configuration takes place via entries of the configuration file below
	NodeID	1 ... 127 or 0x01 ... 0x7F	Node number for interface CAN1 (decimal or hexadecimal with prefix "0x")
	Baudrate	10, 20, 50, 125, 250, 500, 800, 1000	Bitrate for interface CAN1
	MasterMode	0, 1	1: Master mode is activated 0: Master mode is deactivated
[ETH0]	PortNum	Default Port no: 8888	Port number for the communication with the Programming-PC and for program download (only for PLCmodule-C32/Z5, order number 3090002)
[Proclmg]	EnableExtIo	0, 1	0: only process I/O's of the PLCmodule-C32 are used for the process image (except on-board Temperature Sensor) 1: All I/O's are used for the process image (incl. on-board Temperature Sensor)
	EnableSharing	0, 1	0: No sharing of process image 1: Sharing of process image is enabled (see section 8)

[Login]	Authorization	0, 1	0: Configuration via WEB-Frontend is possible without user login 1: Configuration via WEB-Frontend requires user login
	User	Default Name: PlcAdmin	If entry "User=" is available, only the user name defined is accepted for the login to configure via WEB-Frontend. If the entry is not available, any user registered on the PLCmodule-C32 (see section 7.9) may login via WEB-Frontend.

The configuration file *"/home/plc/plcmodule-c32.cfg"* includes the following factory settings:

```
[Login]
Authorization=1
User=PlcAdmin

[CAN0]
Enabled=-1
NodeID=0x20
Baudrate=125
MasterMode=1

[CAN1]
Enabled=0
NodeID=0x30
Baudrate=125
MasterMode=0

[ETH0]
PortNum=8888

[ProcImg]
EnableExtIo=1
EnableSharing=0
```

7.5 Boot configuration of the PLCmodule-C32

The PLCmodule-C32 is configured so that after Reset the PLC firmware starts automatically. Therefore, all necessary commands are provided by the start script *"/home/etc/autostart"*. Hence, the required environment variables are set and drivers are booted.

If required, the start script *"/home/etc/autostart"* may be complemented by further entries. For example, by entering command *"ftpd -D"*, the FTP server is called automatically when the PLCmodule-C32 is booted. The script can be edited directly on the PLCmodule-C32 in the FTP client *"WinSCP"* (compare section 7.1) using pushbutton *"F4"* or *"F4 Edit"*.

7.6 Selecting the appropriate firmware version

The PLCmodule-C32 is delivered with different firmware versions. Those vary in the communication protocol for the data exchange with the programming PC and they differ from each other regarding the availability of FB communication classes (see section 6.3). The selection of the appropriate firmware version takes place in the start script *"/home/etc/autostart"*. By default, the *"BoardID"* of the module

as set in the bootloader "CoLilo" is analyzed. Table 15 lists up the assignments of firmware versions and BoardIDs.

Table 15: Assignment of BoardIDs and firmware versions for the PLCmodule-C32

BoardID	Firmware Version	Remark
1006004	plcmodule-c32-z4	PLCmodule-C32/Z4 (CANopen) communication with the programming PC via CANopen protocol (Interface CAN0)
1006005	plcmodule-c32-z5	PLCmodule-C32/Z5 (Ethernet) communication with the programming PC via UDP protocol (Interface ETH0)

The configuration of BoardIDs takes place via the serial interface ASC0 (resp. COM0). **Therefore, the CoLilo command prompt must be activated as described in section 7.1.** Setting BoardIDs is carried out via the CoLilo command "*assign boardid*" by entering the corresponding number listed in Table 15, e.g.:

```
assign boardid 1006005
```

The modified setting can be verified by entering "*boardinfo*" at the "CoLilo" command prompt.
Command

config save

persistently saves the current selection in the Flash of the PLCmodule-C32. Figure 17 visualizes the configuration of the BoardID.

```

COM1:19200baud - Tera Term VT
File Edit Setup Control Window Help
ECUcore-5484 boot...
Press SPACE key to abort autoboot procedure... Autoboot aborted!

colilo>assign boardid 1006005
Board type ID: 1006005
colilo>boardinfo
Hardware info: PCB: 4152.0 (#1), PLD: 1.00 (#5)
Auto boot DIP: on
Board name: ECUcore-5484
Board type ID: 1006005
Order number: 3090002
Serial number: 111861
Info block 1:
Info block 2:
Info block 3:
Info block 4:
Info block 5:
Info block 6:
Info block 7:
Info block 8:
Finger print:
colilo>config save
Burning 5312 bytes to flash 0xfe020000 from 0x20000970
Starting sector: 1
1 128K sector(s) will be erased starting from sector 1, address 0xfe020000...
Erasing sector 1... ok
Programming...
...
5312 bytes written
colilo>

```

Figure 17: Selecting the appropriate firmware version for the PLCmodule-C32

After completing the configuration, all preconditions for a Linux Autostart must be reestablished according to section 7.1.

Alternatively, the appropriate firmware version may be selected directly in the start script `"/home/etc/autostart"`. Therefore, delete part "Select PLC Type" and insert the appropriate firmware instead, e.g.:

```
PLC_FIRMWARE=$PLC_DIR/plcmodule-c32-z5
```

7.7 Predefined user accounts

All user accounts listed in Table 16 are predefined upon delivery of the PLCmodule-C32. Those allow for a login to the command shell (serial RS232 connection or Telnet) and at the FTP server of the PLCmodule-C32.

Table 16: Predefined user accounts of the PLCmodule-C32

User name	Password	Remark
PlcAdmin	Plc123	Predefined user account for the administration of the PLCmodule-C32 (configuration, user administration, software updates etc.)
root	Sys123	Main user account ("root") of the PLCmodule-C32

7.8 Login to the PLCmodule-C32

7.8.1 Login to the command shell

In some cases the administration of the PLCmodule-C32 requires the entry of Linux commands in the command shell. Therefore, the user must be directly logged in at the module. There are two different possibilities:

- Logging in is possible with the help of a **Terminal program** (e.g. HyperTerminal or TeraTerm, see section 7.1) via the serial interface **COM0** of the PLCmodule-C32 – analog to the procedure described for the Ethernet configuration in section 7.1. **For the configuration of the terminal settings pay attention to only use "CR" (carriage return) as end-of-line character.** Login with user name and password is not possible for "CR+LF" (carriage return + line feed)!
- Alternatively, the login is possible using a **Telnet client** (e.g. Telnet or also TeraTerm) via the Ethernet interface **ETH0** of the PLCmodule-C32.

For logging in to the PLCmodule-C32 via the Windows standard Telnet client, the command "*telnet*" must be called by using the IP address provided in section 7.1, e.g.

```
telnet 192.168.10.248
```

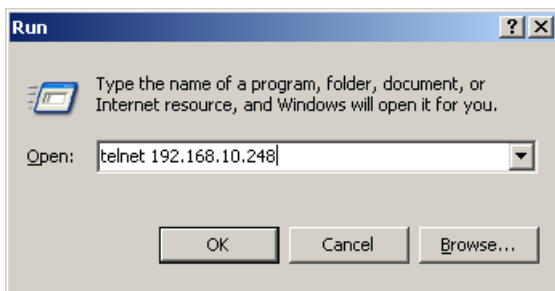


Figure 18: Calling the Telnet client in Windows

Logging in to the PLCmodule-C32 is possible in the Terminal window (if connected via ASC0) or in the Telnet window (if connected via ETH0). The following user account is preconfigured for the administration of the module upon delivery of the PLCmodule-C32 (also compare section 7.7):

User: *PlcAdmin*
 Password: *Plc123*

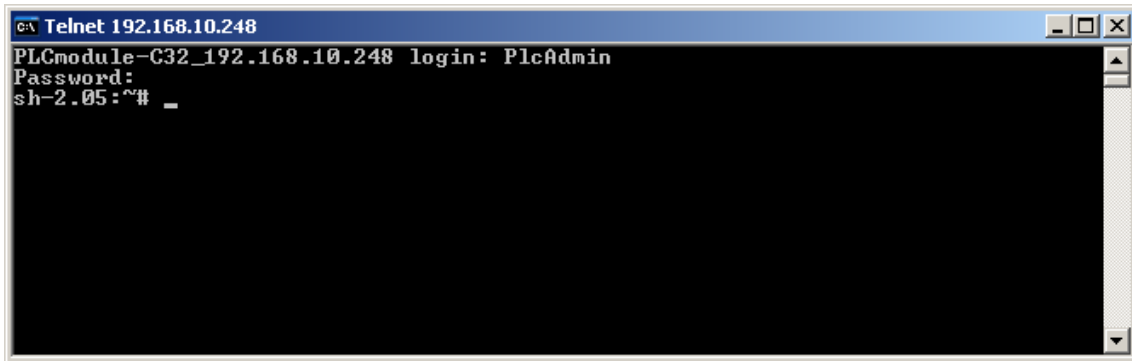


Figure 19: Login to the PLCmodule-C32

Figure 19 exemplifies the login to the PLCmodule-C32 using a Windows standard Telnet client.

7.8.2 Login to the FTP server

The PLCmodule-C32 has available a FTP server (FTP Daemon) that allows file exchange with any computer (up- and download of files). Due to security and performance reasons, the FTP server is deactivated by default and must be started manually if required. Therefore, the user must first be logged in to the command shell of the PLCmodule-C32 following the procedures described in section 7.8.1. Afterwards, the following command must be entered in the Telnet or Terminal window:

```
ftpd -D
```

Figure 20 illustrates an example for starting the FTP server.

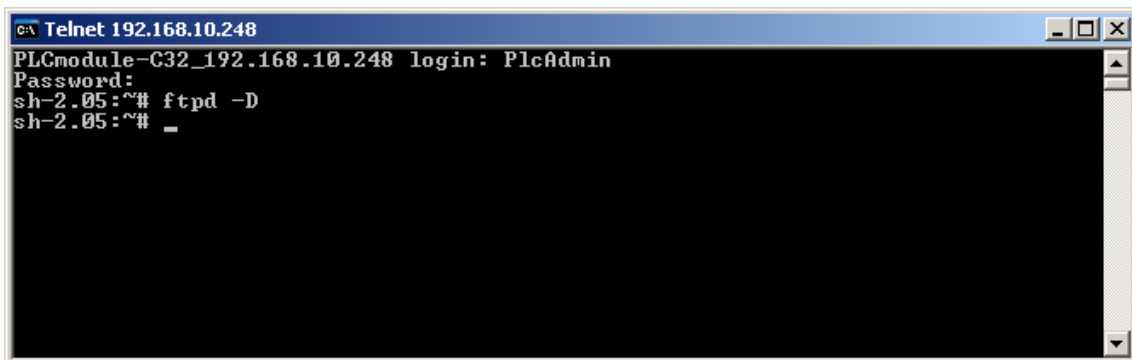


Figure 20: Starting the FTP server

Advice: By entering command "*ftpd -D*" in the start script "*/home/etc/autostart*", the FTP server may be called automatically upon boot of the PLCmodule-C32 (see section 7.5).

"WinSCP" - which is available as open source - would be suitable as FTP client for the computer (see section 7.1). It consists of only one EXE file, needs no installation and may be started immediately. After program start, dialog "WinSCP Login" appears (see Figure 21) and must be adjusted according to the following configurations:

File protocol: FTP
 Host name: IP address for the PLCmodule-C32 as set in section 7.3
 User name: PlcAdmin (for predefined user account, see section 7.7)
 Password: Plc123 (for predefined user account, see section 7.7)

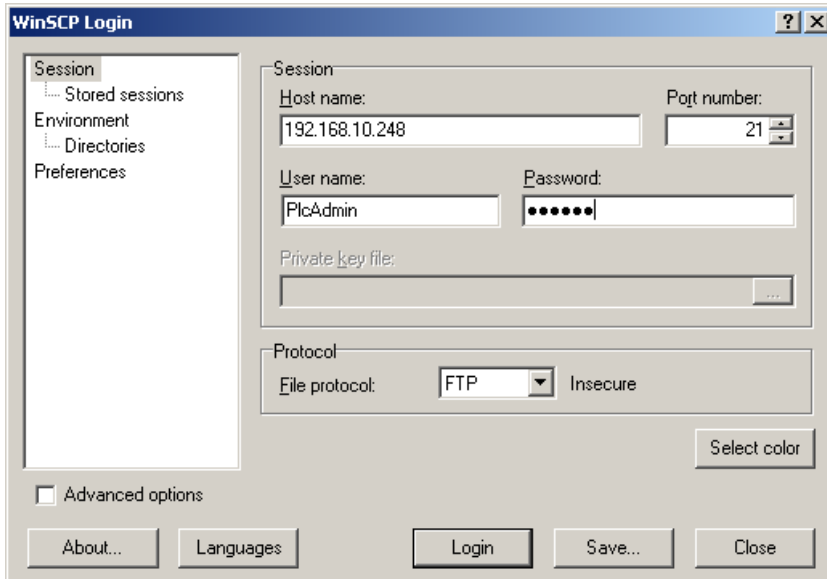


Figure 21: Login settings for WinSCP

After using pushbutton "Login", the FTP client logs in to the PLCmodule-C32 and lists up the active content of directory "/home" in the right window. Figure 22 shows FTP client "WinSCP" after successful login to the PLCmodule-C32.

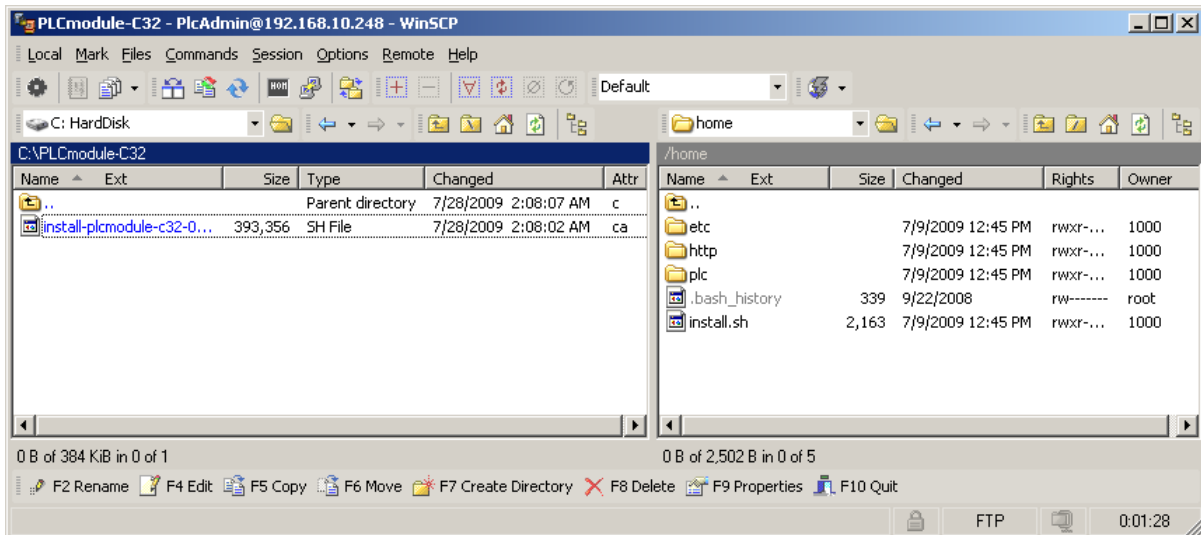


Figure 22: FTP client for Windows "WinSCP"

After successful login, configuration files on the PLCmodule-C32 may be edited by using pushbuttons "F4" or "F4 Edit" within the FTP client "WinSCP" (select transfer mode "Text"). With the help of pushbutton "F5" or "F5 Copy", files may be transferred between the computer and the PLCmodule-C32, e.g. for data backups of the PLCmodule-C32 or to transfer installation files for firmware updates (select transfer mode "Binary").

7.9 Adding and deleting user accounts

Adding and deleting user accounts requires the login to the PLCmodule-C32 as described in section 7.8.1.

Adding a new user account takes place via Linux command *"adduser"*. In embedded systems such as the PLCmodule-C32, it does not make sense to open a directory for every user. Hence, parameter *"-H"* disables the opening of new directories. By using parameter *"-h /home"* instead, the given directory *"/home"* is rather assigned to the new user. To open a new user account on the PLCmodule-C32, Linux command *"adduser"* is to be used as follows:

```
adduser -h /home -H -G <group> <username>
```

Figure 23 exemplifies adding a new account on the PLCmodule-C32 for user *"admin2"*.

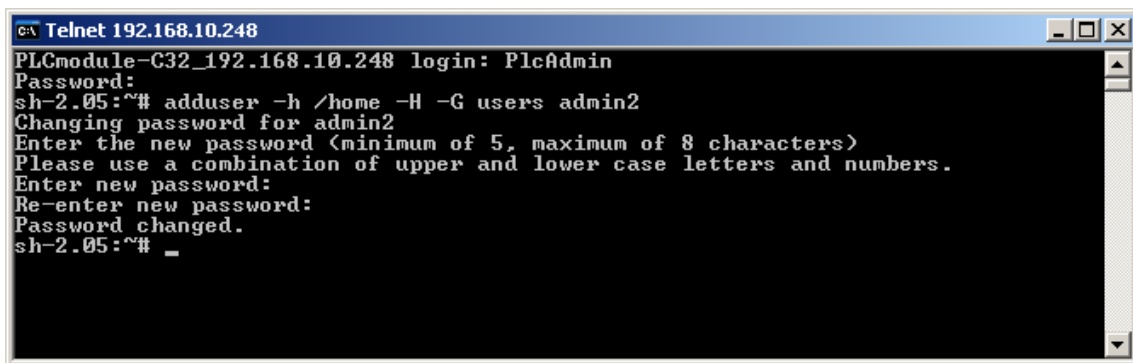


Figure 23: Adding a new user account

Advice: If the new user account shall be used to access WEB-Frontend, the user name must be entered into the configuration file *"plcmodule-c32.cfg"* (for details about logging in to WEB-Frontend please compare section 7.4.1 and 7.4.3).

To **delete** an existing user account from the PLCmodule-C32, Linux command *"deluser"* plus the respective user name must be used:

```
deluser <username>
```

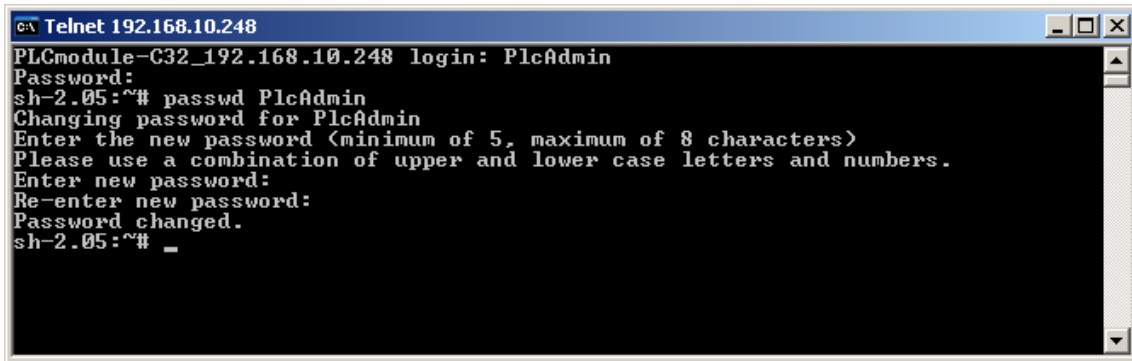
7.10 How to change the password for user accounts

Changing the password for user accounts requires login to the PLCmodule-C32 as described in section 7.8.1.

To change the password for an existing user account on the PLCmodule-C32, Linux command *"passwd"* plus the respective user name must be entered:

```
passwd <username>
```


Figure 24 exemplifies the password change for user "PlcAdmin".



```

c:\ Telnet 192.168.10.248
PLCmodule-C32_192.168.10.248 login: PlcAdmin
Password:
sh-2.05:~# passwd PlcAdmin
Changing password for PlcAdmin
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Re-enter new password:
Password changed.
sh-2.05:~# _

```

Figure 24: Changing the password for an user account

7.11 Setting the system time

Setting the system time requires login to the PLCmodule-C32 as described in section 7.8.1.

There are two steps for setting the system time of the PLCmodule-C32. At first, the current date and time must be set using Linux command "date". Afterwards, by using Linux command "hwclock -w" the system time is taken over into RTC module of the PLCmodule-C32.

Linux command "date" is structured as follows:

```
date [options] [MMDDhhmm[[CC]YY][.ss]]
```

Example:

```

date    09  22  10  20  2008.55
      |  |  |  |  |  |  |
      |  |  |  |  |  |  +--- second
      |  |  |  |  |  +----- year
      |  |  |  |  +----- minute
      |  |  |  +----- hour
      |  |  +----- day
      |  +----- month

```

Spaces in the parameter list are only inserted to present the example above clearly. All spaces do not apply when the command is actually used. To set the system time of the PLCmodule-C32 to 2008/09/22 and 10:20:55 (as shown in the example above), the following commands are necessary:

```
date 092210202008.55
hwclock -w
```

The current system time is displayed by entering Linux command "date" (without parameter). Linux command "hwclock -r" can be used to recall current values from the RTC. Figure 25 exemplifies setting and displaying the system time.

```

CA Telnet 192.168.10.248
PLCmodule-C32_192.168.10.248 login: PlcAdmin
Password:
sh-2.05:~# date 092210202008.55
Mon Sep 22 10:20:55 UTC 2008
sh-2.05:~# hwclock -w
sh-2.05:~#
sh-2.05:~# date
Mon Sep 22 10:21:27 UTC 2008
sh-2.05:~# hwclock -r
Mon Sep 22 10:21:33 2008 0.000000 seconds
sh-2.05:~# _

```

Figure 25: Setting and displaying the system time

Upon start of the PLCmodule-C32, date and time are taken over from the RTC and set as current system time of the module. Therefore, Linux command "hwclock -r" is necessary which is included in start script "**etc/rc.d/rc.modules**".

7.12 File system of the PLCmodule-C32

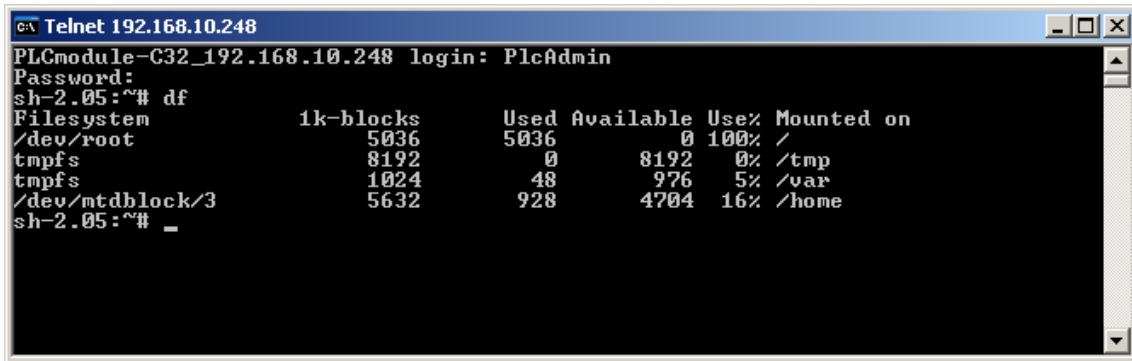
Pre-installed Embedded Linux on the PLCmodule-C32 provides part of the system memory in form of a file system. Being usual for embedded systems, most of this file system is "read/only" which means that changes to this part can only be made by creating a new Linux-Image for the PLCmodule-C32. The advantage hereby is the resistance of a read/only file system against damages in case of power breakdowns. Those occur relatively often in embedded systems because embedded systems are usually simply turned off without previous shutdown.

Table 17 lists up writable paths of the file system during runtime. Path "**/home**" comprises a flash disk that provides part of the on-board flash memory of the PLCmodule-C32 as file system. This path is used to store all files modifiable and updatable by the user, e.g. configuration files, PLC firmware and PLC program files that have been loaded onto the module. Directory "**/tmp**" is appropriately sized to function as temporary buffer for FTP downloads of firmware archives for PLC software updates (see section 7.13.1).

Table 17: File system configuration of the PLCmodule-C32

Path	Size	Description
/home	5632 kByte	Flash disk to permanently store files modifiable and updatable by the user (e.g. configuration files, PLC firmware, PLC program), data preservation in case of power breakdown
/tmp	8192 kByte	RAM disk, suitable as intermediate buffer for FTP downloads, but no data preservation in case of power breakdown
/var	1024 kByte	RAM disk which is used by the system to store temporary files, no data preservation in case of power breakdown
/mnt		Target for integrating remote directories, it is not part of the PLCmodule-C32 standard functionality

Sizes of file system paths that are configured or still available can be identified by using Linux command "df" ("DiskFree") – see Figure 26.



```

Telnet 192.168.10.248
PLCmodule-C32_192.168.10.248 login: PlcAdmin
Password:
sh-2.05:~# df
Filesystem          1k-blocks      Used Available Use% Mounted on
/dev/root            5036           5036      0 100% /
tmpfs                8192            0      8192   0% /tmp
tmpfs                1024            48       976   5% /var
/dev/mtdblock/3     5632           928      4704  16% /home
sh-2.05:~# _

```

Figure 26: Display of information about the file system

Particular information about the system login and handling the Linux command shell of the PLCmodule-C32 is given attention in section 7.8.

7.13 Software update of the PLCmodule-C32

All necessary firmware components to run the PLCmodule-C32 are already installed on the module upon delivery. Hence, firmware updates should only be required in exceptional cases, e.g. to input new software that includes new functionality.

7.13.1 Updating the PLC firmware

PLC firmware indicates the run time environment of the PLC. **PLC firmware** can only be generated and modified by the producer; **it is not identical with the PLC user program** which is created by the PLC user. The PLC user program is directly transferred from the *OpenPCS* programming environment onto the module. No additional software is needed.

Updating the PLC firmware requires login to the command shell of the PLCmodule-C32 as described in section 7.8.1 and login to the FTP server as described in section 7.8.2.

Updating the PLC firmware takes place via a self-extracting firmware archive that is transferred onto the PLCmodule-C32 via FTP. After starting the FTP server on the PLCmodule-C32 (command "*ftpd -D*", see section 7.8.2), the respective firmware archive can be transferred into directory "*/tmp*" of the PLCmodule-C32 (see Figure 27).

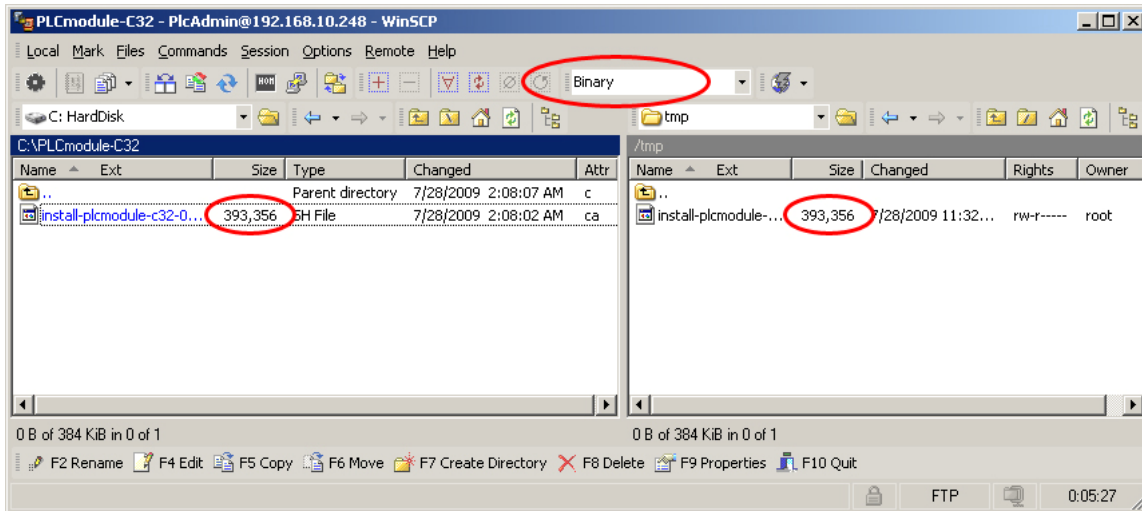


Figure 27: File transfer in FTP client "WinSCP"

Important: To transfer the firmware archive via FTP, transfer type "Binary" must be chosen. If FTP client "WinSCP" is used, the appropriate transfer mode is to be chosen from the menu bar. After downloading the firmware archive, it must be checked if the file transferred to the PLCmodule-C32 has the exact same size as the original file on the computer (compare Figure 27). Any differences in that would indicate a mistaken transfer mode (e.g. "Text"). In that case the transfer must be repeated using transfer type "Binary".

After downloading the self-extracting archive, the PLC firmware must be installed on the PLCmodule-C32. Therefore, the following commands are to be entered in the Telnet window. It must be considered that the file name for the firmware archive is labeled with a version identifier (e.g. "install-plcmodule-c32-0408_0100.sh" for version 4.08.01.00). This number must be adjusted when commands are entered:

```
cd /tmp
chmod +x install-plcmodule-c32-0408_0100.sh
./install-plcmodule-c32-0408_0100.sh
```

Advice: The command shell of the PLCmodule-C32 is able to automatically complete names if the Tab key is used ("tab completion"). Hence, it should be sufficient to enter the first letters of each file name and the system will complement it automatically. For example, "./ins" is completed to "./install-plcmodule-c32-0408_0100.sh" if the Tab key is used.

```

CA Telnet 192.168.10.248
PLCmodule-C32_192.168.10.248 login: PlcAdmin
Password:
sh-2.05:~# ftpd -D
sh-2.05:~# cd /tmp
sh-2.05:/tmp# chmod +x ./install-plcmodule-c32-0408_0100.sh
sh-2.05:/tmp# ./install-plcmodule-c32-0408_0100.sh

--- PLCmodule-C32 Runtime System Installer ---

Checking PLCmodule-C32 hardware for update requirements...
Extract new I/O driver './plc/pmc32drv.ko' to tmp dir...
./plc/pmc32drv.ko
Try to load new I/O driver...
PLCmodule-C32 hardware check ok.

Running installation... please wait

./etc
./etc/rc.usr
./etc/autostart
./http
./http/html
./http/html/PLCmodule-C32.gif
./http/html/systec_logo.jpg
./http/html/index.html
./http/html/PmC32Config.html
./http/boa.conf
./http/cgi-bin
./http/cgi-bin/cfgsetup.cfg
./http/cgi-bin/cfgsetup.cgi
./http/mime.types
./install.sh
./plc
./plc/plcmodule-c32.cfg
./plc/plcmodule-c32-z4
./plc/plcmodule-c32-z5
./plc/shpingdemo
./plc/iodrvdemo
./plc/stopplc
./plc/pmc32drv.ko
./plc/pmc32drv.so
./plc/runplc
./plc/version
./plc/candrv.ko

Installation has been finished.
Please restart system to activate the new firmware.

sh-2.05:/tmp# _

```

Figure 28: Installing PLC firmware on the PLCmodule-C32

Figure 28 exemplifies the installation of PLC firmware on the PLCmodule-C32. After Reset the module is started using the updated firmware.

Advice: If the PLC firmware is updated, the configuration file `"/home/plc/plcmodule-c32.cfg"` is overwritten. This results in a reset of the PLC configuration to default settings. Consequently, after an update, the configuration described in section 7.4 should be checked and if necessary it should be reset.

7.13.2 How to update the Linux-Image

Updating the Linux-Image takes place via TFTP (Trivial FTP) within Linux bootloader "CoLilo". Therefore, an appropriate TFTP server is necessary on the computer, e.g. freeware "TFTPD32" (compare section 7.1). The program consists of only one EXE file that requires no installation and can be run immediately. After the program start, an appropriate working directory ("Current Directory") should be created by clicking on pushbutton "Browse" (e.g. "C:\PLCmodule-C32"). The Linux-Image for the PLCmodule-C32 must be located in this directory (`image.bin.gz`).

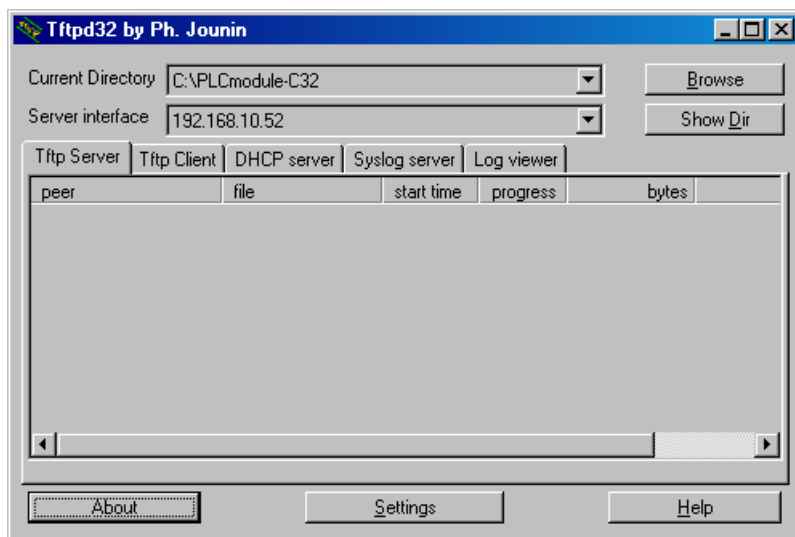


Figure 29: TFTP server for Windows "TFTPD32"

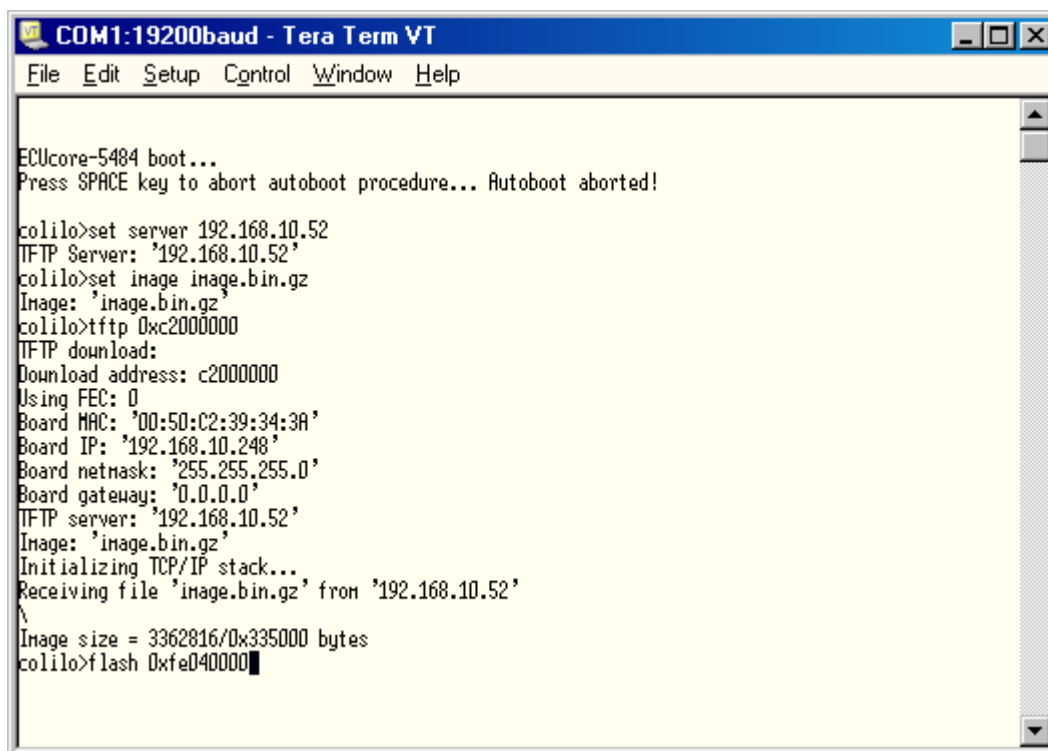
A TFTP download of the Linux-Image **requires** that the **Ethernet configuration** of the PLCmodule-C32 is **completed** according to procedures describes in **section 7.3**. To update the Linux-Image it is necessary to have available another serial connection to the PLCmodule-C32 in addition to the Ethernet connection. All configurations for the terminal program as described in section 7.1 apply (19200 Baud, 8 Data bit, 1 Stop bit, no parity and no flow control).

Updating the Linux-Image of the PLCmodule-C32 is only possible if Linux is not running. Hence, Linux Autostart must be disabled prior to the updating process and "CoLilo" command prompt must be used instead. Procedures are described in section 7.1.

After disabling of Linux Autostart, the "CoLilo" command prompt answers. To update the Linux-Image the following commands must be entered according to the following sequence:

Table 18: Command sequence to update the Linux-Image on the PLCmodule-C32

Command	Meaning
<code>set server <host_ip_addr></code>	Setting the IP address of the TFTP server. If "TFTPD32" is used, the address is shown in field "Server Interface" on the PC.
<code>set image image.bin.gz</code>	Setting the file names for the Linux-Image to be loaded
<code>tftp 0xC2000000</code>	Downloading the Linux-Image from the PC to the PLCmodule-C32
<code>flash 0xFE040000</code>	Saving the Linux-Image in the Flash of the PLCmodule-C32
<code>set kfl 1</code>	To validate the Linux-Image stored in the Flash of the PLCmodule-C32
<code>set auto 1</code>	Autostart of the Linux-Image after Reset is activated
<code>config save</code>	Saving the current configurations in the Flash



```
COM1:19200baud - Tera Term VT
File Edit Setup Control Window Help

ECUcore-5484 boot...
Press SPACE key to abort autoboot procedure... Autoboot aborted!

colilo>set server 192.168.10.52
TFTP Server: '192.168.10.52'
colilo>set image image.bin.gz
Image: 'image.bin.gz'
colilo>tftp 0xc2000000
TFTP download:
Download address: c2000000
Using FEC: 0
Board MAC: '00:50:C2:39:34:3A'
Board IP: '192.168.10.248'
Board netmask: '255.255.255.0'
Board gateway: '0.0.0.0'
TFTP server: '192.168.10.52'
Image: 'image.bin.gz'
Initializing TCP/IP stack...
Receiving file 'image.bin.gz' from '192.168.10.52'
\
Image size = 3362816/0x335000 bytes
colilo>flash 0xfe040000
```

Figure 30: Downloading the Linux-Image to the PLCmodule-C32

After completing the configuration, conditions for a Linux Autostart must be reestablished according to instructions in section 7.1.

After Reset is activated (e.g. pushbutton on the top of the module), the PLCmodule-C32 starts automatically using the current Linux-Image.

Advice: After the configuration is finished, the serial connection between the computer and the PLCmodule-C32 is no longer necessary.

8 Data exchange via shared process image

8.1 Overview of the shared process image

The PLCmodule-C32 is based on the operating system Embedded Linux. Thus, it is possible to execute other user-specific programs simultaneously to running the PLC firmware. The PLC program and a user-specific C/C++ application can exchange data by using the same process image (shared process image). Implementing user-specific C/C++ applications is based on the Software package SO-1095 ("VMware-Image of the Linux development system for the ECUcore-5484").

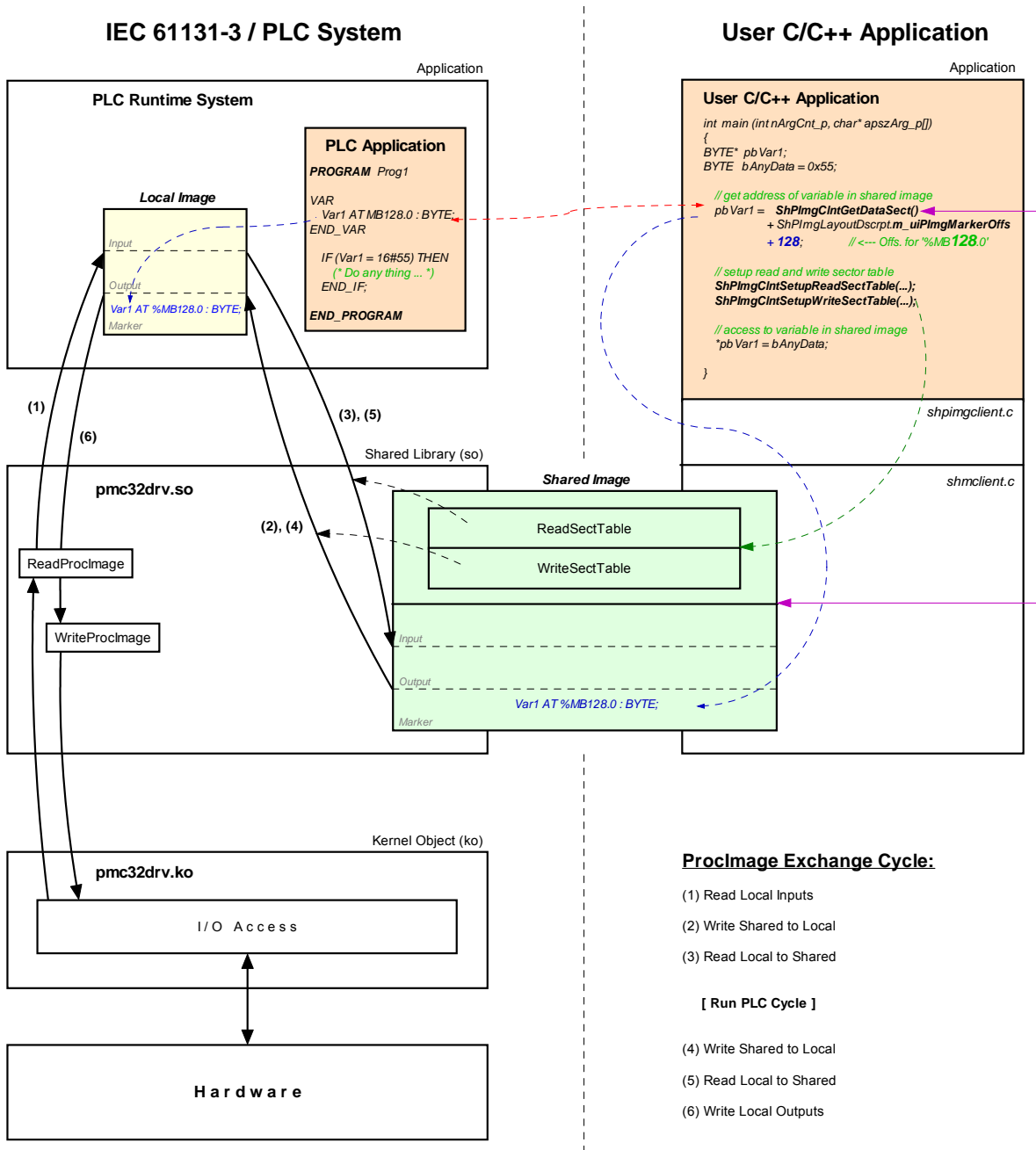


Figure 31: Overview of the shared process image

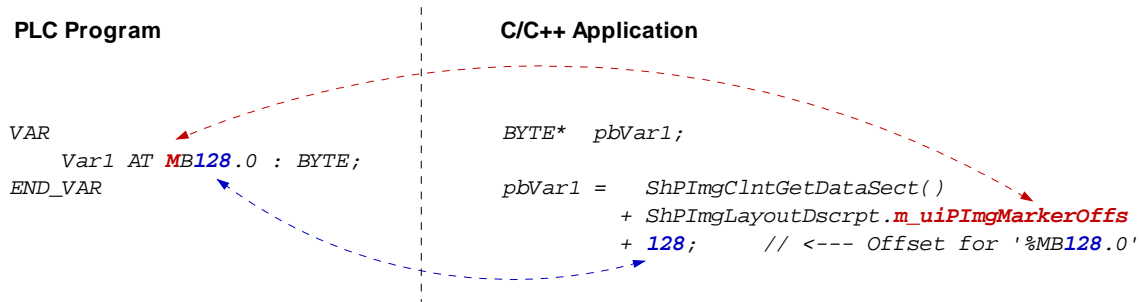
Not all variables are utilizable via the shared process image within a C/C++ application. Only those directly addressed variables that the PLC program generates within the process image. As shown in Figure 31, two separate process images are used for the data exchange with an external application inside of the PLC runtime system. This is necessary to meet the IEC 61131-3 requirement that the initial PLC process image may not be modified during the entire execution of one PLC program cycle. Thereby, the PLC program always operates with the internal process image that is locally generated within the PLC runtime system ("Local Image" in Figure 31). This is integrated within the PLC runtime system and is protected against direct accesses from the outside. On the contrary, the user-specific, external C/C++ application always uses the shared process image ("Shared Image" in Figure 31). This separation of two process images enables isolation between accesses to the PLC program and the external application. Those two in parallel and independently running processes now must only be synchronized for a short period of time to copy the process data.

An activation of option "**Share PLC process image**" within the PLC configuration enables data exchange with external applications (see section 7.4.1). Alternatively, entry "*EnableSharing=*" can directly be set within section "*[Proclmg]*" of the configuration file "*/home/plc/plcmodule-c32.cfg*" (see section 7.4.3). The appropriate configuration setting is evaluated upon start of the PLC firmware. By activating option "**Share PLC process image**", the PLC firmware creates a second process image as Shared Memory ("Shared Image" in Figure 31). Its task is to exchange data with external applications. Hereby, the PLC firmware functions as Server and the external, user-specific C/C++ application functions as Client.

ReadSectorTable and **WriteSectorTable** both control the copying of data between the two process images. Both tables are filled by the Client (external, user-specific C/C++ application) and are executed by Server (PLC runtime system). The Client defines ranges of the PLC process image from which it will read data (*ReadSectorTable*) or in which it will write data (*WriteSectorTable*). Hence, the terms "*Read*" and "*Write*" refer to data transfer directions from the viewpoint of the Client.

Sections to read and write may comprise all sections of the entire process image – input, output as well as marker sections. This allows for example that a Client application writes data into the input section of the PLC process image and reads data from the output section. Figure 31 shows the sequence of single read and write operations. Prior to the execution of a PLC program cycle, the physical inputs are imported into the local process image of the PLC (1). Afterwards, all sections defined in *WriteSectorTable* are taken over from the shared process image into the local process image (2). By following this sequence, a Client application for example is able to overwrite the value of a physical input. This may be used for simulation purposes as well as for setting input data to constant values ("*Forcen*"). Similarly, prior to writing the process image onto the physical outputs (6), sections defined in *WriteSectorTable* are taken over from the shared process image into the local process image. (4). Thus, a Client application is able to overwrite output information generated by the PLC program.

The PLC firmware provides the **setup of the process image**. The Client application receives information about the setup of the process image via function **ShPlmgClntSetup()**. This function enters start offsets and values of the input, output and marker sections into the structure of type *tShPlmgLayoutDscrpt*. Function **ShPlmgClntGetDataSect()** provides the start address of the shared process image. Upon defining a variable within the PLC program, its absolute position within the process image is determined through sections (%I = Input, %Q = Output, %M = Marker) and offset (e.g. %MB128.0). In each section the offset starts at zero, so that for example creating a new variable in the marker section would be independent of values in the input and output section. Creating a corresponding **pair of variables** in the PLC program as well as in the C/C++ application allows for data exchange between the PLC program and the external application. Therefore, both sides must refer to the same address. Structure *tShPlmgLayoutDscrpt* reflects the physical setup of the process image in the PLC firmware including input, output and marker sections. This is to use an addressing procedure for defining appropriate variables in the C/C++ application that is comparable to the PLC program. Hence, also in the C/C++ program a variable is defined in the shared process image by indicating the respective section and its offset. The following example illustrates the creation of a corresponding variable pair in the PLC program and C/C++ application:



As described above, **ReadSectorTable** and **WriteSectorTable** manage the copy process to exchange variable contents between the PLC and the C/C++ program. Following the example illustrated, the Client (C/C++ application) must enter an appropriate value into the **WriteSectorTable** to transfer the value of a variable from the C/C++ application to the PLC program (**WriteSectorTable**, because the Client “writes” the variable to the Server):

```

// specify offset and size of 'Var1' and define sync type (always or on demand?)
WriteSectTab[0].m_uiPIImgDataSectOffs = ShPIImgLayoutDscript.m_uiPIImgMarkerOffs + 128;
WriteSectTab[0].m_uiPIImgDataSectSize = sizeof(BYTE);
WriteSectTab[0].m_SyncType           = kShPIImgSyncOnDemand;

// define the WriteSectorTable with the size of 1 entry
ShPIImgClntSetupWriteSectTable (WriteSectTab, 1);

```

If several variable pairs are generated within the same transfer direction for the data exchange between the PLC program and the C/C++ application, they should possibly all be defined in one coherent address range. Thus, it is possible to list them as one entry in the appropriate **SectorTable**. The address of the first variable must be set as the **SectorOffset** and the sum of the variable sizes as **SectorSize**. Combining the variables improves the efficiency and the performance of the copy processes.

For each entry of the **WriteSectorTable** an appropriate **SyncType** must be defined. It determines whether the section is generally taken over from the shared process image into the local image whenever there are two successive PLC cycles (**kShPIImgSyncAlways**) or whether it is taken over on demand (**kShPIImgSyncOnDemand**). If classified as **SyncOnDemand**, the data only is copied if the respective section before was explicitly marked as updated. This takes place by calling function **ShPIImgClntWriteSectMarkNewData()** and entering the corresponding **WriteSectorTable**-Index (e.g. 0 for **WriteSectTab[0]** etc.).

kShPIImgSyncAlways is provided as **SyncType** for the **ReadSectorTable** (the value of the member element **m_SyncType** is ignored). The PLC firmware is not able to identify which variables were changed by the PLC program of the cycle before. Hence, all sections defined in **ReadSectorTable** are always taken over from the local image into the shared process image. Thus, the respective variables in the shared process image always hold the actual values.

The PLC firmware and the C/C++ application both use the shared process image. To prevent conflicts due to accesses from both of those in parallel running processes at the same time, the shared process image is internally protected by a semaphore. If one process requires access to the shared process image, this process enters a critical section by setting the semaphore first and receiving exclusive access to the shared process image second. If the other process requires access to the shared process image at the same time, it also must enter a critical section by trying to set the semaphore. In this case, the operating system identifies that the shared process image is already being used. It blocks the second process until the first process leaves the critical section and releases the semaphore. Thereby, the operating system assures that only one of the two in parallel running processes (PLC runtime system and C/C++ application) may enter the critical section and receives access to the shared process image. To ensure that both processes do not interfere with each other too much, they should enter the critical section as less as possible and only as long as necessary. Otherwise, the PLC cycle time may be extended and runtime variations (Jitter) may occur.

The client application has available two functions to set the semaphore and to block exclusive access to the shared process image. Function **ShPImgClntLockSegment()** is necessary to enter the critical section and function **ShPImgClntUnlockSegment()** to leave it. The segment between both functions is called protected section, because in this segment the client application holds access to the shared process image without competition. The consistency of read or written data is only guaranteed within such a protected section. Outside the protected section, the shared process image may anytime be manipulated by the PLC runtime system. The following example shows the exclusive access to the shared process image in the C/C++ application:

```
ShPImgClntLockSegment();
{
    // write new data value into Var1
    *pbVar1 = bAnyData;

    // mark new data for WriteSectorTable entry number 0
    ShPImgClntWriteSectMarkNewData (0);
}
ShPImgClntUnlockSegment();
```

For the example above, *kShPImgSyncOnDemand* was defined as *SyncType* upon generating entry *WriteSectorTable*. Hence, taking over variable *Var1* from the shared process image into the local image can only take place if the respective section was beforehand explicitly marked as updated. Therefore, it is necessary to call function **ShPImgClntWriteSectMarkNewData()**. Since function *ShPImgClntWriteSectMarkNewData()* does not modify the semaphore, it may only be used within a protected section (see example) – such as the code section between *ShPImgClntLockSegment()* and *ShPImgClntUnlockSegment()*.

The synchronization between local image and shared process image by the PLC runtime system only takes place in-between two successive PLC cycles. A client application (user-specific C/C++ program) is not directly informed about this point of time, but it can get information about the update of the shared process image from the PLC runtime system. Therefore, the client application must define a callback handler of the type *tShPImgAppNewDataSigHandler*, e.g.:

```
static void AppSigHandlerNewData (void)
{
    fNewDataSignaled_1 = TRUE;
}
```

This callback handler must be registered with the help of function **ShPImgClntSetNewDataSigHandler()**. The handler is selected subsequent to a synchronization of the two images.

The callback handler of the client application is called within the context of a Linux signal handler (the PLC runtime system informs the client using Linux function *kill()*). Accordingly, all common **restrictions** for the Linux signal handler also apply to the callback handler of the client application. In particular, it is only allowed to call a few operating system functions that are explicitly marked as reentrant-proof. Please pay attention to not make reentrant calls of local functions within the client application. As shown in the example, only a global flag should be set for the signaling within the callback handler. This flag will later on be evaluated and processed in the main loop of the client application.

8.2 API of the shared process image client

As illustrated in Figure 31, the user-specific C/C++ application exclusively uses the API (Application Programming Interface) provided by the *shared process image client*. This API is declared in the header file *shpimgclient.h* and implemented in the source file *shpimgclient.c*. It contains the following types (partly defined in *shpimg.h*) and functions:

Structure *tShPImgLayoutDscrpt*

```
typedef struct
{
    // definition of process image sections
    unsigned int    m_uiPImgInputOffs;    // start offset of input section
    unsigned int    m_uiPImgInputSize;    // size of input section
    unsigned int    m_uiPImgOutputOffs;   // start offset of output section
    unsigned int    m_uiPImgOutputSize;   // size of output section
    unsigned int    m_uiPImgMarkerOffs;   // start offset of marker section
    unsigned int    m_uiPImgMarkerSize;   // size of marker section
} tShPImgLayoutDscrpt;
```

Structure *tShPImgLayoutDscrpt* describes the setup of the process image given by the PLC firmware. The client application receives the information about the setup of the process image via function *ShPImgClntSetup()*. This function enters start offsets and values of input, output and marker sections into the structure provided upon function calling.

Structure *tShPImgSectDscrpt*

```
typedef struct
{
    // definition of data exchange section
    unsigned int    m_uiPImgDataSectOffs;
    unsigned int    m_uiPImgDataSectSize;
    tShPImgSyncType m_SyncType;           // only used for WriteSectTab
    BOOL            m_fNewData;
} tShPImgSectDscrpt;
```

Structure *tShPImgSectDscrpt* describes the setup of a *ReadSectorTable* or *WriteSectorTable* entry that must be defined by the client. Both tables support the synchronization between the local image of the PLC runtime system and the shared process image (see section 8.1). Member element *m_uiPImgDataSectOffs* defines the absolute start offset of the section within the shared process images. The respective start offsets of the input, output and marker sections can be determined through structure *tShPImgLayoutDscrpt*. Member element *m_uiPImgDataSectSize* determines the size of the section which may include one or more variables. Member element *m_SyncType* only applies to entries of the *WriteSectorTable*. It determines whether the section is generally taken over from the shared process image into the local image whenever there are two successive PLC cycles (*kShPImgSyncAlways*) or whether it is taken over on demand (*kShPImgSyncOnDemand*). If classified as *SyncOnDemand*, the data must be marked as modified by calling function *ShPImgClntWriteSectMarkNewData()*. It sets the member element *m_fNewData* to TRUE. The client application should never directly modify this member element.

Function ShPImgClntSetup

```
BOOL ShPImgClntSetup (tShPImgLayoutDscrpt* pShPImgLayoutDscrpt_p);
```

Function **ShPImgClntSetup()** initializes the *shared process image client* and connects itself with the storage segment for the shared process image which is generated by the PLC runtime system. Afterwards, it enters the start offsets and values of the input, output and marker sections into the structure of type *tShPImgLayoutDscrpt* provided upon function call. Hence, the client application receives notice about the process image setup managed by the PLC firmware.

If the PLC runtime system is not active when the function is called or if it has not generated a shared process image (option "*Share PLC process image*" in the PLC configuration deactivated, see section 8.1), the function will return with the return value FALSE. If the initialization was successful, the return value will be TRUE.

Function ShPImgClntRelease

```
BOOL ShPImgClntRelease (void);
```

Function **ShPImgClntRelease()** shuts down the *shared process image client* and disconnects the connection to the storage segment generated for the shared process image by the PLC runtime system.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

Function ShPImgClntSetNewDataSigHandler

```
BOOL ShPImgClntSetNewDataSigHandler (
    tShPImgAppNewDataSigHandler pfnShPImgAppNewDataSigHandler_p);
```

Function **ShPImgClntSetNewDataSigHandler()** registers a user-specific callback handler. This callback handler is called after a synchronization of both images. Registered callback handlers are cleared by the parameter NULL.

The **callback handler is called within the context of a Linux signal handler**. Accordingly, all common **restrictions** for the Linux signal handler also apply to the callback handler (see section 8.1).

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

Function ShPImgClntGetHeader

```
tShPImgHeader* ShPImgClntGetHeader (void);
```

Function **ShPImgClntGetHeader()** provides a pointer to the internally used structure type *tShPImgHeader* to manage the shared process image. The client application does usually not need this structure, because all data that it includes can be read and written through functions of the API provided by the *shared process image client*.

Function *ShPImgClntGetDataSect*

```
BYTE* ShPImgClntGetDataSect (void);
```

Function ***ShPImgClntGetDataSect()*** provides a pointer to the beginning of the shared process image. This pointer represents the basic address for all accesses to the shared process image; including the definition of sections *ReadSectorTable* and *WriteSectorTable* (see section 8.1).

Funktionen Functions *ShPImgClntLockSegment* and *ShPImgClntUnlockSegment*

```
BOOL ShPImgClntLockSegment (void);  
BOOL ShPImgClntUnlockSegment (void);
```

To exclusively access the shared process image, the client application has available two functions - function ***ShPImgClntLockSegment()*** to enter the critical section and function ***ShPImgClntUnlockSegment()*** to leave it. The segment between both functions is called protected section, because in this segment the client application holds unrivaled access to the shared process image (see section 8.1). The consistency of read or written data is only guaranteed within such a protected section. Outside the protected section, the shared process image may anytime be manipulated by the PLC runtime system. To ensure that the client application does not interfere with the PLC runtime system too much, the critical sections should be set as less as possible and only as long as necessary. Otherwise, the PLC cycle time may be extended and runtime variations (Jitter) may occur.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

Function *ShPImgClntSetupReadSectTable*

```
BOOL ShPImgClntSetupReadSectTable (  
    tShPImgSectDscrpt* paShPImgReadSectTab_p,  
    unsigned int uiNumOfReadDscrptUsed_p);
```

Function ***ShPImgClntSetupReadSectTable()*** initializes the *ReadSectorTable* with the values defined by the client. The client hereby determines those sections of the PLC process image from which it wants to read data (see section 8.1). Parameter *paShPImgReadSectTab_p* holds elements of the structure *tShPImgSectDscrpt* and must be transferred as start address of a section. Parameter *uiNumOfReadDscrptUsed_p* indicates how many elements the section has.

kShPImgSyncAlways is provided as *SyncType* for the *ReadSectorTable*.

The maximum amount of possible elements for the *ReadSectorTable* is defined by the constant *SHPIMG_READ_SECT_TAB_ENTRIES* and can only be modified if the shared library "*pmc32drv.so*" is generated again and at the time (this requires SO-1098 - "Driver Development Kit for the ECUcore-5484").

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

Function ShPImgClntSetupWriteSectTable

```

BOOL ShPImgClntSetupWriteSectTable (
    tShPImgSectDscrpt* paShPImgWriteSectTab_p,
    unsigned int uiNumOfWriteDscrptUsed_p);

```

Function **ShPImgClntSetupWriteSectTable()** initializes the *WriteSectorTable* with the values defined by the client. The client hereby determines those sections of the PLC process image from which it wants to write data (see section 8.1). Parameter *paShPImgWriteSectTab_p* holds elements of structure *tShPImgSectDscrpt* and must be transferred as start address of a section. Parameter *uiNumOfWriteDscrptUsed_p* indicates how many elements the section has.

For each entry in the *WriteSectorTable* the *SyncType* must be defined. This *SyncType* defines whether the section is always taken over into the local image between two PLC cycles (**kShPImgSyncAlways**) or only on demand (**kShPImgSyncOnDemand**). If taken over on demand, the respective section is explicitly marked as updated by calling *ShPImgClntWriteSectMarkNewData()*.

The maximum amount of possible elements for the *WriteSectorTable* is defined by the constant *SHPIMG_WRITE_SECT_TAB_ENTRIES* and can only be modified if the shared library "pmc32drv.so" is generated again and at the same time (this requires SO-1098 - "Driver Development Kit for the ECUcore-5484").

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

Function ShPImgClntWriteSectMarkNewData

```

BOOL ShPImgClntWriteSectMarkNewData (unsigned int uiWriteDscrptIdx_p);

```

For the content of a section that is held by the *WriteSectorTable*, function **ShPImgClntWriteSectMarkNewData()** marks this content as modified. This function is used (for sections with *SyncType* **kShPImgSyncOnDemand**) to initiate the copy process of data from the shared process image into the local image of the PLC.

Function *ShPImgClntWriteSectMarkNewData()* directly accesses the header of the shared process image without setting a semaphore before. Hence, it may only be used within the protected section – in the code section between *ShPImgClntLockSegment()* and *ShPImgClntUnlockSegment()*.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

8.3 Creating a user-specific client application

Software package SO-1095 ("VMware image of the Linux development system") is the precondition for the implementation of user-specific C/C++ applications. It contains a complete Linux development system in the form of a VMware image. Hence, it allows for an easy introduction into the C/C++ software development for the PLCmodule-C32. Thus, the VMware image is the ideal basis to develop Linux-based user programs on the same host PC that already has the *OpenPCS* IEC 61131 programming system installed on it. The VMware image of the Linux development system includes the GNU-Crosscompiler Toolchain for Freescale MCF54xx processors. Additionally, it includes essential server services that are preconfigured and usable for effective software development. Details about the VMware image of the Linux development system and instructions for its usage are described in the "System Manual ECUcore-5484" (Manual no: L-1102).

As illustrated in Figure 31, the user-specific C/C++ application uses the API (files *shpimgclient.c* and *shpimgclient.h*) which is provided by the *shared process image client*. The *shared process image client* is based on services provided by the *shared memory client* (files *shmclient.c* and *shmclient.h*). Both client implementations are necessary to generate a user-specific C/C++ application. The archive of the *shared process image demos* (***shpimgdemo.tar.gz***) contains the respective files. To create own user-specific client applications, it is recommended to use this demo project as the basis for own adaptations and extensions. Moreover, this demo project contains a Makefile with all relevant configuration adjustments that are necessary to create a Linux application for the PLCmodule-C32. Table 19 lists all files of the archive "*shpimgdemo.tar.gz*" and classifies those as general part of the C/C++ application or as specific component for the demo project "*shpimgdemo*".

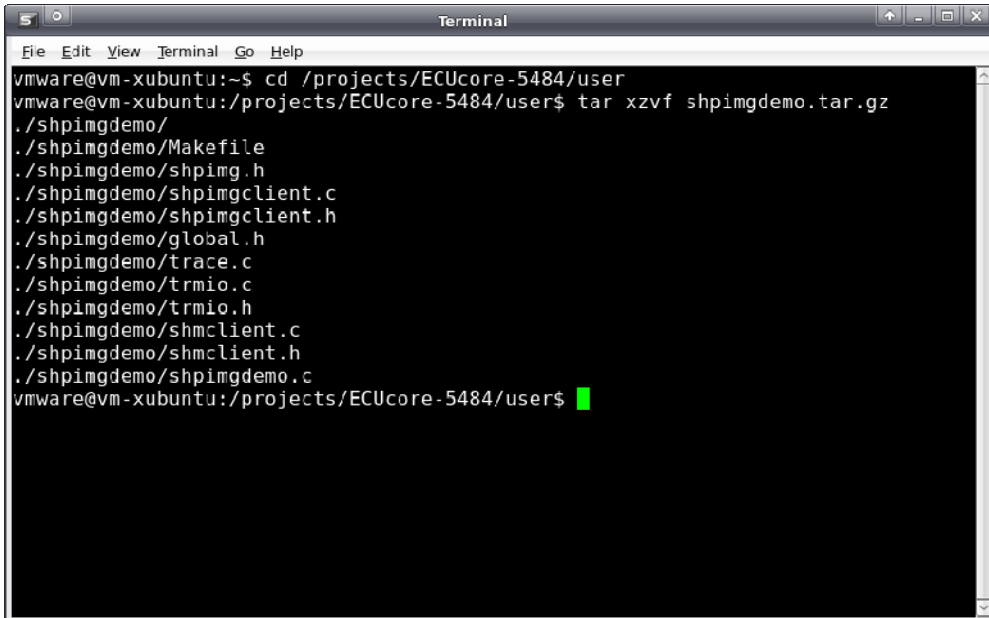
Table 19: Content of the archive files "*shpimgdemo.tar.gz*"

File	Necessary for all C/C++ applications	In particular for demo " <i>shpimgdemo</i> "
<i>shpimgclient.c</i>	x	
<i>shpimgclient.h</i>	x	
<i>shmclient.c</i>	x	
<i>shmclient.h</i>	x	
<i>shpimg.h</i>	x	
<i>global.h</i>	x	
Makefile	draft, to be adjusted	
<i>shpimgdemo.c</i>		x
<i>trmio.c</i>		x
<i>trmio.h</i>		x
<i>trace.c</i>		x

The archive file "***shpimgdemo.tar.gz***" including the *shared process image demo* must be unzipped into any subdirectory following the path "*/projects/ECUcore-5484/user*" within the Linux development system. Therefore, command "*tar*" must be called:

```
tar xzvf shpimgdemo.tar.gz
```

During the unzipping process, command "*tar*" independently generates the subdirectory "*shpimgdemo*". For example, if the command is called in directory "*/projects/ECUcore-5484/user*", all archive files will be unzipped into the path "*/projects/ECUcore-5484/user/shpimgdemo*". Figure 32 exemplifies the unzipping process of "*shpimgdemo.tar.gz*" within the Linux development system.

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Go, Help). The terminal shows the following commands and output:

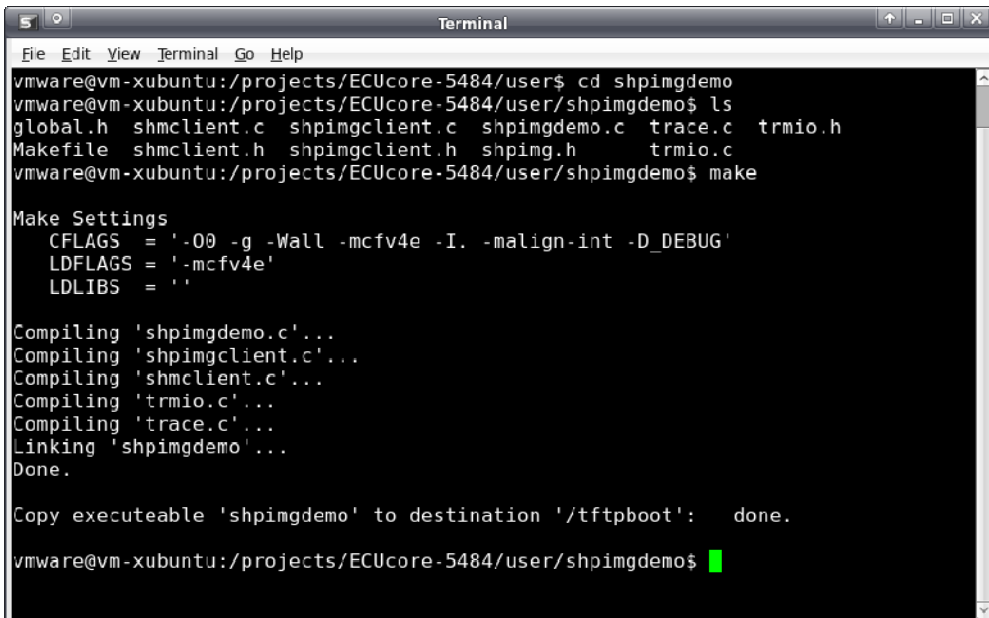
```
vmware@vm-xubuntu:~$ cd /projects/ECUcore-5484/user
vmware@vm-xubuntu:/projects/ECUcore-5484/user$ tar xzvf shpimgdemo.tar.gz
./shpimgdemo/
./shpimgdemo/Makefile
./shpimgdemo/shping.h
./shpimgdemo/shpingclient.c
./shpimgdemo/shpingclient.h
./shpimgdemo/global.h
./shpimgdemo/trace.c
./shpimgdemo/trmio.c
./shpimgdemo/trmio.h
./shpimgdemo/shmclient.c
./shpimgdemo/shmclient.h
./shpimgdemo/shpingdemo.c
vmware@vm-xubuntu:/projects/ECUcore-5484/user$
```

Figure 32: Unzipping the archive files *shpimgdemo.tar.gz* in the Linux development system

After unzipping and switching into subdirectory "*shpimgdemo*", the demo project can be created by calling command "*make*":

```
cd shpimgdemo
make
```

Figure 33 shows how the demo project "*shpimgdemo*" is generated in the Linux development system.

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Go, Help). The terminal shows the following commands and output:

```
vmware@vm-xubuntu:/projects/ECUcore-5484/user$ cd shpimgdemo
vmware@vm-xubuntu:/projects/ECUcore-5484/user/shpimgdemo$ ls
global.h  shmclient.c  shpingclient.c  shpingdemo.c  trace.c  trmio.h
Makefile  shmclient.h  shpingclient.h  shping.h      trmio.c
vmware@vm-xubuntu:/projects/ECUcore-5484/user/shpimgdemo$ make

Make Settings
  CFLAGS = '-O0 -g -Wall -mcfv4e -I. -malign-int -D_DEBUG'
  LDFLAGS = '-mcfv4e'
  LDLIBS = ''

Compiling 'shpingdemo.c'...
Compiling 'shpingclient.c'...
Compiling 'shmclient.c'...
Compiling 'trmio.c'...
Compiling 'trace.c'...
Linking 'shpingdemo'...
Done.

Copy executable 'shpingdemo' to destination '/tftpboot': done.

vmware@vm-xubuntu:/projects/ECUcore-5484/user/shpimgdemo$
```

Figure 33: Generating the demo project "*shpimgdemo*" in the Linux development system

Section 8.4 describes the usage and handling of the demo project "*shpimgdemo*" on the PLCmodule-C32.

8.4 Example for using the shared process image

The demo project "shpimgdemo" (described in section 8.3) in connection with the PLC program example "RunLight" both exemplify the data exchange between a PLC program and a user-specific C/C++ application.

Technical background

The PLC program generates some variables in the process image as directly addressable variables. In a C/C++ application, all those variables are usable via the shared process image. For the PLC program example "RunLight" those are the following variables:

```
(* variables for local control via on-board I/O's *)
bButtonGroup      AT %IB0.0   : BYTE;
iAnalogValue      AT %IW8.0   : INT;
bLEDGroup0       AT %QB0.0   : BYTE;
bLEDGroup1       AT %QB1.0   : BYTE;

(* variables for remote control via shared process image *)
uiRemoteSliderLen AT %MW512.0 : UINT;      (* out: length of sliderbar      *)
bRemoteStatus    AT %MB514.0 : BYTE;      (* out: Bit0: RemoteControl=on/off *)
bRemoteDirCtrl   AT %MB515.0 : BYTE;      (* in: direction left/right      *)
iRemoteSpeedCtrl AT %MW516.0 : INT;       (* in: speed                      *)
```

Variables of the PLC program are accessible from a C/C++ application via the shared process image. Therefore, sections must be generated for the *ReadSectorTable* and *WriteSectorTable* on the one hand and on the other hand, pointers must be defined for accessing the variables. The following program extract shows this using the example "shpimgdemo.c". Function *ShPIImgClntSetup()* inserts the start offsets of input, output and marker sections into the structure *ShPIImgLayoutDscrpt*. Hence, on the basis of the initial address provided by *ShPIImgClntGetDataSect()*, the absolute initial addresses of each section in the shared process image can be determined. To identify the address of a variable, the variable's offset within the particular section must be added. For example, the absolute address to access the variable "bRemoteDirCtrl AT %MB515.0 : BYTE;" results from the sum of the initial address of the shared process image (*pabShPIImgDataSect*), the start offset of the marker section (*ShPIImgLayoutDscrpt.m_uiPIImgMarkerOffs* für "%M...") as well as the direct address within the marker section which was defined in the PLC program (515 for "%MB515.0"):

```
pabPIImgVar_61131_bDirCtrl = (BYTE*) (pabShPIImgDataSect
    + ShPIImgLayoutDscrpt.m_uiPIImgMarkerOffs + 515);
```

The following code extract shows the complete definition of all variables in the demo project used for exchanging data with the PLC program:

```
// ---- Setup shared process image client ----
fRes = ShPIImgClntSetup (&ShPIImgLayoutDscrpt);
if ( !fRes )
{
    printf ("\n*** ERROR *** Init of shared process image client failed");
}

pabShPIImgDataSect = ShPIImgClntGetDataSect();
```

```

// ---- Read Sector Table ----
// Input Section:      bButtonGroup AT %IB0.0
{
    ShPImgReadSectTab[0].m_uiPImgDataSectOffs =
        ShPImgLayoutDscrpt.m_uiPImgInputOffs + 0;
    ShPImgReadSectTab[0].m_uiPImgDataSectSize = sizeof(BYTE);
    ShPImgReadSectTab[0].m_SyncType
        = kShPImgSyncAlways;

    pbPImgVar_61131_bButtonGroup = (BYTE*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgInputOffs + 0);
}

// Output Section:    bLEDGroup0 AT %QB0.0
//                   bLEDGroup1 AT %QB1.0
{
    ShPImgReadSectTab[1].m_uiPImgDataSectOffs =
        ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 0;
    ShPImgReadSectTab[1].m_uiPImgDataSectSize = sizeof(BYTE) + sizeof(BYTE);
    ShPImgReadSectTab[1].m_SyncType
        = kShPImgSyncAlways;

    pbPImgVar_61131_bLEDGroup0 = (BYTE*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 0);
    pbPImgVar_61131_bLEDGroup1 = (BYTE*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 1);
}

// Marker Section:   uiSlidbarLen AT %MW512.0
//                   bStatus      AT %MB514.0
{
    ShPImgReadSectTab[2].m_uiPImgDataSectOffs =
        ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 512;
    ShPImgReadSectTab[2].m_uiPImgDataSectSize = sizeof(unsigned short int)
        + sizeof(BYTE);
    ShPImgReadSectTab[2].m_SyncType
        = kShPImgSyncAlways;

    pbPImgVar_61131_usiSlidbarLen = (unsigned short int*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 512);
    pbPImgVar_61131_bStatus = (BYTE*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 514);
}

fRes = ShPImgClntSetupReadSectTable (ShPImgReadSectTab, 3);
if ( !fRes )
{
    printf ("\n*** ERROR *** Initialization of read sector table failed");
}

// ---- Write Sector Table ----
// Marker Section:    bDirCtrl   AT %MB513.0
//                   iSpeedCtrl AT %MB514.0
{
    ShPImgWriteSectTab[0].m_uiPImgDataSectOffs =
        ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 515;
    ShPImgWriteSectTab[0].m_uiPImgDataSectSize = sizeof(BYTE) + sizeof(WORD);
    ShPImgWriteSectTab[0].m_SyncType
        = kShPImgSyncOnDemand;

    pbPImgVar_61131_bDirCtrl = (BYTE*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 515);
    psiPImgVar_61131_iSpeedCtrl = (short int*) (pabShPImgDataSect
        + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 516);
}

fRes = ShPImgClntSetupWriteSectTable (ShPImgWriteSectTab, 1);
if ( !fRes )
{
    printf ("\n*** ERROR *** Initialization of write sector table failed");
}

```

Realization on the PLCmodule-C32

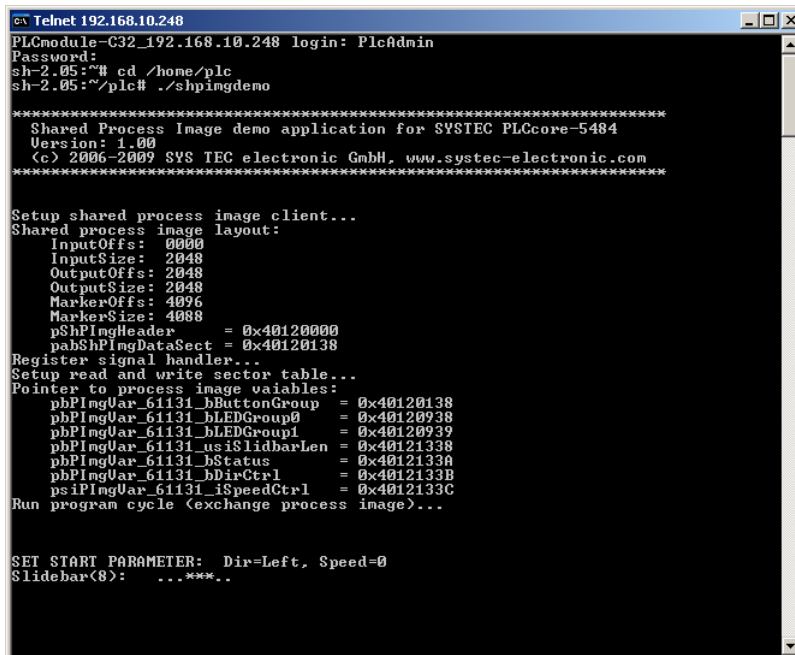
To enable the execution of the *shared process image demo* without previous introduction into the Linux-based C/C++ programming for the PLCmodule-C32, the module comes with a preinstalled, translated and ready-to-run program version and PLC firmware ("*/home/plc/shpimgdemo*"). The following description refers to this program version. Alternatively, the demo project can be newly-generated from the corresponding source files (see section 8.3) and can be started afterwards. As I/O-Simulator for practical controlling of the demo-program an I/O-Box is available from SYS TEC (Ordering code: phyPS-451, see also Chapter .4.1).

The following steps are necessary to run the *shared process image demo* on the PLCmodule-C32:

1. **Activate option "Shared PLC process image"** in the PLC configuration (see sections 8.1, 7.4.1 and 7.4.3).
2. Open the PLC program example "*RunLight*" in the *OpenPCS* IEC 61131 programming system und build the project for a target hardware of the type "*SYSTEC - PLCmodule-C32*"
3. Select the network connection to the PLCmodule-C32 und download the program.
4. Start the PLC program on the PLCmodule-C32
5. Login to the command shell of the PLCmodule-C32 as described in section 7.8.1.
6. Switch to the directory "*/home/plc*" and call the demo program "*shpimgdemo*".

```
cd /home/plc
./shpimgdemo
```

The digital outputs of the PLCmodule-C32 are selected as runlight. The speed is modifiable via the analog input AI0. With the help of digital inputs DI0 and DI1, the running direction can be changed. After starting the demo program "*shpimgdemo*" on the PLCmodule-C32, actual status information about the runlight is indicated cyclically in the terminal (see Figure 34).



```

Telnet 192.168.10.248
PLCmodule-C32_192.168.10.248 login: PlcAdmin
Password:
sh-2.05:~# cd /home/plc
sh-2.05:~/plc# ./shpimgdemo
*****
  Shared Process Image demo application for SYSTEC PLCcore-5484
  Version: 1.00
  (c) 2006-2009 SYS TEC electronic GmbH, www.systec-electronic.com
*****
Setup shared process image client...
Shared process image layout:
  InputOffs: 0000
  InputSize: 2048
  OutputOffs: 2048
  OutputSize: 2048
  MarkerOffs: 4096
  MarkerSize: 4088
  pShPingHeader = 0x40120000
  pabShPingDataSect = 0x40120138
Register signal handler...
Setup read and write sector table...
Pointer to process image variables:
  pbPingVar_61131_bButtonGroup = 0x40120138
  pbPingVar_61131_bLEDGroup0 = 0x40120938
  pbPingVar_61131_bLEDGroup1 = 0x40120939
  pbPingVar_61131_usiSlidbarLen = 0x40121338
  pbPingVar_61131_bStatus = 0x4012133A
  pbPingVar_61131_bDirCtrl = 0x4012133B
  psIPingVar_61131_iSpeedCtrl = 0x4012133C
Run program cycle (exchange process image)...

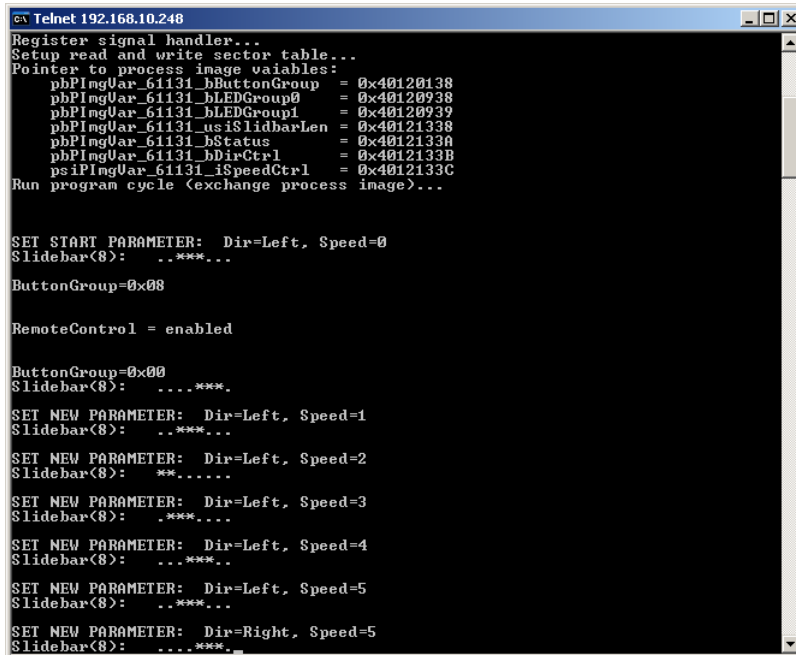
SET START PARAMETER: Dir=Left, Speed=0
Slidebar(8): ...***..

```

Figure 34: Terminal outputs of the demo program "*shpimgdemo*" after start

7. By activating of digital input DI3, the control of the runlight direction and speed is handed over to the demo program "*shpimgdemo*". Afterwards, the running direction may be set by the C

application by using the cursor pushbuttons left and right (← und →) in the terminal window and the speed may be changed by using cursor pushbuttons up and down (↑ und ↓).



```
ca Telnet 192.168.10.248
Register signal handler...
Setup read and write sector table...
Pointer to process image variables:
  pbPingVar_61131_bButtonGroup = 0x40120138
  pbPingVar_61131_bLEDGroup0   = 0x40120938
  pbPingVar_61131_bLEDGroup1   = 0x40120939
  pbPingVar_61131_usiSliderLen = 0x40121338
  pbPingVar_61131_bStatus     = 0x4012133A
  pbPingVar_61131_bDirCtrl    = 0x4012133B
  psiPingVar_61131_iSpeedCtrl = 0x4012133C
Run program cycle (exchange process image)...

SET START PARAMETER: Dir=Left, Speed=0
Slider(8): ..***.

ButtonGroup=0x00

RemoteControl = enabled

ButtonGroup=0x00
Slider(8): ....***.

SET NEW PARAMETER: Dir=Left, Speed=1
Slider(8): ..***.

SET NEW PARAMETER: Dir=Left, Speed=2
Slider(8): **.....

SET NEW PARAMETER: Dir=Left, Speed=3
Slider(8): ..***.

SET NEW PARAMETER: Dir=Left, Speed=4
Slider(8): ..***.

SET NEW PARAMETER: Dir=Left, Speed=5
Slider(8): ..***.

SET NEW PARAMETER: Dir=Right, Speed=5
Slider(8): ..***.
```

Figure 35: Terminal outputs of the demo program "shpimgdemo" after user inputs

Figure 35 shows the terminal outputs of the demo program "shpimgdemo" in answer to activating the cursor pushbuttons.

The demo program "shpimgdemo" may be terminated by pressing "Ctrl+C" in the terminal window.

Appendix A: Firmware function scope of PLCmodule-C32

Table 20 lists all firmware functions and function blocks available on the PLCcore-5484.

Sign explanation:

FB	Function block
FUN	Function
Online Help	<i>OpenPCS</i> online help
L-1054	Manual "SYS TEC-specific extensions for <i>OpenPCS</i> / IEC 61131-3", Manual no.: L-1054)
PARAM:={0,1,2}	values 0, 1 and 2 are valid for the given parameter

Table 20: Firmware functions and function blocks of PLCmodule-C32

Name	Type	Reference	Remark
PLC standard Functions and Function Blocks			
SR	FB	Online Help	
RS	FB	Online Help	
R_TRIG	FB	Online Help	
F_TRIG	FB	Online Help	
CTU	FB	Online Help	
CTD	FB	Online Help	
CTUD	FB	Online Help	
TP	FB	Online Help	
TON	FB	Online Help	
TOF	FB	Online Help	
Functions and Function Blocks for string manipulation			
LEN	FUN	L-1054	
LEFT	FUN	L-1054	
RIGHT	FUN	L-1054	
MID	FUN	L-1054	
CONCAT	FUN	L-1054	
INSERT	FUN	L-1054	
DELETE	FUN	L-1054	
REPLACE	FUN	L-1054	
FIND	FUN	L-1054	
GETSTRINFO	FB	L-1054	
CHR	FUN	L-1054	
ASC	FUN	L-1054	
STR	FUN	L-1054	
VAL	FUN	L-1054	
Functions and Function Blocks for OpenPCS specific task controlling			
ETRC	FB	L-1054	
PTRC	FB	L-1054	
GETVARDATA	FB	Online Help	
GETVARFLATADDRESS	FB	Online Help	
GETTASKINFO	FB	Online Help	

Functions and Function Blocks for handling of non-volatile data			
NVDATA_BIT	FB	L-1054	DEVICE:={0}, see ⁽¹⁾
NVDATA_INT	FB	L-1054	DEVICE:={0}, see ⁽¹⁾
NVDATA_STR	FB	L-1054	DEVICE:={0}, see ⁽¹⁾
Functions and Function Blocks for handling of time			
GetTime	FUN	Online Help	
GetTimeCS	FUN	Online Help	
DT_CLOCK	FB	L-1054	
DT_ABS_TO_REL	FB	L-1054	
DT_REL_TO_ABS	FB	L-1054	
Functions and Function Blocks for counter inputs and pulse outputs			
CNT_FUD	FB	L-1054	CHANNEL:={0,1,2}
PTO_PWM	FB	L-1054	CHANNEL:={0,1}
PTO_TAB	FB	L-1054	CHANNEL:={0,1}
Functions and Function Blocks for Serial interfaces			
SIO_INIT	FB	L-1054	PORT:={0,1,2}, see ⁽²⁾
SIO_STATE	FB	L-1054	PORT:={0,1,2} see ⁽²⁾
SIO_READ_CHR	FB	L-1054	PORT:={0,1,2} see ⁽²⁾
SIO_WRITE_CHR	FB	L-1054	PORT:={0,1,2} see ⁽²⁾
SIO_READ_STR	FB	L-1054	PORT:={0,1,2} see ⁽²⁾
SIO_WRITE_STR	FB	L-1054	PORT:={0,1,2} see ⁽²⁾
Functions and Function Blocks for CAN interfaces / CANopen			
CAN_GET_LOCALNODE_ID	FB	L-1054	NETNUMBER:={0,1}
CAN_CANOPEN_KERNEL_STATE	FB	L-1054	NETNUMBER:={0,1}
CAN_REGISTER_COBID	FB	L-1054	NETNUMBER:={0,1}
CAN_PDO_READ8	FB	L-1054	NETNUMBER:={0,1}
CAN_PDO_WRITE8	FB	L-1054	NETNUMBER:={0,1}
CAN_SDO_READ8	FB	L-1054	NETNUMBER:={0,1}
CAN_SDO_WRITE8	FB	L-1054	NETNUMBER:={0,1}
CAN_SDO_READ_STR	FB	L-1054	NETNUMBER:={0,1}
CAN_SDO_WRITE_STR	FB	L-1054	NETNUMBER:={0,1}
CAN_GET_STATE	FB	L-1054	NETNUMBER:={0,1}
CAN_NMT	FB	L-1054	NETNUMBER:={0,1}
CAN_RECV_EMCY_DEV	FB	L-1054	NETNUMBER:={0,1}
CAN_RECV_EMCY	FB	L-1054	NETNUMBER:={0,1}
CAN_WRITE_EMCY	FB	L-1054	NETNUMBER:={0,1}
CAN_RECV_BOOTUP_DEV	FB	L-1054	NETNUMBER:={0,1}
CAN_RECV_BOOTUP	FB	L-1054	NETNUMBER:={0,1}
CAN_ENABLE_CYCLIC_SYNC	FB	L-1054	NETNUMBER:={0,1}
CAN_SEND_SYNC	FB	L-1054	NETNUMBER:={0,1}

Functions and Function Blocks for Ethernet interfaces / UDP

LAN_GET_HOST_CONFIG	FB	L-1054	NETNUMBER:={0}
LAN_ASCII_TO_INET	FB	L-1054	NETNUMBER:={0}
LAN_INET_TO_ASCII	FB	L-1054	NETNUMBER:={0}
LAN_GET_HOST_BY_NAME	FB	L-1054	NETNUMBER:={0}
LAN_GET_HOST_BY_ADDR	FB	L-1054	NETNUMBER:={0}
LAN_UDP_CREATE_SOCKET	FB	L-1054	NETNUMBER:={0}
LAN_UDP_CLOSE_SOCKET	FB	L-1054	NETNUMBER:={0}
LAN_UDP_RECVFROM_STR	FB	L-1054	NETNUMBER:={0}
LAN_UDP_SENDTO_STR	FB	L-1054	NETNUMBER:={0}

- (1) All nonvolatile data is filed into directory *"/home/plc/PlcPData.bin"* on the PLCmodule-C32. This file has a fix size of 32 kByte. By calling function blocks of type *NVDATA_Xxx* in a writing mode, the modified data is directly stored into file *"/home/plc/PlcPData.bin"* ("*flush*"). Thus, unsecured data is not getting lost in case of power interruption.
- (2) Interface ASC0 resp. COM0 (PORT:=0) primarily serves as service interface to administer the PLCmodule-C32. Hence, this interface should only be used for sign output. The module always tries to interpret and execute sign inputs as Linux commands (see section 6.5.1).

Appendix B: Technical Specification

Environmental Parameters		Typical	Maximum
Power Supply	V_{CPU}	24VDC	-15%, +20%
	V_{IO}	24VDC	-15%, +20%
	power fail level	20V	
	power fail delay time	4ms	
Current Consumption (inactive IOs)	I_{CPU}	155mA	
	I_{IO}	30mA	
Temperature Range	Storage temperature		-20..+70°C
	Operating temperature		0..+50°C
Protection class	Housing	IP20	
Weight	without any cable and packing	325g	
Dimensions	Width		160mm
	Height		75mm
	Depth		90mm
Connector type	Spring type connector		

RTC		Typical	Maximum
RTC	$I_{Power\ Down}$ at $V_{BAT} = 3V$	275nA	700nA
	$I_{Batt\ Low-Detection}$	150nA	175nA
	capacity		220mAh
	Clock Voltage Range	0.9	1.0V

I/O-configuration (digital)		Typical	Maximum
Digital Outputs DO0 .. 15			
24VDC-Output (High Side Switch)	U_{OH} at $I_{OH} = 500\text{ mA}$	$V_{IO} - 0,16V < U_{OH} < V_{IO}$	
	U_{OL} at $I_{OL} = 0\text{ mA}$		0.5V
	Current limitation $I_{OH\ max}$		625mA
	Max. current		tbd2x5A
	$I_{OL(off)}$		10µA
	t_{off} at $I_{OH} = 500\text{ mA}$	115µs	190µs
t_{on} at $I_{OH} = 500\text{ mA}$	75µs	125µs	
Digital Outputs P0 .. 1			
24VDC-PWM-Output (Low Side Switch)	U_{OL} at $I_{OL} = -500\text{mA}$		<1V
	$I_{OH(off)}$		20µA
	$I_{OH\ max}$		0.6A
	t_{on} at $I_{OL} = -500\text{mA}$		2.5µs
	t_{off} at $I_{OL} = -500\text{mA}$		3.5µs
Digital Outputs REL0 .. 3			
Relay output (N.O.)	Switching Voltage		250AC
	Switching Current		5A
	Durability (mech.) (5A/250VAC/ohmic load)	100000x	
	Durability (elec.) (2A/250VAC/cosφ0,4)	200000x	
	t_{on}	5ms	
	t_{off}	2.5ms	
Isolation	1000Vrms		
Digital Inputs DI0 .. 23			
24VDC- Inputs, plus switching	U_{IH}	15V	30V

	U_{IL}	-3V	5V
	I_{IH}	3mA	8,5mA
Counter Inputs C0 .. 2			
24VDC- Inputs, plus switching	U_{IH}	15V	30V
	U_{IL}	-3V	5V
	I_{IH}	3mA	8,5mA
	Frequency		70kHz

I/O-configuration (analog)		Typical	Maximum
Analog Inputs AI0 .. 3			
0 .. +10V	Measurement range U_I	0 - +10V	
	Destructive voltage $U_{I \max}$		>30V
	Input resistance R_I	59k Ω \pm 0.1%	
	Reference voltage U_{REF}	4,528V	\pm 0.6%
	Physical Resolution		10Bit
0 .. +20mA (optional mounting)	Measurement range U_I	0 - +20mA	
	Destructive voltage $U_{I \max}$		>35mA
	Input resistance R_I	200 Ω \pm 0.1%	
	Reference voltage U_{REF}	4,528V	\pm 0.6%
	Physical Resolution		10Bit
Analog Outputs AO 0 .. 1			
0 .. +10V	Voltage Range U_O	0 – +10V	
	Output current I_O		30mA
	Output capacity		10nF
	Reference voltage U_{REF}	4.528V	\pm 0,6%
	Internal Resolution		10 Bit

Communication Interfaces		Minimum	Maximum
CAN-Bus			
CAN1, CAN2	Baudrate	5kBaud	1Mbaud
	Max. number of nodes		64
	Transceiver	PCA82C251	
	CAN-H, CAN-L short-circuit-proof towards 24V		
RS-232			
ASC0	Baudrate	1200Baud	115200Baud
ASC1	Baudrate	1200Baud	115200Baud
ASC2	Baudrate	1200Baud	115200Baud
Ethernet			
10Base-T/100Base-TX	Baudrate	10Mbit/s	100Mbit/s

Appendix C: GNU GENERAL PUBLIC LICENSE

The Embedded Linux used on PLCmodule-C32 is licensed under GNU General Public License, version 2. The entire license text is specified below. A German translation is available from <http://www.gnu.de/documents/gpl-2.0.de.html>. Be advised that this translation is not official or legally approved.

The PLC system used and the PLC and C/C++ programs developed by the user are **not** subject to the GNU General Public License!

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software -- to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>  
Copyright (C) <year> <name of author>
```

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author  
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.  
This is free software, and you are welcome to redistribute it under certain conditions;  
type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items -- whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

```
<signature of Ty Coon>, 1 April 1989  
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index

/	
/home	49
/home/etc/autostart	20, 41
/home/plc/plcmodule-c32.cfg	37
/home/plc/PlcPData.bin	71
/tmp	49, 50
A	
Accessory	10, 11
adduser	47
Administration	
System Requirements	32
AI0 ... AI3	17
Analog inputs	17
Analog outputs	18
AO0 ... AO1	18
ASC	23
ASC0	19, 23
ASC1	19, 23
ASC2	19, 24
autostart	20, 41
AWL	9
B	
Bitrate	39
Bitrate CAN0	39
Boot conditions	33
Boot configuration	41
C	
C0 ... C2	15
CAN0	24, 28
CAN0, CAN1	19
CAN1	24, 29
Network variables	23
PLC program example	31
CANopen	9, 27
CANopen IO modules	9
CANopen Master	9
CE conformity	5
CFG File	39
CoLilo command	
BoardID configuration	42
CoLilo Command	
Update Linux Image	53
CoLilo Command Prompt	
Activation	33
Terminal Configuration	34
CoLilo Commands	
Ethernet Configuration	34
COM	23
Communication FB	21
Communication interfaces	
ASC	18, 23
CAN	24
CAN	19
COM	18, 23
ETH	24
Ethernet	19
ConfigCAN1	31
Configuration	
CAN0	38
Command	34
PLC	36
Configuration Mode	33
Control Elements	
Error-LED	26
Run/Stop switch	25
Run-LED	25
Counter inputs	15, 24
D	
date	48
Deletion of PLC Program	27
deluser	47
DI0 ... DI23	15
Digital inputs	15
Digital outputs	16
Dimensions	8
DIP Switch	38
DO0 ... DO23	16
Driver Development Kit	11
E	
Embedded Linux	9
EMC law	5
Environmental Parameters	72
Error-LED	26
ETH0	19, 24
PLC program example	24
F	
File System	49
Firmware version	
Selection	42
FTP	
Login to the PLCmodule-C32	45
FTP Client	32
FUB	9
G	
GNU	9
GPL	74
H	
Hex-Encoding Switch	38
hwclock	48
I	
IO Box	10

K		Setting the System Time.....	48
KOP	9	Shared Process Image	
L		Activation.....	56
limiting values	72	API Description.....	59
Linux	9	Example	65
M		Overview	55
Manuals		signaling	58
Overview	6	Variable Pairs	56
Master Mode.....	39	<i>ShPImgClntGetDataSect</i>	61
Master Mode CAN0	39	<i>ShPImgClntGetHeader</i>	60
N		<i>ShPImgClntLockSegment</i>	61
Node Address.....	39	<i>ShPImgClntRelease</i>	60
Node Address CAN0	39	<i>ShPImgClntSetNewDataSigHandler</i>	60
O		<i>ShPImgClntSetup</i>	60
<i>OpenPCS</i>	9	<i>ShPImgClntSetupReadSectTable</i>	61
P		<i>ShPImgClntSetupWriteSectTable</i>	62
P0 ... P1	17	<i>ShPImgClntUnlockSegment</i>	61
passwd.....	47	<i>ShPImgClntWriteSectMarkNewData</i>	62
Pin assignment.....	12	<i>shpimgdemo</i>	63
PLC program example		<i>shpimgdemo.tar.gz</i>	63
CAN1.....	31	SO-1095.....	62
ETH0.....	24	Software Update	
PLCcore-5484	9	PLC Firmware	50
plcmodule-c32.cfg	37, 39, 52	Softwareupdate	
PlcPData.bin	71	Linux Image.....	52
Power supply	14	ST.....	9
VCPU	14	System Start	20
VIO	15	sysWORXX Automation Series	9
Power Supply	72	T	
Predefined User Accounts.....	43	Technical Specifications	72
Process Image		Telnet	
Layout and Addressing	22	Login to the PLCmodule-C32.....	44
Programming	21	Telnet Client.....	32
Pulse outputs.....	25	Terminal Configuration	34
Pulse outputs.....	17	Terminal Program	32
R		TFTPD32	52
ReadSectorTable.....	56	<i>tShPImgLayoutDscrpt</i>	59
REL0 ... REL3	16	<i>tShPImgSectDscrp</i>	59
Relay outputs.....	16	U	
RTC setting.....	48	<i>UdpRemoteCtrl</i>	24
Run/Stop switch.....	25	USB-RS232 Adapter Cable	10
Run/Stop Switch		User Accounts	
Deletion of PLC Program	27	Adding and deleting	47
Run-LED	25	Changing Passwords	47
S		Predefined	43
Selecting the firmware version	42	W	
		WEB-Frontend	36
		WinSCP	45
		WriteSectorTable	56

Document: System Manual PLCmodule-C32
Document number: L-1072e_1, 1st Edition September 2009

How would you improve this manual?

Did you detect any mistakes in this manual?

Page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Please return your suggestions to: SYS TEC electronic GmbH
August-Bebel-Str. 29
D - 07973 Greiz
GERMANY
Fax : +49 (0) 36 61 / 6279-99
Email: info@systec-electronic.com