

Manual

CP 1413plus

Order No.: VIPA SSN HB83E
Rev. 00/07

The information contained in this manual is subject to change without notice. VIPA GmbH does not accept any liabilities, express or implied, with this document. Any hardware and software described in this manual is supplied on the basis of a general licence. Software usage and disclosure to third parties is only permitted where this is subject to the contractual conditions. Anyone who prepares copies of the software or the manual on magnetic tape, diskette or any other medium for purposes other than for personal use will be prosecuted, unless prior written permission is obtained from VIPA GmbH.

The software described in this manual is protected by international Copyright laws. Any information that might have become available after this manual was printed are supplied in a file on the accompanying disk. If this information is available insert the VIPA driver diskette #1 into drive A and enter the following command for MS-DOS:

A>TYPE README.TXT

If you should be using Windows[®], you can use "NOTEPAD" to view the file.

© Copyright 2000 VIPA, Gesellschaft für Visualisierung und Prozeßautomatisierung mbH,
Ohmstraße 4, D-91074 Herzogenaurach

Tel.: +49 (91 32) 744-0

Fax.: +49 (91 32) 744-144

E-Mail: info@vipa.de

<http://www.vipa.de>

Hotline: +49 (91 32) 744-114

All rights reserved

VIPA[®] is a registered trade mark of VIPA company for visualisation and process automation Ltd..

Windows[®] is a registered trade mark of Microsoft Corp.

Windows[®] NT is a registered trade mark of Microsoft Corp.

Windows[®] 95 is a registered trade mark of Microsoft Corp.

MS-DOS[®] is a registered trade mark of Microsoft Corp.

SIMATIC[®] is a registered trade mark of Siemens AG.

NetWare[®] is a registered trade mark of Novell Corp.

IBM-LAN-Server[®] and OS/2[®] are registered trade marks of IBM

3Com[®] is a registered trade mark of 3Com Corp.

EtherLink[®] III is a registered trade mark of 3Com Corp.

Any other trade marks referred to in the text are the trade marks of the respective owner and we acknowledge their registration.

About this manual

This manual describes the installation of the hardware and the software for a Industrial Ethernet (H1) network.

The manual also contains a comprehensive description of the configuration and the programming of a Industrial Ethernet network using VIPA H1 components.

Overview

Chapter 1: Introduction

This chapter contains a short description of two adapters that you may use for the respective type of LAN. The chapter is concluded by a description of the operation and properties of these two adapters.

Chapter 2: Planning a Network

Here you will find a summary of available cabling methods for networks together with the respective hardware and guidelines as well as a list of possible combinations. The chapter also presents an introduction to network planning and the preparation and installation of the respective hardware.

Chapter 3: VIPA Network module CP1413plus

This chapter contains a technical description of the VIPA ethernet adapter CP1413plus for personal computers. It also contains instructions for the configuration and installation of the VIPA ethernet adapter for personal computers with ISA- and PCI-bus interfaces and for the required software.

Chapter 4: Programming

This chapter contains a complete reference for the VIPA H1 interfacing software. This includes specifications of the H1 layer 4 and the PLC layer 7 programming interface.

Chapter 5: Files constants and structures

This chapter contains a description of the parameter file. It also contains up-to-date listings for the header files H1Def.h, S5Access.h, DrvFCall.h and WMKTypes.h.

Chapter 6: Examples

These examples demonstrate how you may receive and transmit with or without a parameter file.

Chapter 7: H1-driver V3.xx for Windows NT

This chapter contains a complete description of the application of the CP1413plus PC-adapter for Windows NT.

Contents

1 INTRODUCTION	1-1
1.1 Safety and handling precautions for the user	1-1
1.1.1 Handling electrostatically sensitive modules	1-1
1.1.2 Shipping electrostatically sensitive modules.....	1-2
1.1.3 Tests and modifications to electrostatically sensitive modules.....	1-2
1.2 General.....	1-3
1.3 Operation	1-4
1.4 Construction.....	1-4
1.5 Communication functions	1-5
1.6 Access functions.....	1-5
1.7 Special features.....	1-5
2 NETWORK PLANNING.....	2-1
2.1 Ethernet network-terminology	2-1
2.2 Thin-ethernet-cable networks	2-3
2.2.1 Thin-ethernet-cable network-layout	2-5
2.2.2 Regulations and specifications	2-6
2.2.3 Technical data thin-ethernet.....	2-7
2.3 Thick-ethernet-cable networks	2-8
2.3.1 Thick-ethernet-cable network-layout.....	2-10
2.3.2 Regulations and specifications	2-11
2.3.3 Technical data thick-ethernet	2-13
2.4 Combining thin/thick ethernet cable in networks	2-14
2.4.1 Combination of thin/thick ethernet cable hardware	2-14
2.4.2 Thin/thick ethernet cable combination network-layout.....	2-15
2.5 Twisted Pair	2-16
2.5.1 Twisted pair-cable network-layout	2-16
2.6 Planning a network-layout.....	2-18
2.7 Standards and specifications	2-19
3 VIPA CP1413PLUS NETWORK ADAPTER	3-1
3.1 Z'nyx PCI-bus adapter.....	3-2
3.1.1 Properties	3-2
3.1.2 Shipment	3-2
3.1.3 Hardware installation	3-4
3.1.4 Software installation	3-7
3.2 3COM ISA-bus adapter	3-30
3.2.1 Properties	3-30
3.2.2 Shipment	3-30
3.2.3 Hardware installation	3-32
3.2.4 Software installation	3-39
3.3 Entry into protocol file [H1PROT_NIF]	3-62

4 PROGRAMMING	4-1
4.1 General information on programming	4-1
4.1.1 Summary, operation - procedure	4-3
4.1.2 Description of the H1-parameters	4-6
4.1.3 Error messages from .Fehler	4-8
4.1.4 Determining the ethernet address.....	4-10
4.2 H1 Layer 4 program interface	4-11
4.2.1 General information on the H1 program interface.....	4-11
4.2.2 General H1 Layer 4 functions	4-13
4.2.3 Specific H1 Layer 4 functions	4-29
4.3 PLC Layer 7 program interface	4-41
4.3.1 General information on the PLC program interface	4-41
4.3.2 General Layer 7 functions.....	4-42
4.3.3 File functions Layer 7	4-47
4.3.4 Specific Layer 7 functions	4-61
5 FILES, CONSTANTS AND STRUCTURES.....	5-1
5.1 Parameter file description	5-1
5.2 Constants and structures	5-4
5.2.1 H1 protocol driver definitions.....	5-4
5.2.2 Layer 7 interface definition.....	5-7
5.2.3 Definition of calling codes for drivers	5-10
5.2.4 Definition of call types	5-12
6 EXAMPLES	6-1
6.1 Simple H1 transmission example.....	6-1
6.2 PLC reception example using a parameter file.....	6-3
6.2.1 Source listing	6-3
6.2.2 Source diagrams.....	6-5
6.2.3 The respective PLC program	6-8
6.2.4 The respective CP143plus parameter settings	6-9
6.2.5 The respective parameter file Net.net.....	6-10
7 H1 DRIVER V3.XX FOR WINDOWS NT	7-1
7.1 General	7-1
7.2 Uninstalling older Windows NT drivers.....	7-1
7.3 Installation of the VIPA CP1413plus network adapter.....	7-1
7.4 Installation of the H1 driver V3.xx for Windows NT 3.51.....	7-2
7.4.1 Installation of SSN-BG88 (Z'nyx adapter) for PCI bus	7-2
7.4.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus	7-3
7.5 Installation of the H1 driver V3.xx for Windows NT 4.0.....	7-4
7.5.1 Installation of SSN-BG88 (Z'NYX adapter) for PCI-Bus.....	7-4
7.5.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus	7-5
7.5.3 Extensions to the configuration options	7-6

7.6 Programming	7-7
7.6.1 General notes on programming	7-7
7.6.2 H1 Layer 4 programming interface	7-21
7.6.3 PLC Layer 7 software interface	7-55
7.6.4 PLC Net file functions	7-85
7.6.5 Files, constants and structures	7-97
Appendix	A-1
A Technical data	A-1
B Abbreviations	B-1
C List of figures	C-1
D Index	D-1

1 Introduction

1.1 Safety and handling precautions for the user	1-1
1.1.1 Handling electrostatically sensitive modules	1-1
1.1.2 Shipping electrostatically sensitive modules	1-2
1.1.3 Tests and modifications to electrostatically sensitive modules	1-2
1.2 General	1-3
1.3 Operation	1-4
1.4 Construction	1-4
1.5 Communication functions	1-5
1.6 Access functions	1-5
1.7 Special features	1-5

1 Introduction

1.1 Safety and handling precautions for the user

1.1.1 Handling electrostatically sensitive modules

VIPA-modules contain MOS LSI components. These components are highly susceptible to voltage transients which may occur when an electrostatic discharge takes place.

The following sign is affixed to these modules:



This symbol is attached to the module, the frame or the packing box and it indicates the sensitive nature of the module.

Electrostatically sensitive modules may be destroyed by energies or voltages which are far lower than human perception. It is possible that an electrostatic discharge occurs if a person who has built up an electrostatic charge handles electrostatically sensitive modules. The resulting voltages may damage the components on the module and thus damage or destroy the module. These faults are often not recognized immediately. Faults may only occur after prolonged operation.

Modules that were damaged by an electrostatic discharge may exhibit erratic or faulty operating characteristics when the ambient temperature changes, with mechanical shock or when electrical changes occur.

The only sure protection against the destruction of electrostatically sensitive modules is provided by consistent use of protective measures and by the responsible application of the respective rules and regulations.

1.1.2 Shipping electrostatically sensitive modules

You should always use the original packing when a module must be shipped. In addition you may also pack the module in a conductive enclosure. Conductive enclosures consist of antistatic foils or boxes made of metallized plastic.



Some modules carry a battery. When shipping such a module, please ensure that the conductive enclosure is not in contact with or shorting the battery terminals.

1.1.3 Tests and modifications to electrostatically sensitive modules

When tests are performed on electrostatically sensitive modules please note the following:

- discharge “floating” instruments before use.
- ground any instrument that you want to use.

Use a properly grounded soldering iron when you are installing modifications on electrostatically sensitive modules.



Ensure proper grounding for the worker and his tools before any attempt is made to operate on electrostatically sensitive modules.

1.2 General

The VIPA CP1413plus module is a network interface for your PC which is compatible with a number of protocols. You may use the module to provide a network interface for Industrial Ethernet (H1) links between your offices and the shop floor. The module supports the most common operating systems, e.g. MS-DOS[®], WINDOWS[®] 3.11, WINDOWS[®] 95, WINDOWS[®] NT and OS/2[®].

Depending on the type of PC bus you may choose between two types of VIPA network modules:

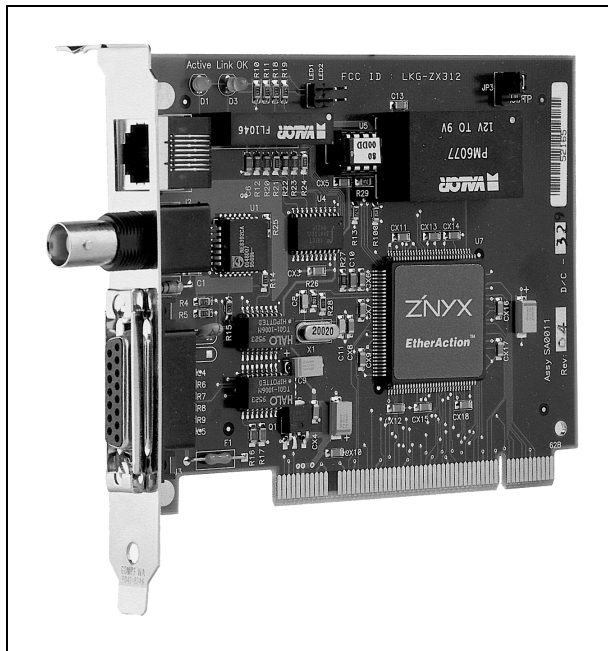


Fig. 1-1: CP1413plus for the PCI-Bus

PCI-Bus board

Z'nyx -board:(Order No.:VIPA SSN-BG88)

Locator

Board description: chapter 3.1

Hardware installation: chapter 3.1.3

Software installation: chapter 3.1.4

This board is plug and play compatible, i.e. it is integrated into your system without any user intervention.

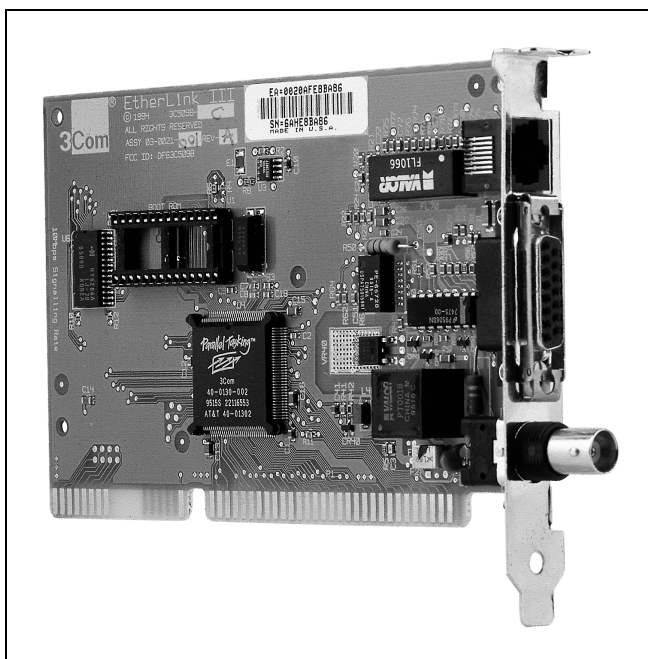


Fig. 1-2: CP1413plus for the ISA-Bus

ISA-Bus-board

3COM-board:(Order No.:VIPA SSN-BG85C)

Locator

Board description: Chapter 3.2

Hardware installation: Chapter 3.2.3

Software installation: Chapter 3.2.4

This board requires configuration by means of software. The respective software is located on the diskette 1/1 (see chapter 3.2.3.2).

1.3 Operation

You must install the CP1413plus board directly into your PC. The board has 3 interfaces:

1. An interface for the Industrial Ethernet (H1) LAN cable via a transceiver
2. An interface for Industrial Ethernet (H1) using thin-ethernet cable
3. An interface for Industrial Ethernet (H1) via an RJ-45-socket for twisted pair cables

The required operating mode is selected automatically.

The CP1413plus communication processor communicates via the Industrial Ethernet and other IEEE 802.3 compatible cellular networks. **NDIS** drivers control the exchange of data between the CP1413plus and the PC. This provides a simple path for expanding the system when other networks and protocols are required. It is also possible to operate multiple networking protocols in parallel..

You are provided with a simple menu-controlled configuration program to configure connection parameters and other system related settings, like transport connections and links to the PLC. The entered parameters are saved into a configuration file.

The module can control up to 256 connections. It is also possible to operate the module with priority 0/1 (datagram, multicast and broadcast), 2/3 and 4.

1.4 Construction

The modules are provided with:

- a standard AT or PCI interface
- an AUI interface (IEEE 802.3) for connecting to a LAN via a transceiver
- a BNC connector for thin ethernet (Cheapernet)
- an RJ-45 connector for interfacing to twisted pair cable
- a power supply for the transceiver

Interface connectors:

- a 15-pole D-type socket for connecting to a cellular network
- a BNC connector for interfacing to thin ethernet
- an RJ-45 connector for interfacing to twisted pair cable

1.5 Communication functions

In addition to the general test applications which are intended for common functional tests the module is also provided with a programming interface which is independent of the operating system:

- **MS-DOS**: C-interface as a library (C-source code)
- **Windows® 3.11, Windows® NT and Windows® 95**: language-independent DLL (Dynamic Link Library)
- **OS/2®**: language-independent DLL (Dynamic Link Library)

1.6 Access functions

The supplied software contains a programming interface for application programs which allows for **unrestricted access** to the PLC (DB:DW, module transfer, memory access, conditions and many more). Like the communication interface this application-level support is functionally independent of the operating system:

- **MS-DOS**: C-interface as a library (C-source code)
- **Windows® 3.11, Windows® NT and Windows® 95**: language-independent DLL (Dynamic Link Library)
- **OS/2®**: language-independent DLL (Dynamic Link Library)

1.7 Special features

- **Special features**The CP1413plus board may be installed as a standard interface into any compatible PC which has provisions for an ISA- or a PCI slot.
- In addition to H1, the NDIS interface of the board may be used for other IEEE 802.3 compatible protocols.
- The CP1413plus communication processor supplies power to the transceiver.
- The generation of drivers and application programs is easily accomplished by means of the operating-system independent libraries supplied with the module.
- Simple menu-driven configuration program.
- The module is delivered complete with all drivers and libraries.

2 Network planning

2.1 Ethernet network-terminology	2-1
2.2 Thin-ethernet-cable networks	2-3
2.2.1 Thin-ethernet-cable network-layout	2-5
2.2.2 Regulations and specifications	2-6
2.2.3 Technical data thin-ethernet	2-7
2.3 Thick-ethernet-cable networks	2-8
2.3.1 Thick-ethernet-cable network-layout	2-10
2.3.2 Regulations and specifications	2-11
2.3.3 Technical data thick-ethernet	2-13
2.4 Combining thin/thick ethernet cable in networks	2-14
2.4.1 Combination of thin/thick ethernet cable hardware	2-14
2.4.2 Thin/thick ethernet cable combination network-layout	2-15
2.5 Twisted Pair	2-16
2.5.1 Twisted pair-cable network-layout	2-16
2.6 Planning a network-layout	2-18
2.7 Standards and specifications	2-19

2 Network planning

2.1 Ethernet network-terminology

An ethernet network provides a communication link between different stations on the network. The stations on the network may consist of personal computers, industrial computers, automation equipment etc. Stations are connected to the network and located a certain minimum distance from each other. The stations on the network and the LAN cable form a network segment. (See Fig. 2-1).

The cable of the network segment usually consists of a number of network cables of varying length that are linked by means of connectors.

The length of the network segment and the number of stations that may be connected to it is limited. It is possible to extend a network segment by linking multiple segments via repeaters.

A repeater increases the physical topology of a network. The repeater requires two transceivers to link two different segments to each other. In this case the repeater is transparent to the network and combines the two segments into a single logical channel.

In this situation the repeaters task is to regenerate and to amplify the signals in both directions. It must also be able to recognize, process and to communicate inter-segment collisions. Repeaters do not possess a unique address and are therefore not accessible as they are invisible to the stations connected to the network.

The overall network cable consists of the sum of all segment cables.

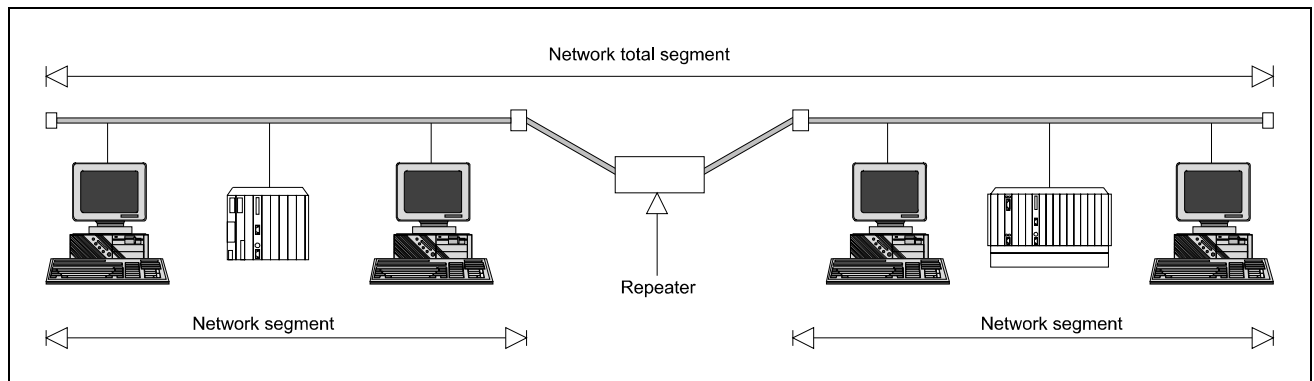


Fig. 2-1: Parts of an ethernet network

Three types of ethernet cable exist:

- Thin-ethernet-cable (also called thin ethernet cable or Cheapernet-cable).
- Thick-ethernet-cable (also called thick ethernet cable or standard ethernet-cable or yellow-cable).
- Twisted pair-cable (telephone cable)

Thin ethernet-cable is far more economical than thick ethernet-cable. This applies to the cost of the cable, the cost of installation and any additional hardware. It does, however, suffer from a higher level of susceptibility for EMC interference. Connections to the thin ethernet-cable are provided by BNC T-pieces.

Thick ethernet-cable consists of a thick yellow cable which is much like a pipe. The cable has markings at a distance of 2,5 m which indicate the positions where a connection to the cable may be attached.

These two types of network cable can be employed to produce in three different versions of ethernet network:

1. a network using only thin ethernet cable
2. a network consisting only of thick ethernet cable and
3. a combination of these two

Under certain conditions you may also use twisted telephone twin-cable instead of the coaxial cable (twisted pair).

Twisted pair cable consists of four insulated copper wires of 1 mm diameter that have been twisted together in pairs.

In contrast to the coaxial ethernet networks which provide a type of bus-topology, the twisted pair network is a point-to-point network. The resulting network has a star-topology. Every station is coupled individually with a central hub to form the ethernet network.

2.2 Thin-ethernet-cable networks

Thin-ethernet-cable network hardware

This chapter contains a description and an illustration (Fig. 2-2) of the hardware required for a thin-ethernet-network.

Network adapter

Network adapters are installed in every station on the network and linked by means of a network cable to provide the communication medium between these stations.

BNC-connectors

BNC-plugs and -sockets provide the electrical link between the hardware on the network. The network cable is connected to the BNC-socket located on the network adapter. BNC-plugs connected to both ends of the network cable link the different segments of the cable via a T-piece or a barrel connector. The T-piece is connected to the BNC-socket on the network adapter.

Thin-ethernet-cable

Thin-ethernet-cable consists of RG-58A/U 50 Ohm coaxial cable with a diameter of 0,2 inch (5,08 mm).

BNC-connectors

BNC-connectors are used to interconnect two thin-ethernet network cables. Where two portions of a thin-ethernet network must be linked these connectors have a distinct advantage over T-pieces. You may obtain these BNC-connectors from VIPA GmbH.

BNC-T-pieces

The two sockets located at opposite ends of the T-piece provide the connection to the BNC-connectors on the thin-ethernet cable. The third connector on the T-piece is connected to the BNC-connector on the network adapter and provides the physical link to the network. You may obtain BNC-T-pieces from VIPA.

BNC-terminator

Every network must end in a 50 Ω BNC terminator which acts as a matched load and which reduces electrical interference on the network. The terminator is connected to one of the two sockets of the T-piece when no other cable is connected. Some BNC terminators have a ground connection which may be used to ground the screen of the network cable. You may obtain BNC terminators with and without ground connection from VIPA GmbH.



One side of a network segment (it is immaterial which side) must be fitted with a BNC-terminator which has a ground connection!

The other side of the segment may not have a terminator!

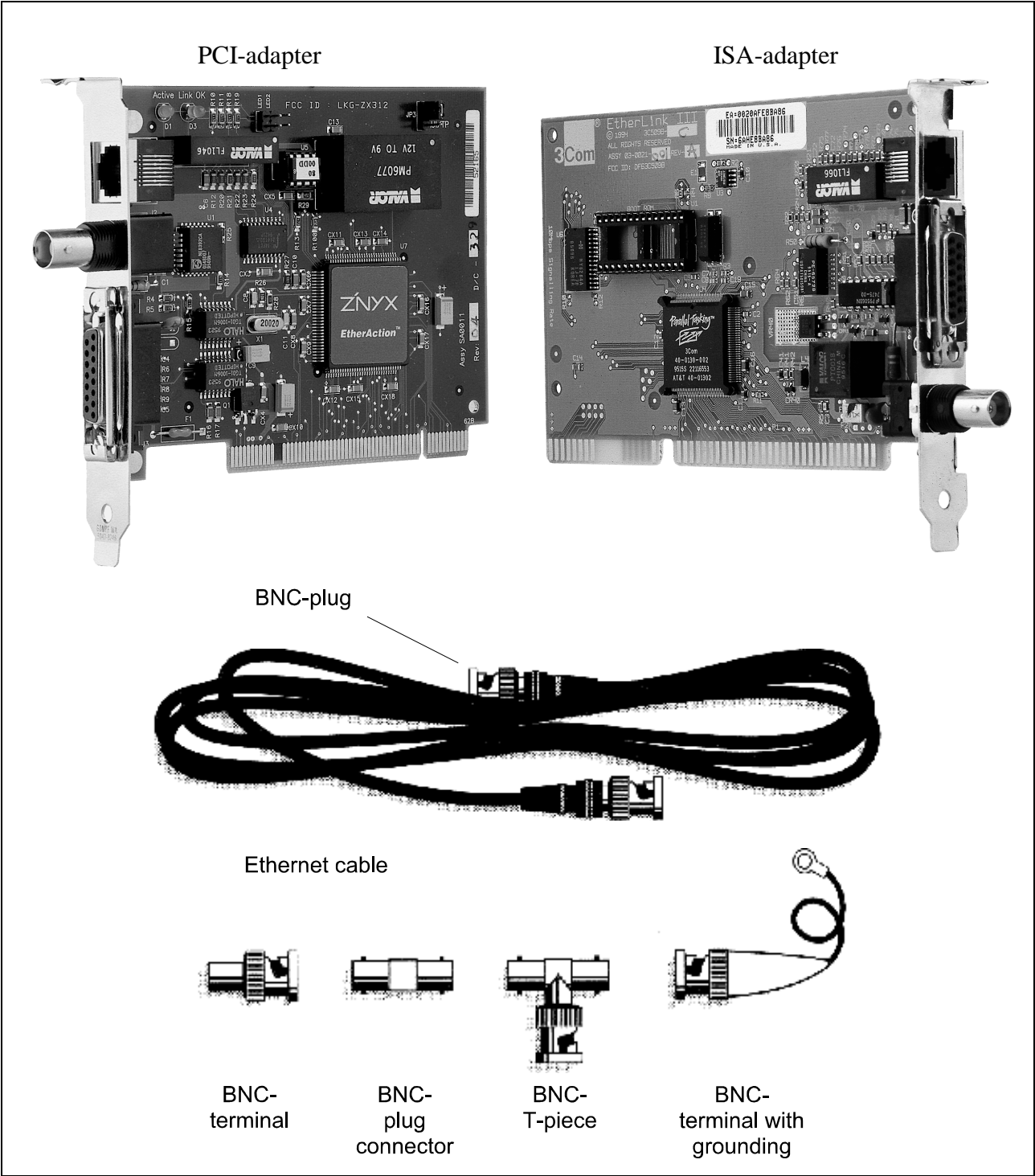


Fig. 2-2: Thin ethernet network hardware

2.2.1 Thin-ethernet-cable network-layout

Here follows a summary of the restrictions and the rules that apply to a thin-ethernet-cable network. Fig. 2-3 is an illustration of this summary.

Restrictions

- Maximum number of network segments: 5 (3 coaxial segments with network stations and 2 interconnecting segments without network stations)
- Maximum length of a network segment: 185 m
- Maximum total length of the network cable: 925 m
- Maximum number of network stations on a segment: 30 (every repeater is counted as a station)
- Minimum distance between two BNC-T-pieces: 0,5 m

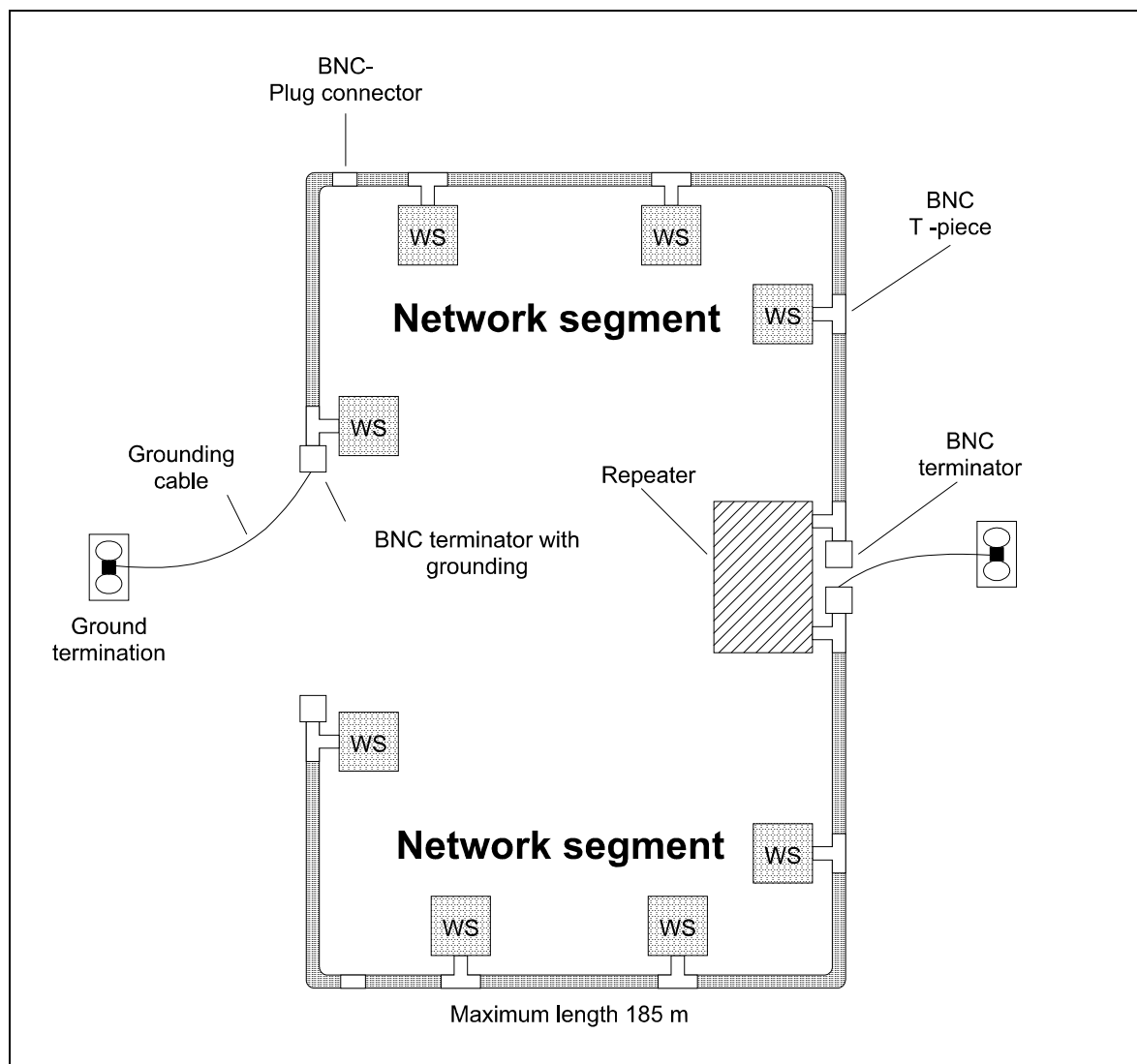


Fig. 2-3: Example for a thin-ethernet cable network

2.2.2 Regulations and specifications

Certain basic regulations must be observed when planning or installing a thin ethernet network:

- The cabling of a network consists of a number of segments. Each segment must be terminated with its characteristic impedance (terminator) of 50 Ω .
- According to the ISO 8802.3 specification the maximum length of an individual segment is 185 m.
- Wherever possible, the network should consist of a homogeneous segments of coaxial cable as every connector introduces losses. When it becomes necessary to extend existing segments of coaxial cable you should always use preconfigured cables. Please note that the maximum length of a segment may not exceed 185 m.
- The screen of the coaxial cable may only be grounded at a single point on the entire cable. This point must possess a defined low impedance. All other connectors and terminators must be insulated from ground.
- Any two stations on the network may only have a single connection between them. Where multiple physical paths exist the resulting interference will be interpreted as a signal and cause collisions which may abort the data transfer.
- Single drop-cables are not permitted.

Guidelines for the routing and installation of network components

- As a rule, a minimum distance of 1m is prescribed with respect to electrical equipment, cabling and other components that could produce electromagnetic and electrostatic fields. This includes parallel runs of power or high-tension cables, circuit breakers etc.
- The ISO-specifications allow for a maximum field strength of 2 V/m at frequencies between 10 kHz and 30 MHz and 5 V/m at frequencies between 30 MHz and 1 GHz.
- Once a coaxial cable segment and all its components have been installed it is essential that the worst-case reflections on the cable are determined. These should never exceed 7% of the injected amplitude of the original signal. The reflection coefficient can be checked by means of a reflectometer.
- The overall loop resistance of a coaxial cable segment together with all its plugs and connectors may not exceed a maximum of 4Ω .
- The minimum radius of any bend in the cable may not be less than 5 cm. Where multiple bends are anticipated the radius must not be less than 8 cm.

2.2.3 Technical data thin-ethernet

Coaxial cable

Impedance	$50 \Omega \pm 2 \Omega$
Attenuation	$\leq 18 \text{ dB/km}$ for a 10 MHz sine wave $\leq 12 \text{ dB/km}$ for a 5 MHz sine wave
Propagation	$\geq 0,77 \text{ c}$ (velocity of light)
Minimum radius for bends, single bend	5 cm
Minimum radius for bends, multiple bends	8 cm
Diameter of center conductor solid copper	$0,89 \text{ mm} \pm 0,05 \text{ mm}$
Outside diameter for polyvinylchloride (PVC)	4,9 mm
Outside diameter for fluoropolymer	4,8 mm

2.3 Thick-ethernet-cable networks

Thick-ethernet-cable network hardware

This chapter contains a description and an illustration (Fig. 2-4) of the hardware required for a thick-ethernet-network.

Network adapter

Network adapters are installed in every station on the network and linked by means of a network cable to provide the communication medium between these stations.

Transceiver

The stations on a thick-ethernet cable communicate via an external transceiver which is attached to the network cable. External transceivers are required when a thick-ethernet cable is used as communication medium.

Transceiver-cable

The transceiver-cable (drop-cable) connects the station (network adapter) to the external transceiver.

AUI-connector

The end of a transceiver cable is terminated in an AUI (DIX) plug or an AUI-socket. The AUI-plug is connected to the network adapter located in a station. The AUI-socket is connected to the external transceiver.

Thick-ethernet-cable

The thick-ethernet-cable consists of a 50 Ohm coaxial cable with a diameter of 0,4 inch (10,16 mm). Different lengths of this cable are available complete with standard 0,4 inch N-type plugs at each end. The cable may be obtained from any reputable cabling supplier.

N-type plug

An N-type plug is attached to each end of the thick-ethernet cable.

N-type piece

N-type connectors are used to connect two thick-ethernet cables together.

N-type-terminator

An N-type 50 Ω termination resistance “terminates” the network and prevents electrical interference from the network. This is attached to the N-type socket of the transceiver that is not connected to another cable. Certain terminators are provided with a grounding strap to connect the screen of the network cable to ground.



One side of a network segment (it is immaterial which side) must be fitted with a terminator which has a ground connection!

The other side of the segment may not have a terminator!

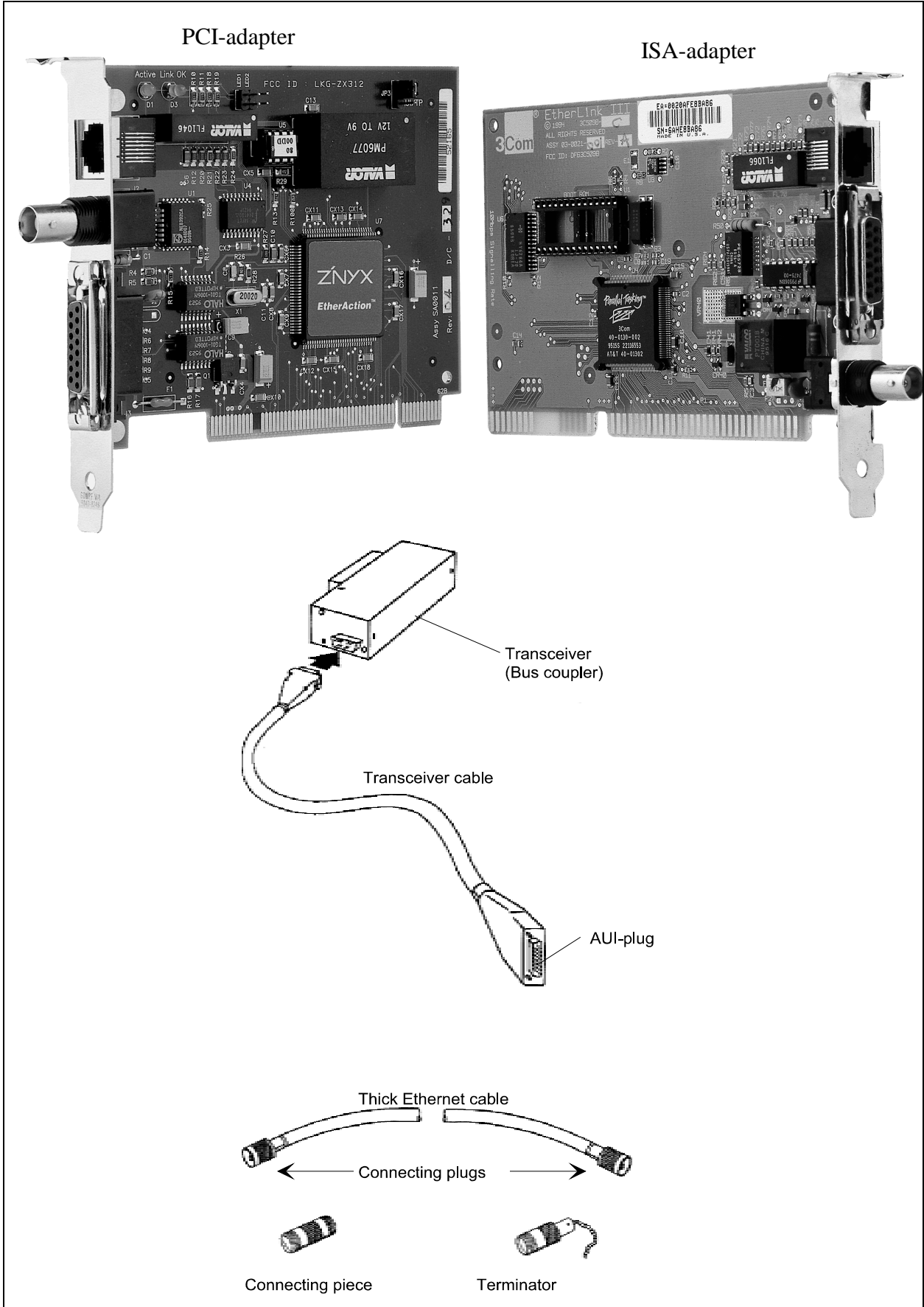


Fig. 2-4: thick-ethernet network hardware

2.3.1 Thick-ethernet-cable network-layout

Here follows a summary of the restrictions and the rules that apply to a thick-ethernet-cable network. Fig. 2-5 is an illustration of this summary.

Restrictions

- Maximum number of network segments: 5 (3 coaxial segments with network stations and 2 interconnecting segments without network stations)
- Maximum length of a network segment: 500 m
- Maximum total length of the network cable: 2.500 m
- Maximum number of network stations on a segment: 100 (every repeater is counted as a station)
- Minimum distance between two transceivers: 2,5 m
- Maximum length for a transceiver cable: 50 m

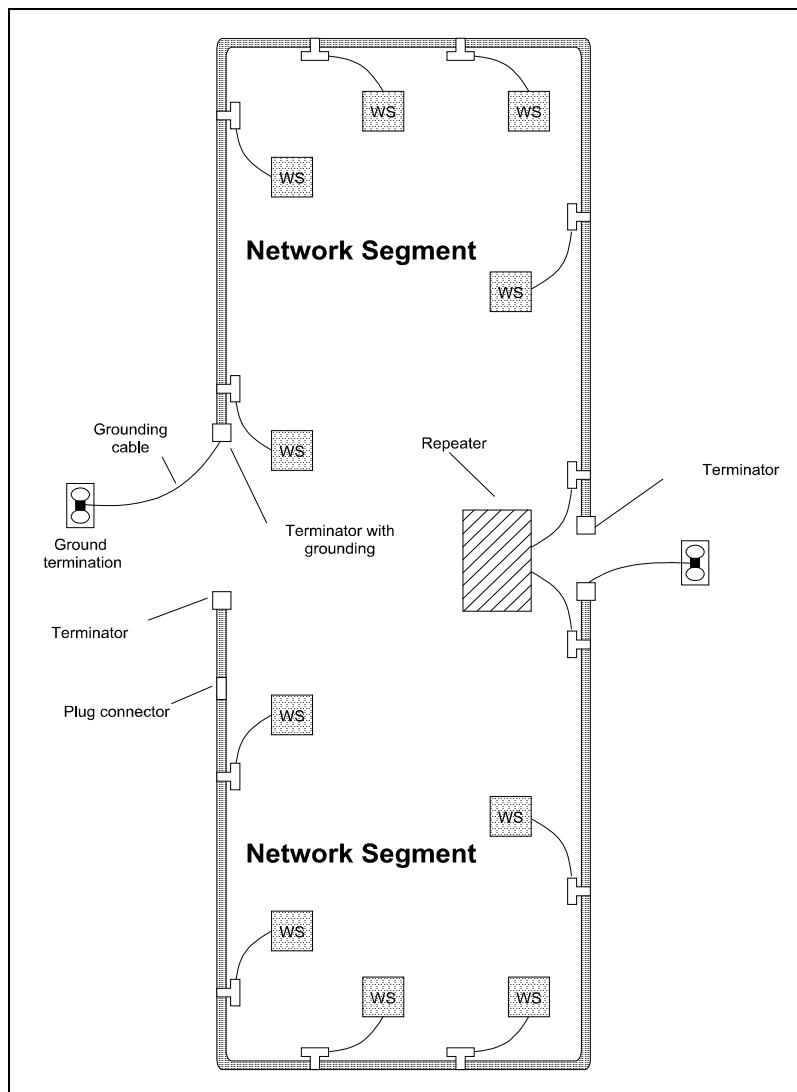


Fig. 2-5: example for a thick-ethernet network

2.3.2 Regulations and specifications

Certain basic regulations must be observed when planning or installing a thick ethernet network:

- The cabling of a network consists of a number of segments. Both ends of each segment must be terminated with its characteristic impedance (terminator) of 50 Ω .
- According to the ISO 8802.3 specification the maximum length of an individual segment is 500 m.
- Any segment may consist of a number of different parts. In this case you must ensure that the length of each part is an odd multiple of 23,4 m to eliminate reflections at the joints and the ends of any portion of the cable.
- The length of any part may thus be calculated as follows:
$$\text{TSL} = (2n + 1) * 23,4 \text{ m}; \quad N = 0, 1, 2, 3...$$
- Wherever possible, the network should consist of a homogeneous segments of coaxial cable as every connector introduces losses. When it becomes necessary to extend existing segments of coaxial cable you should always use standard portions of the cable. Please note that the maximum length of a segment may not exceed 500 m.
- The screen of the coaxial cable may only be grounded at a single point on the entire cable. This point must possess a defined low impedance. All other connectors and terminators must be insulated from ground.
- Any two stations on the network may only have a single connection between them. Where multiple physical paths exist the resulting interference will be interpreted as a signal and cause collisions which may abort the data transfer.
- There are no spur lines allowed.

Guidelines for the routing and installation of network components:

- As a rule, a minimum distance of 1m is prescribed with respect to electrical equipment, cabling and other components that could produce electromagnetic and electrostatic fields. This includes parallel runs of power or high-tension cables, circuit breakers etc.
- The ISO-specifications allow for a maximum field strength of 2 V/m at frequencies between 10 kHz and 30 MHz and 5 V/m at frequencies between 30 MHz and 1 GHz.
- Once a coaxial cable segment and all its components have been installed it is essential that the worst-case reflections on the cable are determined. These should never exceed 7% of the injected amplitude of the original signal. The reflection coefficient can be checked by means of a reflectometer.
- The overall loop resistance of a coaxial cable segment together with all its plugs and connectors may not exceed a maximum of 5Ω .
- The minimum radius of any bend in the cable may not be less than 21 cm. Where multiple bends are anticipated the radius must not be less than 40 cm.

2.3.3 Technical data thick-ethernet

Coaxial cable

Impedance	$50 \Omega \pm 2 \Omega$
Impedance linearity	$\pm 3 \Omega$ with a period > 2 m
Inductivity	$0,21 \mu\text{H/km}$
Capacity	$85 \text{ pF} \pm 5 \text{ pF}$
Attenuation	$\leq 18 \text{ dB/km}$ for a 10 MHz sine wave $\leq 12 \text{ dB/km}$ for a 5 MHz sine wave
Propagation	$\geq 0,77$ c (velocity of light)
Loop resistance	$\leq 10 \text{ m } \Omega/\text{m}$ at 20° C
Minimum radius for bends, single bend	21 cm
Minimum radius for bends, multiple bend	40 cm
Diameter of center conductor solid copper	$2,17 \text{ mm} \pm 0,013 \text{ mm}$
Thickness of the dielectric	1,99 mm
Thickness of the screen	$1,06 \text{ mm} \pm 0,089 \text{ mm}$
Screening effectiveness	$\geq 92 \%$
Thickness of the outer insulator	$1,25 \text{ mm} \pm 0,089 \text{ mm}$
Outside diameter for polyvinylchloride (PVC)	$10,287 \text{ mm} \pm 0,178 \text{ mm}$
Outside diameter for fluoropolymer	$9,525 \text{ mm} \pm 0,178 \text{ mm}$

Transceiver cable

Maximum operating voltage	30 V
Maximum current capacity	2,8 A
Capacity of a single pair of lines	55 pF/m
Impedance	$78 \Omega \pm 5 \Omega$
Screening effectiveness	78 %
Maximum attenuation	3 dB / 50 m

The yellow outer cover of the cable bears imprinted marks at a distance of $2,5 \text{ m} \pm 5 \text{ cm}$. These marks indicate the positions where a transceiver may be connected. The coaxial connectors at the end of every cable are N-type plugs.

2.4 Combining thin/thick ethernet cable in networks

It is possible to combine thin-ethernet and thick-ethernet cabling in a single network.. The lower price of the thin-ethernet cable may provide a more cost-effective solution than a system that is purely based on thick-ethernet cable.

When thick-ethernet cable is used to link repeaters the distance may be substantially larger than if the link was provided by a thin-ethernet cable.

You may use one of the two methods below to establish a network that combines thick and thin-ethernet cable:

- Connecting the thin-ethernet cable segment to a network based on thick-ethernet cable via a repeat. It is possible to combine a maximum of five unlike network segments (using four repeaters) in this manner.
- It is also possible to use thin and thick-ethernet cables within the same network segment. The structure of such a network is described below under the heading "Combining thin/thick ethernet cable hardware".

2.4.1 Combination of thin/thick ethernet cable hardware

Networks that use a combination of thin and thick ethernet cable employ the network hardware as the individual thin-ethernet and the thick-ethernet networks would if they are not combined. It is, however necessary to include adapters that provide the interconnection between the different types of cable. Fig. 2-6: shows two such adapters.

- N-type socket to BNC-socket
- N-type plug to BNC-socket

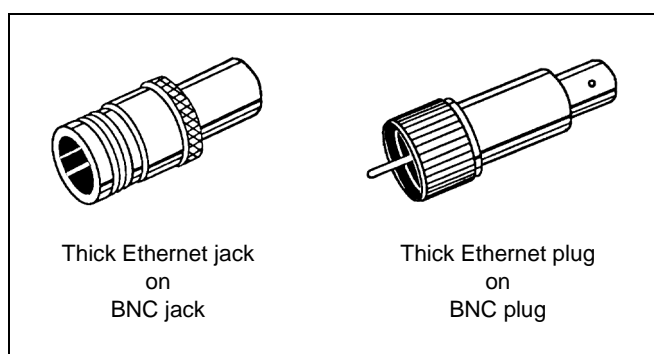


Fig. 2-6: Adapters for linking unlike segments

2.4.2 Thin/thick ethernet cable combination network-layout

This chapter describes a method of combining thin ethernet cable and thick ethernet cable in a single network segment. This method employs as much thin ethernet cable as possible. Fig 2-7 illustrates this technique of combining thin and thick ethernet cable.

The length of a network employing a combination of thin and thick ethernet cable may be between 185 m and 500 m. In this case the minimum length has been specified as 185 m as shorter segments can consist entirely of thin ethernet cable. The maximum length of 500m is determined by the physical limitations of the thick ethernet cable.

You may use the following equation to calculate the maximum length of the thin ethernet cable in a combination network:

$$\frac{500\text{m} - L}{3.28} = t$$

(L = length of network segment that must be implemented

t = maximum length of the thin ethernet cable)

e.g. you must implement a network segment of 457 m and reduce the cost of the network hardware to a minimum. You would use the equation above to determine the maximum length of the thin ethernet cable as follows.

$$\frac{500\text{m} - 457\text{m}}{3.28} = 13,1\text{m}$$

You may thus use up to 13,1 m of the cost-effective thin ethernet cable. For the remainder of 443,9 m of the network segment you must use thick ethernet cable.

Please note: you can interconnect up to 5 different network segments!

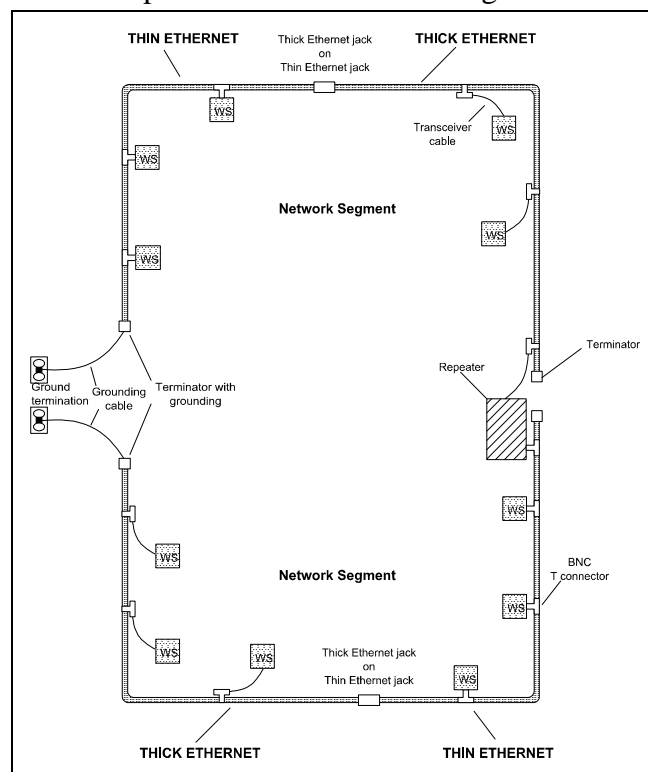


Fig. 2-7: Example of a network combining thin and thick ethernet cable

2.5 Twisted Pair

Twisted Pair network hardware

A twisted pair-network can only have a star-type configuration. This topology requires a hub that supports the star-type configuration.

The following paragraphs contain a description of the hardware required for implementing a twisted pair network.

Network adapter

The network adapter which is installed in every station on the network provides the interface that allows every station to communicate with every other station..

Hub (star-type configuration)

The Hub is the central element that provides the facilities required to implement an ethernet network using twisted pair cable. Every station is connected directly to the hub to form the ethernet network. A hub operates like a repeater. It may be interfaced with thin or thick ethernet cable..

Twisted pair-cable

The twisted pair cable consists of a four-core cable where the cores are twisted together in pairs. The conductors have a diameter between 0,4 and 0,6 mm.

2.5.1 Twisted pair-cable network-layout

Here follows a summary of the restrictions and the rules that apply to twisted pair networks

Restrictions

- Maximum number of hubs per segment: 2
- Maximum segment length: 100 m

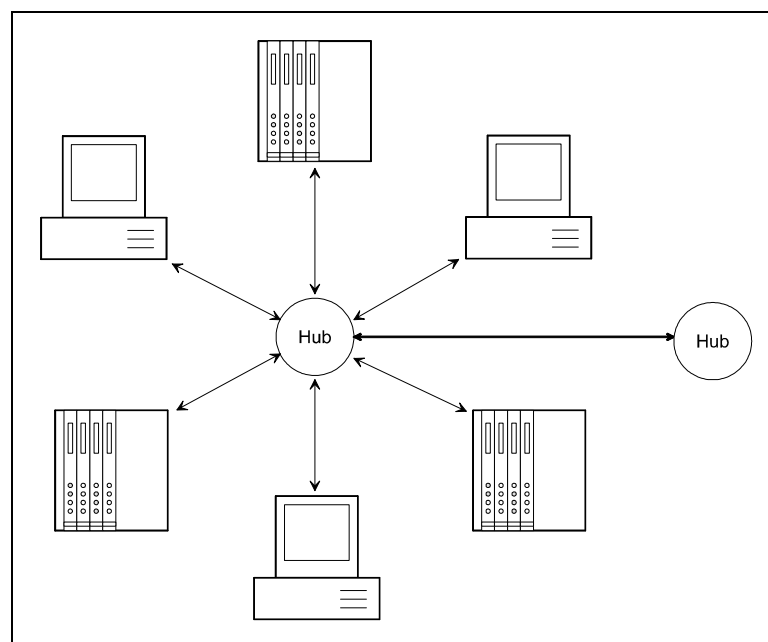


Fig. 2-8: Star topology of a twisted pair network

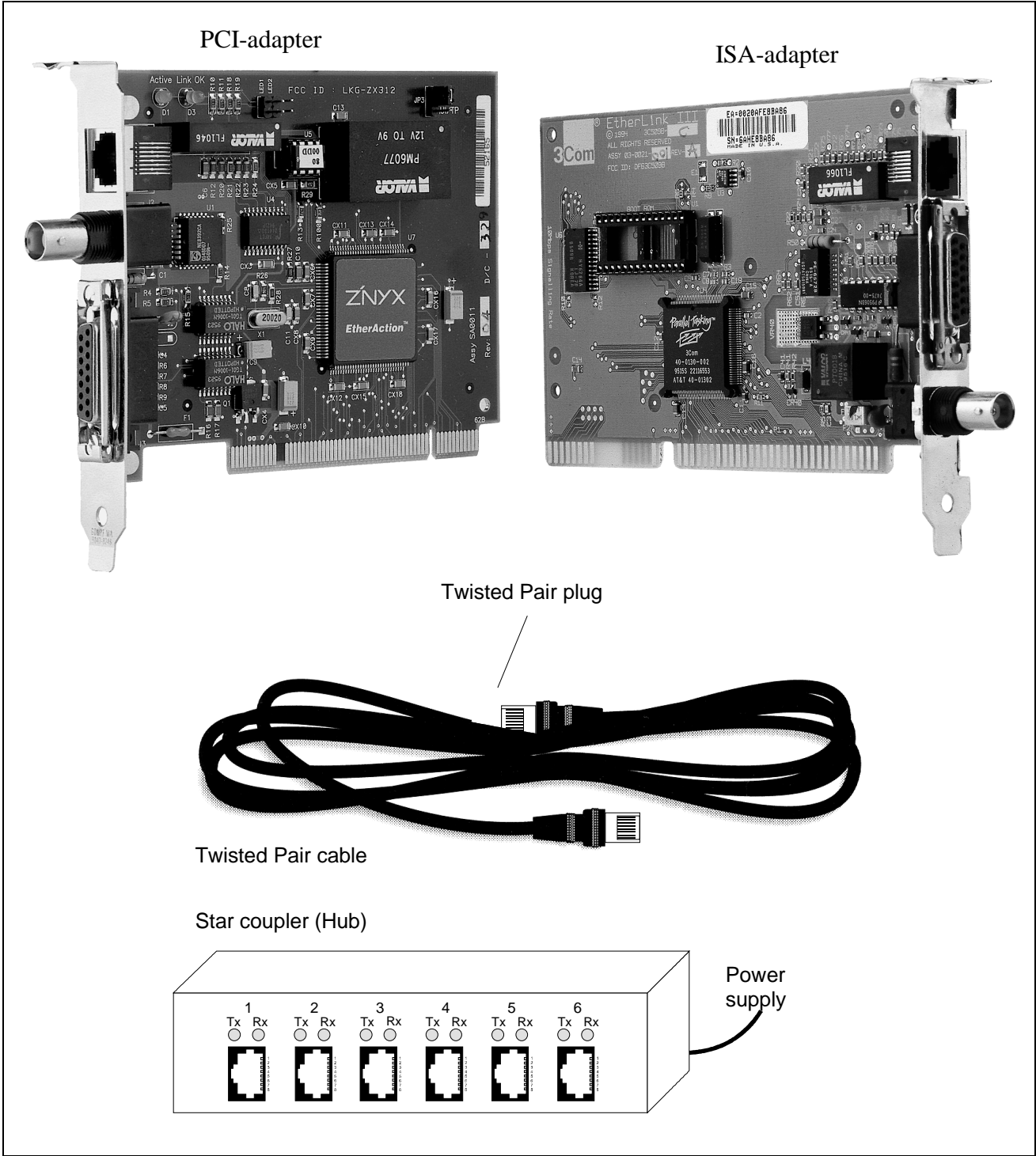


Fig. 2-9: Twisted pair network hardware

2.6 Planning a network-layout

At this point you are aware of the limitations of the different cabling systems, and you can provide answers to the following questions with respect to the network that you have to implement:

Determination of network requirements

- What is the extent of the coverage of the proposed network?
- What is the optimum number of network segments required to satisfy the physical (space, interference) conditions applicable to the proposed system?
- How many network stations (PLC, IPC, PC, transceiver, possible bridges) should be connected to the network?
- What is the distance between the different stations on the network?
- What is the expected increase in coverage and number of connections that the system must be able to handle?

Drawing a plan of the network

Prepare a drawing of the plan of the network! Identify every item of hardware (like cables, transceivers, amplifiers, termination resistors). Observe the rules and limitations that were outlined in the previous chapters.

Measure the distance between all components to ensure that they are within the prescribed parameters (min. distance, max. length...).

2.7 Standards and specifications

Standards and specificationsThe main characteristic of a LAN is that it provides a single physical communication path. The physical communication medium may consist of:

- one or more electrical connections (twisted cables)
- coaxial cable (triaxial cable)
- fiber-optic cable

The communications between individual stations is subject to certain standards and rules that must be met by every station on the network. These determine the form of the communication protocol, the access method to the LAN and other basic principles that are important for communicating on the network.

The VIPA H1-network was developed to satisfy the ISO standards and specifications.

The standards and specifications for networking systems were determined by international and national committees.

ANSI **American National Standards Institute**

This body is currently busy formulating agreements for high-speed LAN's (100 MB/s) based on fiber-optic technology in ANSI X3T9.5. (FDDI) Fiber Distributed Data Interface.

CCITT **Committee Consultative Internationale de Telephone et Telegraph.**

Amongst others, this advisory committee produces the specifications governing the interfacing of industrial communication networks (MAP) and of office networks (TOP) to Wide Area Networks (WAN).

ECMA **European Computer Manufacturers Association.**

generates the different standards for MAP and TOP.

EIA **Electrical Industries Association (USA)**

this committee determines standards like RS-232 (V.24) and RS-511.

IEC **International Electrotechnical Commission.**

This committee sets specific standards, e.g. those for the Field Bus.

ISO **International Organisation for Standardization.**

This association of national standards offices has determined the OSI-model (ISO/TC97/SC16). It outlines the framework for the standards that are applicable to data communications. The ISO standards are converted to the different national standards like UL and DIN.

IEEE **Institute of Electrical and Electronic Engineers (USA).**

The project group 802 determines LAN-standards for communication speeds from 1 to 20 MB/s. IEEE standards are often the basis ISO-standards, e.g. IEEE 802.3 = ISO 8802.3.

The serial Industrial Ethernet LAN is an open communication system that can be used for the exchange of data between all devices that comply with a set of common rules. These devices may also include units manufactured by other companies.

Data is communicated on the basis on the basis of the OSI ISO-reference model. This model provides the specifications for all the processes, rules and the strategy for common data communications.

3 VIPA CP1413plus network adapter

3.1 Z'nyx PCI-bus adapter	3-2
3.1.1 Properties	3-2
3.1.2 Shipment	3-2
3.1.3 Hardware installation	3-4
3.1.4 Software installation	3-7
3.2 3COM ISA-bus adapter	3-30
3.2.1 Properties	3-30
3.2.2 Shipment	3-30
3.2.3 Hardware installation	3-32
3.2.4 Software installation	3-39
3.3 Entry into protocol file [H1PROT_NIF]	3-62

3 VIPA CP1413plus network adapter

Note

Depending on the type of bus in your PC, VIPA can supply two different network adapters:

- for PCI-bus: Z'nix-adapter (Order-No.: VIPA SSN-BG88)
- for ISA-bus: 3Com-adapter (Order.-No.: VIPA SSN-BG85C)

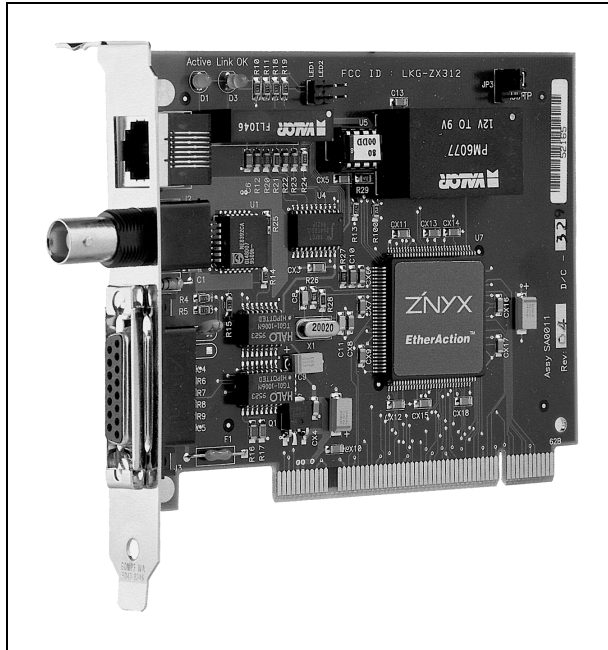


Fig. 3-1: CP1413plus for PCI-bus

PCI-bus adapter

Z'nix : (Best.-Nr.:VIPA SSN-BG88)

Guide

Description of adapter: Chapter 3.1

Hardware installation: Chapter 3.1.3

Software installation: Chapter 3.1.4

This adapter is plug- and play-compatible i.e. it is recognized by the system and installed without any user intervention.

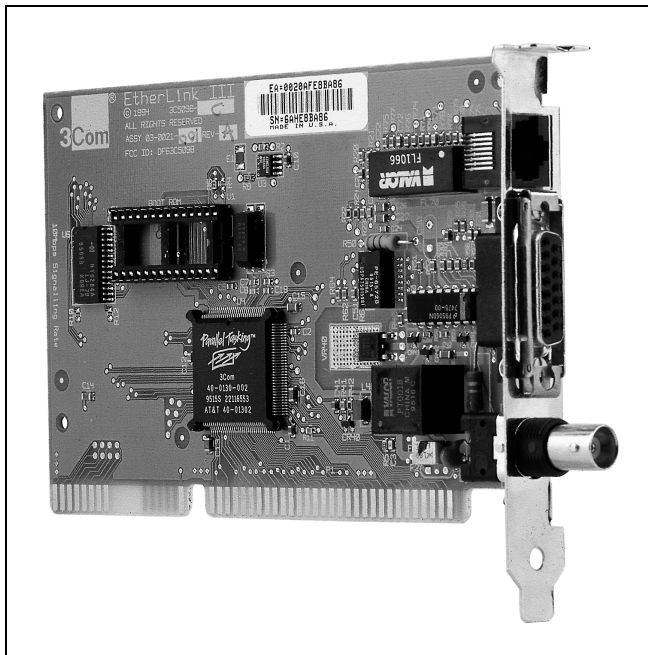


Fig. 3-2: CP1413plus for ISA-bus

ISA-bus adapter

3Com: (Order-No.:VIPA SSN-BG85C)

Guide

Description of adapter: Chapter 3.2

Hardware installation: Chapter 3.2.3

Software installation: Chapter 3.2.4

This adapter must be configured by means of software. The required software is supplied on disk 1/1 (see chapter 3.2.3.2).

Please verify what type of adapter you are using!

3.1 Z'nyx PCI-bus adapter

3.1.1 Properties

- Two LED's display adapter and network activity
- Support for plug- and play technology, i.e. fully automatic recognition of the adapter, no user intervention required.
- Auto Select Media Type (automatic recognition of the connector that is connected to the network).
- Adapter has a BNC socket (10 Base 2), an AUI (10 Base 5) socket and a RJ-45 Phone Jack (10 Base T)
- 32 Bit PCI-bus
- Network management software, installed with the drivers.

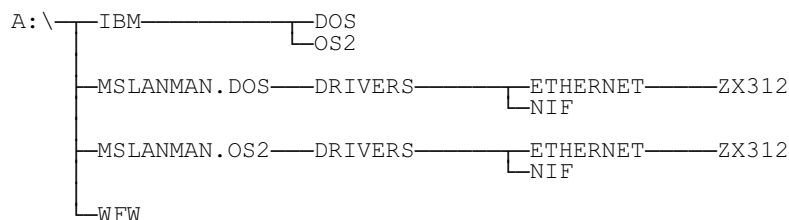
3.1.2 Shipment

Hardware

- 32-Bit-PCI bus network adapter
- BNC T-piece

Software

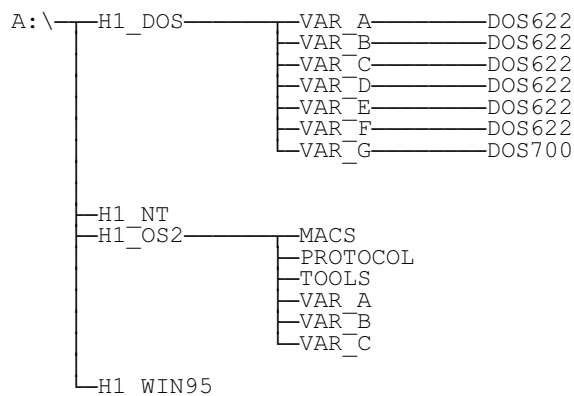
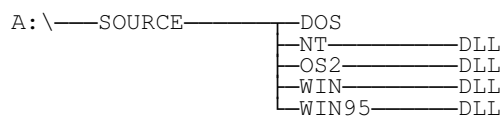
Disk 1: SW82Z disk 1/1, Z'nyx ethernet driver



Disk 2: SW83Z disk 1/2, adapter 1413plus H1-bus

- Driver software H1 for the operating systems :

- MS-DOS® 5.0 and 6.22
- Windows® 3.1 and WfW® 3.11
- Windows® NT 3.5, 3.51
- Windows® 95
- OS/2® 2.1 and OS/2® Warp

**Disk 3: SW83Z disk 2/2, adapter 1413plus H1-bus**

As of version 2.6 the files required for the different operating systems are compressed into a separate archive on the disk. Common files that are used by all operating systems have not been archived.

To extract the files required for your specific operating system you must copy the respective file to your hard disk and change to that directory. Now you must enter the command:

A:\PKUNZIPF <file name>

“file name” represents the file that you wish to extract.

Example: A:\PKUNZIPF DOS.ZIP

The extraction program may be executed in MS-DOS from version 3.0. You can also execute the extraction program in the DOS shell of Windows 95, Windows NT and OS2.

3.1.3 Hardware installation

3.1.3.1 Installing the adapter

The adapter must be installed in the PC before it is configured.

Please note the following:

1. Switch your computer off. Remove the power cable.
2. Remove the covers from your computer according to the manufacturers instructions.
3. Remove one of the metal covers from a slot at the rear of your PC.
4. Insert the adapter as shown below.
5. Once the adapter has been installed you can test it by means of the test program "diag312.exe" located on the disk SW82Z disk 1/1

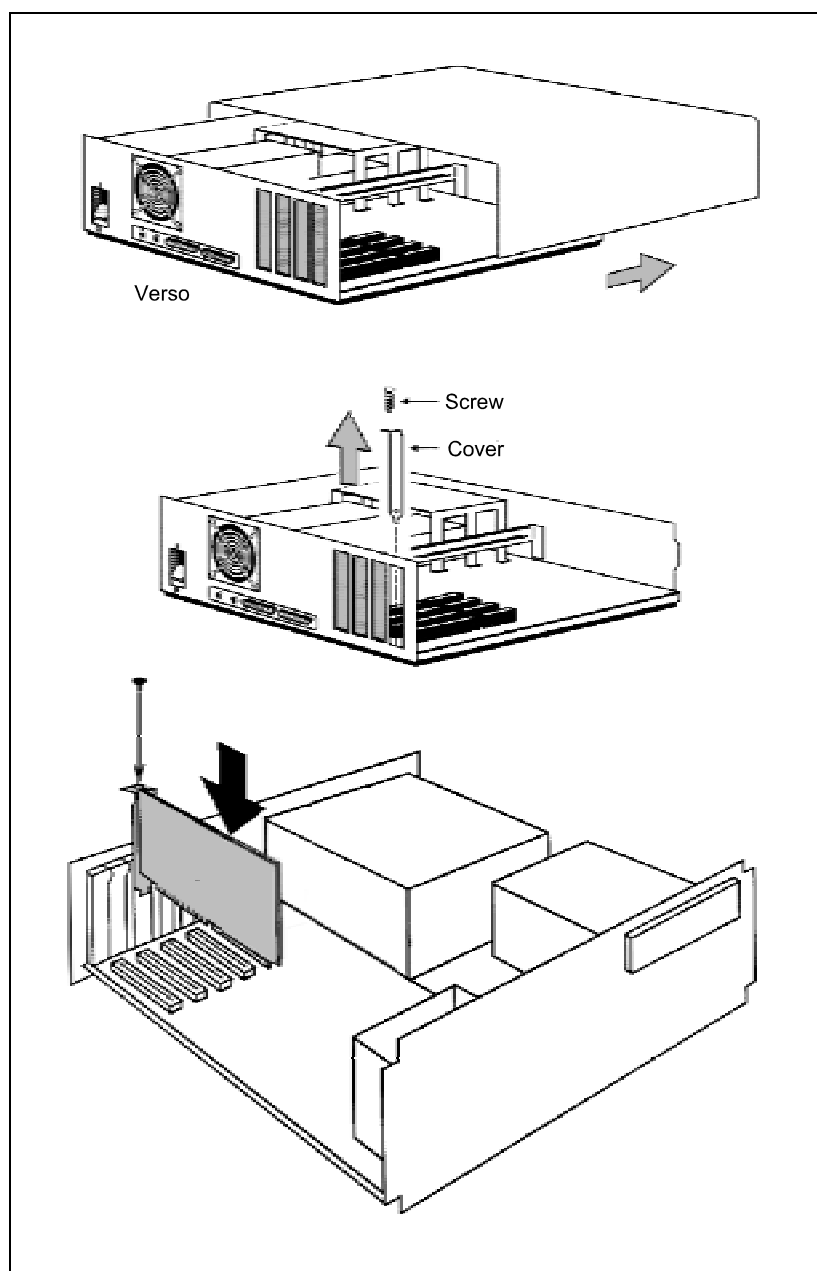


Fig. 3-1: Installation of the Z'nyx adapter

3.1.3.2 Network cabling

3.1.3.2.1 Thin-ethernet network

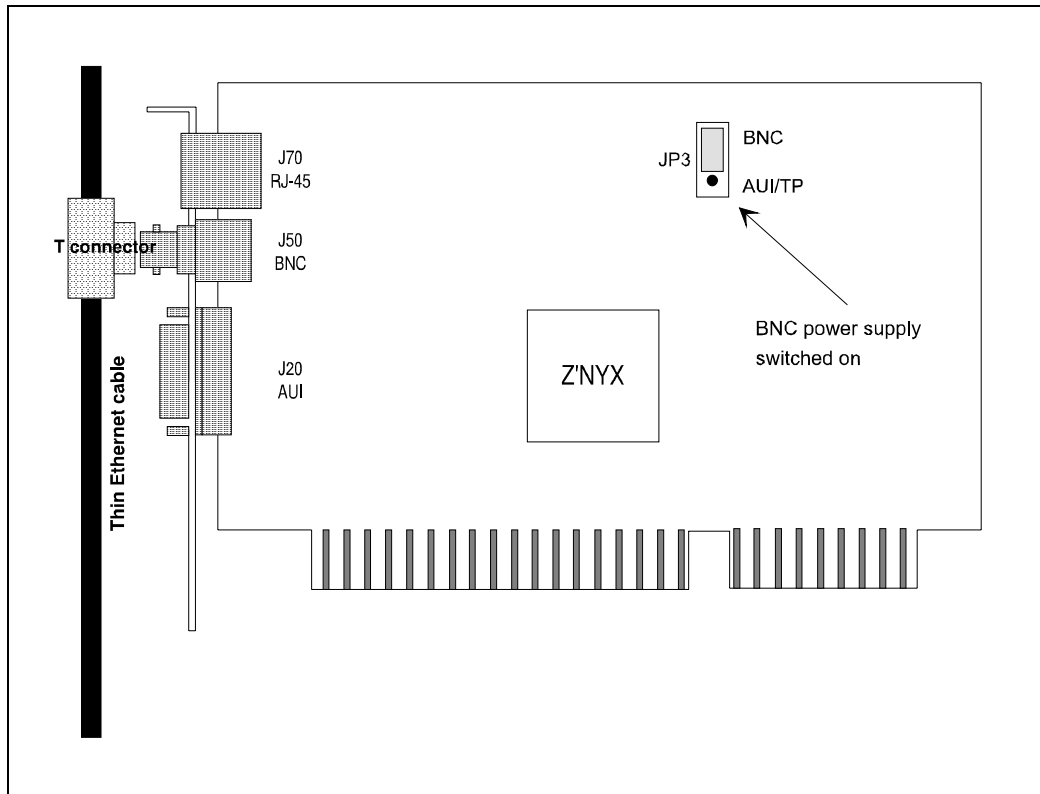


Fig. 3-2: PCI adapter connection to a thin-ethernet-network

3.1.3.2.2 Thick-ethernet network

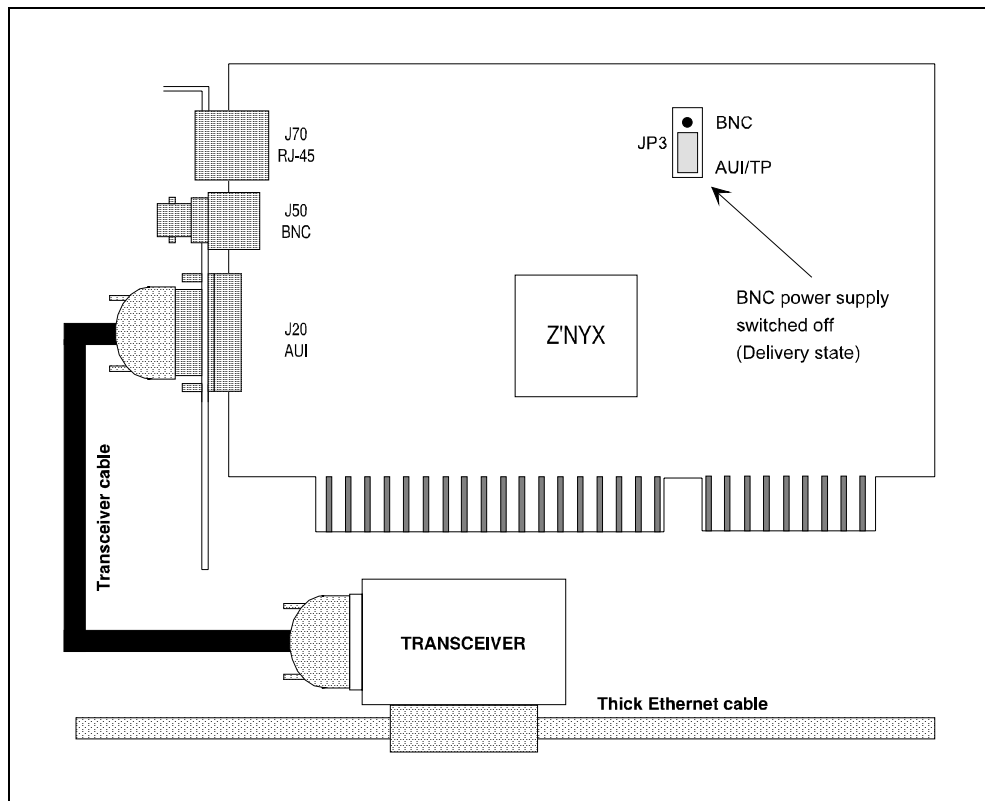


Fig. 3-3: PCI adapter connection to a thick ethernet-network

3.1.3.2.3 UTP-network

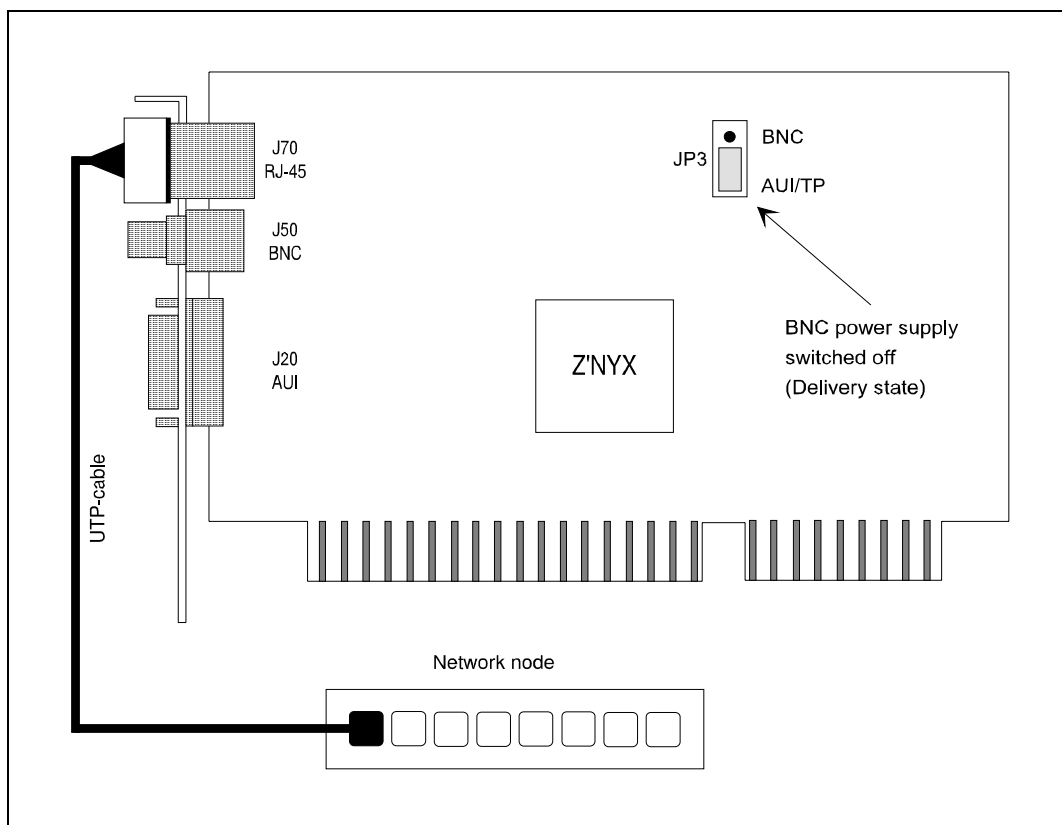


Fig. 3-4: PCI adapter connection to a twisted pair-network

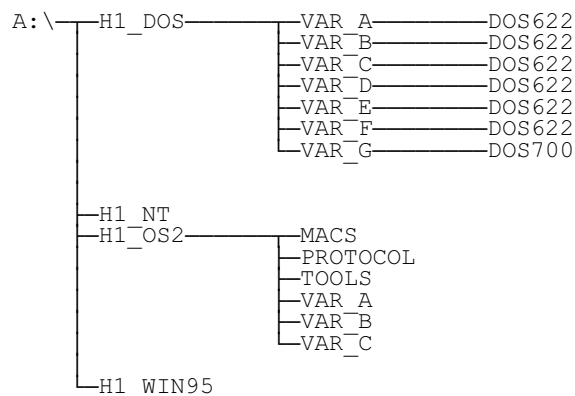
3.1.4 Software installation

The software installation depends on the configuration and on the operating system.

The selected combination depends on the operating system and other required protocols.

Below follow a number of sample installations for certain protocols and operating systems. Other protocols may be installed in a similar manner.

The installation files are located on the disk **SW83Z disk 1/2**



Sample installations

• MS-DOS installation

- | | |
|-----------|---|
| Version A | - H1 under MS-DOS without additional protocols |
| | - H1 under MS-DOS with Windows 3.1 (without workgroups), without additional protocols |
| Version B | - H1 under MS-DOS with Windows for Workgroups |
| Version C | - H1 under MS-DOS with Windows for Workgroups and Novell IPX |
| Version D | - H1 under MS-DOS with IBM LAN Requester 4.0 |
| Version E | - H1 under MS-DOS with IBM LAN Requester 4.0 and Novell IPX |
| Version F | - H1 with 2 network adapters under MS-DOS without additional protocols |
| | - H1 with 2 network adapters under MS-DOS with Windows 3.1 (without Workgroups), without additional protocols |

• OS/2 installation

- | | |
|-----------|---|
| Version A | - H1 under OS/2 without additional protocols |
| Version B | - H1 under OS/2 with IBM LAN Requester 4.0 |
| Version C | - H1 under OS/2 with IBM LAN Requester 4.0 and Novell IPX |

• Windows NT installation

• Windows 95 installation

3.1.4.1 MS-DOS installation

3.1.4.1.1 MS-DOS Version A without additional protocols

Installation with the following components:

- H1 under MS-DOS without additional protocols

Step 1

Install the network adapter into your PC (see chapter 3.1.3)

Step 2

Set the required parameters, if necessary.

Step 3

Copy the files from the directory \H1_DOS\VAR_A located on the disk SW83Z 1/2 to your hard disk C: into the directory \H1_DOS:

```
xcopy a:\h1_dos\var_a\*. * c:\h1_dos\
```

Step 4

Add the following lines to your C:\CONFIG.SYS and C:\AUTOEXEC.BAT files:

additional lines in C:\AUTOEXEC.BAT

```
      :  
      :  
C:\H1_DOS\NETBIND  
      :  
      :
```

additional lines in C:\CONFIG.SYS

```
      :  
      :  
DEVICE=C:\H1_DOS\PROTMAN.DOS /I:C:\H1_DOS  
DEVICE=C:\H1_DOS\ZX312.DOS  
DEVICE=C:\H1_DOS\H1PROT.DOS  
      :  
      :
```

Step 5

Change the entry for NETADDRESS in section [H1PROT_NIF] of the file C:\H1_DOS\PROTOCOL.INI.

Please ensure that the network address is unique.

Extract from the file C:\H1_DOS\PROTOCOL.INI

```
[PROT_MAN]
```

```
DRIVERNAME = PROTMAN$
```

```
[IBMLXCFG]
```

```
ZX312_NIF = ZX312.NIF
```

```
H1PROT_NIF = H1PROT.NIF
```

```
[H1PROT_NIF]
```

```
DRIVERNAME = H1PROT$
```

```
BINDINGS = ZX312_NIF
```

```
;example for a station address (own station address)
```

```
; NETADDRESS = I0020D582FFFF
```

```
MAXVERBINDUNGEN = 10
```

```
MAXSENDEBUFFER = 10
```

```
MAXACKBUFFER = 10
```

```
MSDOSVEKTOR = 0X7D
```

— The leading "I" for the network address ist mandatory!

```
[ZNYX_NIF]
```

```
DRIVERNAME = ZX312$
```

```
IOADDRESS = 0X340
```

Step 6

Restart the PC and make sure that **all programs are loaded without error messages.**

3.1.4.1.2 MS-DOS Version B with Windows for Workgroups

Installation including the following components:

- H1 with Windows for Workgroups (WfW) components

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Set the required parameters, if necessary.

Step 3

When Windows for Workgroups (WfW) is initialized the network configuration is requested automatically. If you have already installed WfW on your PC you must please start the "Network-Setup" program in the program group "Network". The chronological sequence of operations is shown below.

Please select the following items in "Network-Setup":

```
->Network
    -> Install Microsoft Windows Network
    -> No additional Network
->Network
    -> Install Microsoft Windows Network
-> More
-> Driver
    -> Network driver
    -> Add Adapter
        -> Non listed or updated driver
        -> OK
            Please insert the VIPA SW82Z disk 1/1 Z'nyx Ethernet-
            driver- and diagnostic disk into drive A:
        -> OK
        -> select the ZNYX ZX312/V PCI ethernet driver
        -> OK
        -> Settings
            -> Type of driver
                Real-Mode NDIS driver
        -> More
            Check that the settings are correct.
            -> OK
        -> Microsoft NetBEUI
            (save settings)
        -> IPX/SPX Compatible Transport with NetBIOS -> delete
        -> Close
    -> OK terminates Setup.
```

Step 4

restart your PC. WfW should start and run correctly. Once WfW has started the network services are available. You can verify this by means of the file manager.

Step 5

Please copy the H1 directory for Windows multi-protocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_B\*. * C:\H1_DOS\
```

Step 6

Add the entries from the file C:\H1_DOS\CONFIG.SYS to your C:\CONFIG.SYS file

```
LASTDRIVE=p  
device=c:\windows\protman.dos /I:c:\windows  
device=c:\windows\ndishlp.sys  
device=c:\h1_dos\zx312.dos  
device=c:\h1_dos\h1prot.dos
```

Modify any existing LASTDRIVE entry to read LASTDRIVE=P.

Step 7

Check that the following line exists in your C:\AUTOEXEC.BAT file:

```
C:\WINDOWS\net start
```

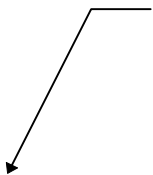
If the entry does not exist you must please enter it at the beginning of the file.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the end of the C:\WINDOWS\PROTOCOL.INI. file

```
[H1Prot_nif]  
DriverName = H1Prot$  
Bindings = ZX$ZX312  
MaxSendeBuffer = 10  
MaxVerbindungen = 10  
MaxAckBuffer = 10  
MsDosVektor = 0x7d  
;NetAddress = I0020D582FFFF
```

The leading "I" for the network address is mandatory!



The file C:\H1_DOS\PROTOCOL.WIN is an example of a valid installation and can possibly be used as is.

```
copy c:\h1_dos\protocol.win c:\windows\protocol.ini
```

Subsequently you must please change the parameter NETADDRESS in the section H1PROT_NIF] of the file C:\WINDOWS\PROTOCOL.INI

Please ensure that the network address is unique!

Step 9

Restart the PC and make sure that **all programs are loaded without error messages**.

Make sure that Windows for Workgroups also starts without error messages.

Note

The installation assumes that WINDOWS is located on your C: drive in a directory C:\WINDOWS. If this is not true all references to WINDOWS in the configuration must be modified to reflect the current directory for WINDOWS.

The configuration was created under MS-DOS 6.22 and WfW 3.11.060.

The directory A:\H1_DOS\VAR_B\DOS622 contains an example of the files AUTOEXEC.BAT, CONFIG.SYS, PROTOCOL.INI, WIN.INI and SYSTEM.INI which was taken from a valid installation.

3.1.4.1.3 MS-DOS Version C with WfW and Novell IPX

The installation includes the following components:

- H1 with Windows for Workgroups (WfW) components
- Novell IPX

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Set the required parameters, if necessary.

Step 3

When Windows for Workgroups (WfW) is initialized the network configuration is requested automatically. If you have already installed WfW on your PC you must please start the "Network-Setup" program in the program group "Network". The chronological sequence of operations is shown below.

Please select the following items in "Network-Setup":

```
->Network
    -> Install the Microsoft Windows Network
    -> No additional Network
->Network
    -> Install Microsoft Windows Network
-> More
-> Driver
    -> Network driver
    -> Add adapter
        -> Non listed or updated driver
        -> OK
            Please insert the VIPA SW82Z disk 1/1 Z'nyx
            ethernet-driver- and diagnostic disk into drive A:
        -> OK
        -> select the ZNYX ZX312/V PCI ethernet driver
        -> OK
        -> Settings
            -> Type of driver
                Real-Mode NDIS driver
        -> More
            Check that the settings are correct.
            -> OK
        -> Microsoft NetBEUI
            (save settings)
        -> IPX/SPX Compatible Transport with NetBIOS -> delete
        -> Close
    -> OK terminates Setup.
```

Step 4

restart your PC. WfW should start and run correctly. Once WfW has started the network services are available. You can verify this by means of the file manager.

Step 5

Please copy the H1 directory for Windows multi-protocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_C\*.* C:\H1_DOS\
```

Step 6

Add the lines from the file C:\H1_DOS\CONFIG.SYS to your C:\CONFIG.SYS file:

```
LASTDRIVE=p
device=c:\windows\protman.dos /I:c:\windows
device=c:\windows\ifshlp.sys
device=c:\h1_dos\zx312.dos
device=c:\windows\ndishlp.sys
device=c:\h1_dos\msipx.sys
device=c:\h1_dos\hlprot.dos
```

Modify any existing LASTDRIVE entry to read LASTDRIVE=P.

Step 7

Check that the following lines exist in the file C:\AUTOEXEC.BAT:

```
C:\WINDOWS\net start
C:\H1_DOS\MSIPX
.....\NETX
```

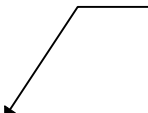
If these lines do not exist add them to the beginning of the file as shown above.

You can obtain the NETX program from your Novell installation disks.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the C:\WINDOWS\PROTOCOL.INI file.

```
[H1Prot_nif]
DriverName = H1Prot$
Bindings = ZX$ZX312
MaxSendeBuffer = 10
MaxVerbindungen = 10
MaxAckBuffer = 10
MsDosVektor = 0x7d
;NetAddress = I0020D582FFFF
[msipx_nif]
DriverName = IPX$
MEDIATYPE = NOVELL/ETHERNET
BINDINGS = ZX$ZX312
```



The leading "I" for the network address is mandatory!

The file C:\H1_DOS\PROTOCOL.WIN is an example of a valid installation and may be used as is.

```
copy c:\h1_dos\protocol.win c:\windows\protocol.ini
```

Subsequently you must please change the NETADDRESS parameter in the section [H1PROT_NIF] of the file C:\WINDOWS\PROTOCOL.INI.

Please ensure that the network address is unique!

Step 9

Restart the PC and make sure that **all programs are loaded without error messages**.

Make sure that Windows for Workgroups also starts without error messages.

Note The installation assumes that WINDOWS is located on your C: drive in a directory C:\WINDOWS. If this is not true all references to WINDOWS in the configuration must be modified to reflect the current directory for WINDOWS.

The configuration was created under MS-DOS 6.22 and WfW 3.11.060.

The directory A:\H1_DOS\VAR_C\DOS622 contains an example of the files AUTOEXEC.BAT, CONFIG.SYS, PROTOCOL.INI, WIN.INI and SYSTEM.INI which was taken from a valid installation.

3.1.4.1.4 MS-DOS Version D with IBM LAN Requester 4.0

This installation includes the following components:

- H1 under MS-DOS
- IBM LAN Requester 4.0

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Set the required parameters, if necessary.

Step 3

Install the IBM LAN Requester according to the instructions supplied by IBM. Next you must please install the "ZNYX ZX312/V PCI Ethernet Driver"-adapter.

Step 4

Restart the PC. The network should operate correctly.

You can check the network function by transmitting a message to another user on the network.

```
net send [name of recipient] ["Message"]
```

Step 5

Copy the H1 directory for Windows multi-protocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_D\*.* C:\H1_DOS\
```

Step 6

Add the lines from the file C:\H1_DOS\CONFIG.SYS to your C:\CONFIG.SYS file:

```
LASTDRIVE=p  
device=c:\net\protman.dos /I:c:\net  
device=c:\net\ndishlp.sys  
device=c:\h1_dos\zx312.dos  
device=c:\h1_dos\hlprot.dos
```

Modify any existing LASTDRIVE entry to read LASTDRIVE=P.

Step 7

Check that the following line exists in your C:\AUTOEXEC.BAT file:

```
C:\net\net start
```

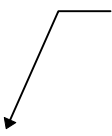
If this line does not exist, please add it to the beginning of the file.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the end of the file C:\NET\PROTOCOL.INI.

```
[H1Prot_nif]
DriverName = H1Prot$
Bindings = ZXSZX312
MaxSendeBuffer = 10
MaxVerbindungen = 10
MaxAckBuffer = 10
MsDosVektor = 0x7d
;NetAddress = I0020D582FFFF
```

The leading "I" for the network address is mandatory!



The file C:\H1_DOS\PROTOCOL.NET contains an example of a valid installation and you may use it as is if required.

```
copy c:\h1_dos\protocol.net c:\net\protocol.ini
```

Subsequently you must please modify the NETADDRESS parameter in the section [H1PROT_NIF] of the file C:\NET\PROTOCOL.INI, if you wish to change your own network address.

Please ensure that the network address is unique!

Step 9

Restart the PC and make sure that **all programs are loaded without error messages**.

Note

It is assumed that the IBM LAN Requester is located on your C: drive in the directory C:\NET . If this is not true you must modify all references to NET in the configuration accordingly.

This configuration was created with MS-DOS 6.22, IBM LAN Requester 4.0.

The directory A:\H1_DOS\VAR_D\DOS622 contains an example of the AUTOEXEC.BAT, CONFIG.SYS and PROTOCOL.INI files of a valid installation.

3.1.4.1.5 MS-DOS Version E with IBM LAN Requester 4.0 and Novell IPX

This installation includes the following components:

- H1 under MS-DOS
- IBM LAN Requester 4.0
- Novell IPX

Install the IBM LAN Requester and Novell NetWare according to the instructions provided by the respective manufacturer.

Install the "ZNYX ZX312/V PCI Ethernet Driver" network adapter.

Now you must please perform the installation as per items 4 to 9 of the MS-DOS version C in chapter 3.1.4.1.3. Use the directory of the IBM LAN Requester instead of the Windows directory.

3.1.4.1.6 MS-DOS Version F with 2 network adapters, no additional protocols

This installation includes the following components:

- H1 with 2 network adapters under MS-DOS, without any additional protocols.

Step 1

Install the network adapter in your PC (see chapter 3.1.3)

Step 2

Set the required parameters, if necessary.

Step 3

Start the diagnostic program diag312.exe from the disk SW82Z disk 1/1. The description of this program is available from the “Getting Started...” section of the Z'NYX manual.

The adapters must be have different I/O base addresses and interrupt request levels.

Step 4

Copy the files from the directory \H1_DOS\VAR_F of the disk SW83Z 1/2 onto your hard disk C: into the directory \H1_DOS:

```
xcopy a:\h1_dos\var_f\*. * c:\h1_dos\
```

Step 5

Please add the following lines to your C:\CONFIG.SYS and C:\AUTOEXEC.BAT files:

Additional lines for C:\AUTOEXEC.BAT

```
:
:
C:\H1_DOS\NETBIND
:
:
```

Additional lines for C:\CONFIG.SYS

```
:
:
DEVICE=C:\H1_DOS\PROTMAN.DOS /I:C:\H1_DOS
DEVICE=C:\H1_DOS\ZX312.DOS
DEVICE=C:\H1_DOS\ZX312.DOS
DEVICE=C:\H1_DOS\H1PROT.DOS
:
:
```

Step 6

Now you must please modify the file C:\H1_DOS\PROTOCOL.INI as follows:

- 1.) the entries in the section [ZX312_NIF] and [ZX3122_NIF]
Use the parameters supplied by the program diag312.exe for the Device specification.
- 2.) the NETADDRESS parameter in section [H1PROT_NIF]

Please ensure that the network address is unique!

Extract from C:\H1_DOS\PROTOCOL.INI

```
[PROT_MAN]

DRIVERNAME = PROTMAN$

[IBMLXCFG]

ZX312_NIF = ZX312.NIF
ZX3122_NIF = ZX3122.NIF
H1PROT_NIF = H1PROT.NIF

[H1PROT_NIF]

DRIVERNAME = H1PROT$
BINDINGS = ZX312_NIF,ZX3122_NIF
; NETADDRESS = I0020D582FFFF,I0020D582FFFE
MAXVERBINDUNGEN = 10,10
MAXSENDEBUFFER = 10,10
MAXACKBUFFER = 10,10
MSDOSVEKTOR = 0X7D,0X7D

[ZX312_NIF]

DRIVERNAME = ZX312$
DEVICE = 14

[ZX3122_NIF]

DRIVERNAME = ZX3122$
DEVICE=15
```

The leading "I" for the network address is mandatory!

Step 7

Restart the PC and make sure that **all programs are loaded without error messages.**

3.1.4.2 OS/2 installation

3.1.4.2.1 OS/2 Version A without additional protocols

This installation includes the following components:

- H1 under OS/2 without additional protocols

Step 1

Install the network adapter into your PC (see chapter 3.1.3)

Step 2

Boot into DOS and set any required parameters

Boot OS/2 because the following steps are applicable only to OS/2.

Step 3

Copy all the files from the directories \H1_OS2\MACS, H1_OS2\PROTOCOL, H1_OS2\TOOLS and H1_OS2\VAR_A of the disk SW83Z 1/2 into the directory \H1_OS2 on hard disk C:

```
xcopy a:\h1_os2\MACS c:\h1_os2\  
xcopy a:\h1_os2\PROTOCOL c:\h1_os2\  
xcopy a:\h1_os2\TOOLS c:\h1_os2\  
xcopy a:\h1_os2\VAR_A c:\h1_os2\
```

Step 4

Add the following lines to your CONFIG.SYS file:

additional lines for C:\CONFIG.SYS

```
:  
:  
DEVICE=C:\H1_OS2\PROTMAN.OS2 /I:C:\H1_OS2  
DEVICE=C:\H1_OS2\ZX312.OS2  
DEVICE=C:\H1_OS2\H1PROT.OS2  
RUN=C:\H1_OS2\NETBIND.EXE  
:  
:
```

Step 5

Now you must edit the NETADDRESS parameter in the section [H1PROT_NIF] of the file C:\H1_OS2\PROTOCOL.INI.

Please ensure that the network address is unique!

Extract from the file PROTOCOL.INI

```
[PROT_MAN]

DRIVERNAME = PROTMAN$

[IBMLXCFG]


ZX312_NIF = ZX312.NIF
H1PROT_NIF = H1PROT.NIF

[H1PROT_NIF]

DRIVERNAME = H1PROT$
BINDINGS = ZX312_NIF
;Example for a station address (own station address)
; NETADDRESS = I00001C011010
MAXVERBINDUNGEN = 32
MAXSENDEBUFFER = 32
MAXACKBUFFER = 64

[ZX312_NIF]

DRIVERNAME = ZX312$
```



The leading "I" for the network address is mandatory!

Step 4

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages**.

3.1.4.2.2 OS/2 Version B with IBM LAN Requester 4.0

This installation includes the following components:

- H1 under OS/2
- IBM LAN Requester 4.0

Note you must have the IBM LAN Requester!

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Boot into DOS and set any required parameters

Boot OS/2 because the following steps are applicable only to OS/2.

Step 3

Install the IBM LAN Requester 4.0 as shown below:

```

:
In the window "Network Adapters"
-> Other adapters (select)
    Insert the disk SW83Z 1/2
    -> A:\H1_OS2\MACS (enter)
        The files will be copied
In the window "Network Adapters"
    VIPA CP 1413plus (Znyx ZX312/V PCI Ethernet Driver) (select)
    -> Add
In the window "Current Configuration"
-> Edit
    enter the hardware settings
In the window "Protocols"
-> other Protocols (select)
    -> A:\H1_OS2\PROTOCOL (enter)
        -> OK
            (the driver is copied to the hard disk)
            VIPA H1 Protokoll (select)
            -> Add
                select any other protocol,
                that must be installed.
            -> OK
```

Step 4

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages**.

Note The directory A:\H1_OS2\VAR_B contains the CONFIG.SYS and PROTOCOL.INI files of a valid installation. You may wish to refer to them if you require help.

3.1.4.2.3 OS/2 Version C with IBM LAN Requester 4.0 and Novell IPX

This installation includes the following components:

- H1 under OS/2
- IBM LAN Requester 4.0
- Novell IPX

Note you must have the IBM LAN Requester and the Novell NetWare drivers for OS/2!

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Boot into DOS and set any required parameters

Boot OS/2 because the following steps are applicable only to OS/2.

Step 3

Install the IBM LAN Requester 4.0 if you have not already done so. Set the "Protocols" parameter to "IBM NetWare Requester Support".

Step 4

Once you have rebooted your PC and if the network operates properly you can start the "install" program located in the directory "ibmcom".

Proceed as follows:

```
-> Configure
-> select LAN adapters and protocols
    ->configure
        for "Protocols"
            -> other Protocols
                -> A:\H1_OS2\PROTOCOL (enter)
                ->OK
                    (The driver is copied to your hard disk)
                select the VIPA H1 protocol
            -> Add
        -> OK
-> close
```

Now you must also install the IPX drivers and link them by means of NDIS.

Step 5

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages**.

Note The directory A:\H1_OS2\VAR_C contains the CONFIG.SYS and PROTOCOL.INI files of a valid installation. You may wish to refer to them if you require help.

3.1.4.3 Windows NT installation

Installation procedure for expert users

A script file is available for the installation of the H1 drivers for Windows NT. This script file will be processed as soon as the H1 driver is installed for the first time.

Please proceed as follows:

Step 1

Install the MAC-driver for the network adapter

Step 2

Insert the H1 driver disk into drive A: and select "other protocol" from the window. A further selection panel will be displayed.

Step 3

Select "A:\H1_NT". The disk will be read and the selection panel will display "VIPA H1-Protokoll". Please select this protocol.

Step 4

If you have installed a number of network adapters and some of these are not VIPA CP1413plus adapters, you must remove the link between the H1 protocol driver and all non-VIPA adapters. Otherwise the H1 driver will not operate properly.

Installing H1prot.sys for WindowsNT

Step 1

Please install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.1.

Note you may have to change certain settings. We recommend that you set the address to 340h if this is unused in your PC.

Step 2

Install the network adapter as follows:

- start the "Control Panel" in the "Main" group.
- select "Network" in the "Control Panel".
- in "Network Settings" select "Add adapter".
- select the "DEC DC21040 PCI Ethernet Controller" from the list.
- The type of cable may be set to "Auto Detection".

Step 3

Install the H1_Prot driver as described below:

1. Copy the "**h1prot.sys**" driver from \H1_NT directory of the SW83Z 1/2 disk to your hard disk C: into the \WINNT35 directory of Windows NT:

```
copy a:\H1_NT\H1prot.sys  
c:\winnt35\system32\drivers
```

2. Make the following entries into the "Registry".

These entries are located in the file "H1ZNYX35.ini" in the directory "\H1_NT" of the VIPA installation disk 1/2.

You can make these entries by means of the "Regini.exe" tool:

```
a:\H1_NT\regini a:\H1_NT\H1ZNYX35.ini
```

Step 4

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages**.

Note

If the parameter "cr.fehler" contains 13 decimal (H1_NO_ADAPTER) you should change the "Bind"-parameter in the H1-protocol selection of the NT-System.

At newer NT-versions there is a change of the device name.

To do:

1. Please start the registry editor (regedt32.exe).

2. Chose the following path:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\DC21X4
```

3. There is the following entry in the directory Linkage:

```
Export: REG_MULTI_SZ:\Device\DC21X41
```

4. Please note the "export" device name e.g. "DC21X41".

5. Now transfer this name to the "Bind" device name of the H1Prot-definition.

You have to chose the following path:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\H1Prot
```

Transfer the noted name in the directory Linkage:

```
Bind: REG_MULTI_SZ:\Device\DC21X41
```

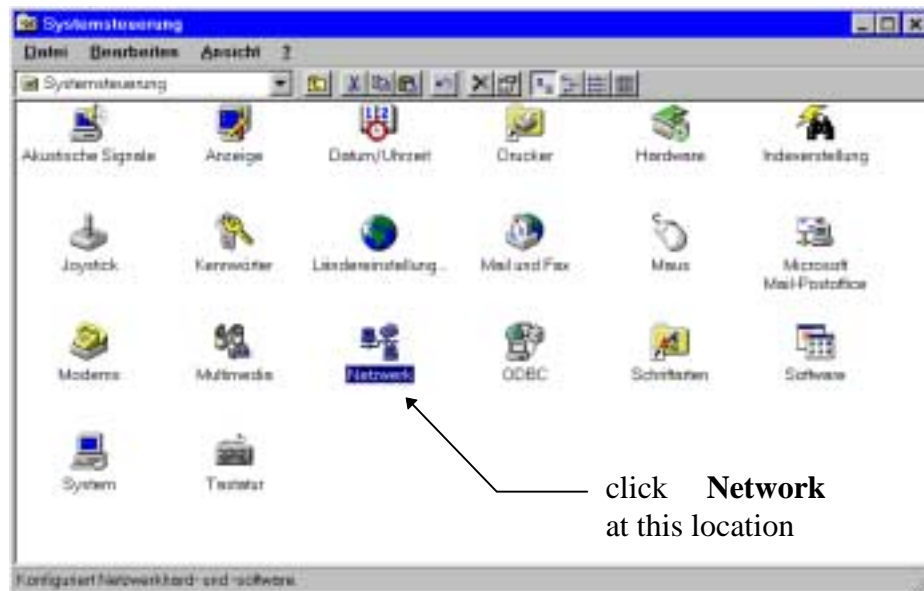
3.1.4.4 Windows 95 installation

Step 1

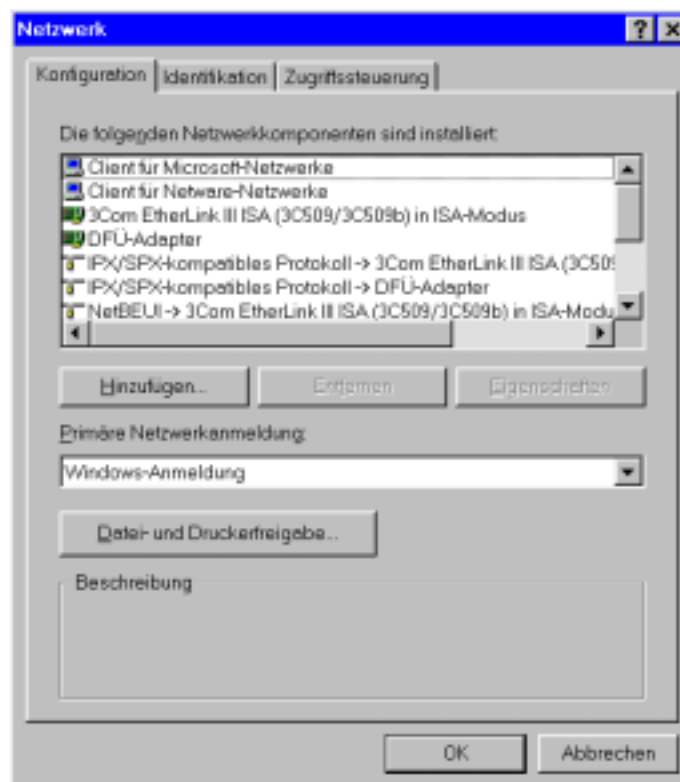
Chapter 3.2.3 contains the description of the hardware installation

Step 2

Start Windows. Select “Settings” on the Start menu. Select “Control Panel” and the window *Control Panel* will be displayed.

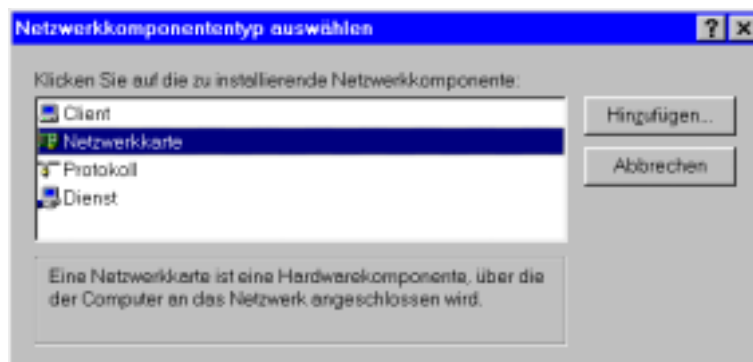


Execute the “Network” program from this window. The *Network* panel will be displayed.



Step 3

Click on the button **[Add]**. The panel for selecting *Network components* will be displayed. You may now enter the VIPA network adapter and the VIPA-H1 protocol driver. First you must enter the network adapter. Select "Adapter" and click on **[Add]**.

**Step 4**

Select "Znyx" from the list of manufacturers. The list of adapters will then display all adapters that are available from Z'nyx. Select the following adapter and then click on **[OK]**:

"Znyx ZX312/314 EtherAction PCI LAN Adapter"



The installation of the adapter has now been completed. The following steps define the respective protocol.

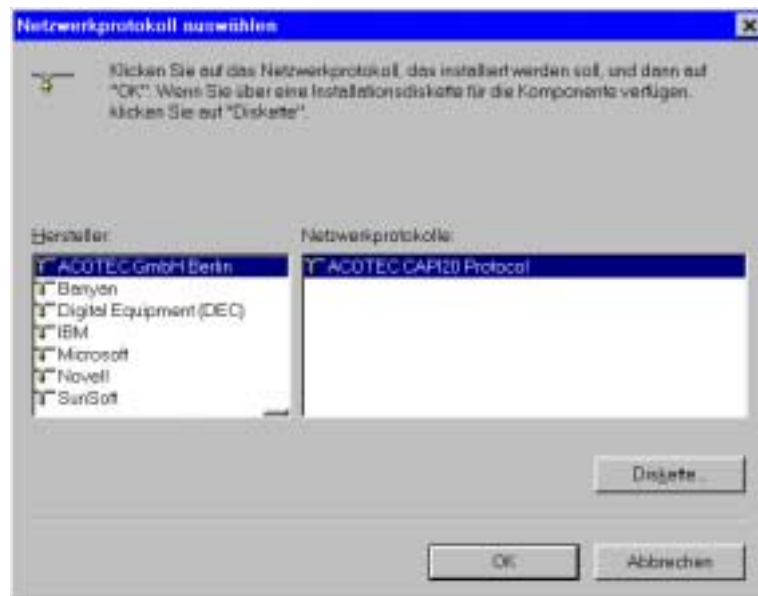
Step 5

Your H1 system requires that you make the H1 protocol available. Select the manufacturer "3Com" from the *Network* panel and click on **[Add]**.

The window *Select network component type* will appear. Enter the H1 protocol into the window by selecting "Protocol" and clicking on **[Add]**. The window *Select network protocol* will be displayed.

Step 6

Insert the H1 driver disk and click on **[Have Disk...]**.



Click on **[Browse...]** in the window *Install From Disk*.

In the window *Open* select the file "H1PROT.INF" from the directory A:\H1_WIN95 and click **[OK]**.

The window *Install From Disk* now shows "A:\H1_WIN95". Click **[OK]**. The respective protocol will be installed.

Select "VIPA H1-protocol" from the *Network Protocol* window and click **[OK]**.

Click **[OK]** in the *Network* window. At this point you have completed the installation of the adapter and the H1-protocol.



If you have installed network adapters that are not VIPA CP1413plus adapters, you must remove the binding between the H1-protocol driver and all non-VIPA adapters. Otherwise the H1-driver will not operate properly.

3.2 3COM ISA-bus adapter

3.2.1 Properties

- Downwards compatible with drivers for (3C509) EtherLink III ISA adapters.
- Automatic installation of the Novell NetWare DOS ODI client software into the operating system.
- Auto Select Media Type (automatic recognition of the socket that is connected to the network).
- Network management software is installed when the driver is installed.
- 16 Bit AT-Bus
- The adapter carries a BNC (10 Base 2), an AUI (10 Base 5) and an RJ-45 Phone Jack (10 Base T)
- Socket for remote boot - EPROM

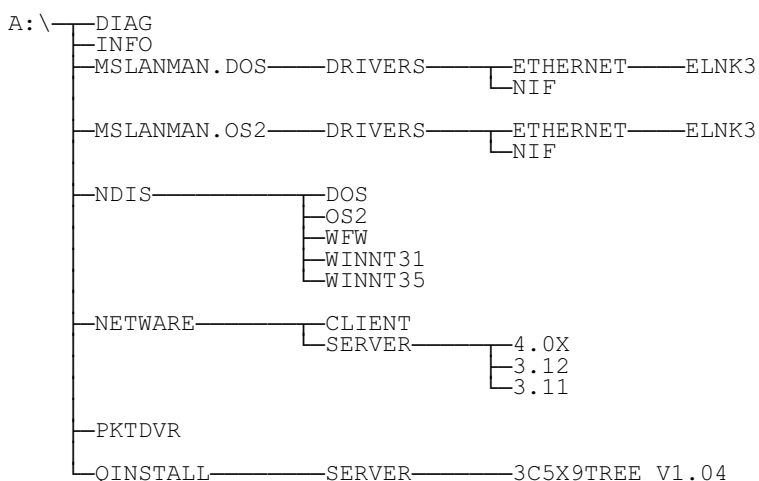
3.2.2 Shipment

Hardware

- 16-bit network adapter
- BNC T-piece

Software

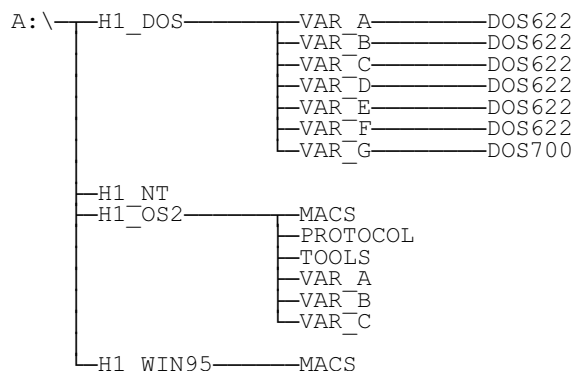
Disk 1: SW82C disk 1/1, 3Com Ethernet driver



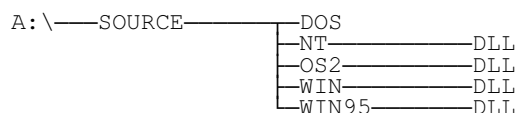
Disk 2: SW83C disk 1/2, PC adapter 1413plus H1-LAN

- Driver software H1 for the following operating systems :

- MS-DOS® 5.0 and 6.22
- Windows® 3.1 and WfW® 3.11
- Windows® NT 3.5, 3.51
- Windows® 95
- OS/2® 2.1 and OS/2® Warp

**Disk 3: SW83C disk 2/2, PC adapter 1413plus H1-LAN**

- Sources



As of version 2.6 the files required for the different operating systems are compressed into a separate archive on the disk. Common files that are used by all operating systems have not been archived.

To extract the files required for your specific operating system you must copy the respective file to your hard disk and change to that directory. Now you must enter the command:

A:\PKUNZIPF <file name>

“file name” represents the file that you wish to extract.

Example: A:\PKUNZIPF DOS.ZIP

The extraction program may be executed in MS-DOS version 3.0 or higher. You can also execute the extraction program in the DOS shell of Windows 95, Windows NT and OS2.

3.2.3 Hardware installation

3.2.3.1 Installing the adapter

Before you can configure the adapter you must install it in the PC.

Please note the following:

1. Switch your computer off. Remove the power cable.
2. Remove the covers from your computer according to the manufacturers instructions.
3. Remove one of the metal covers from a slot at the rear of your PC.
4. Insert the adapter as shown below.

Once the adapter has been installed you can start the configuration from the disk SW82C disk 1/1

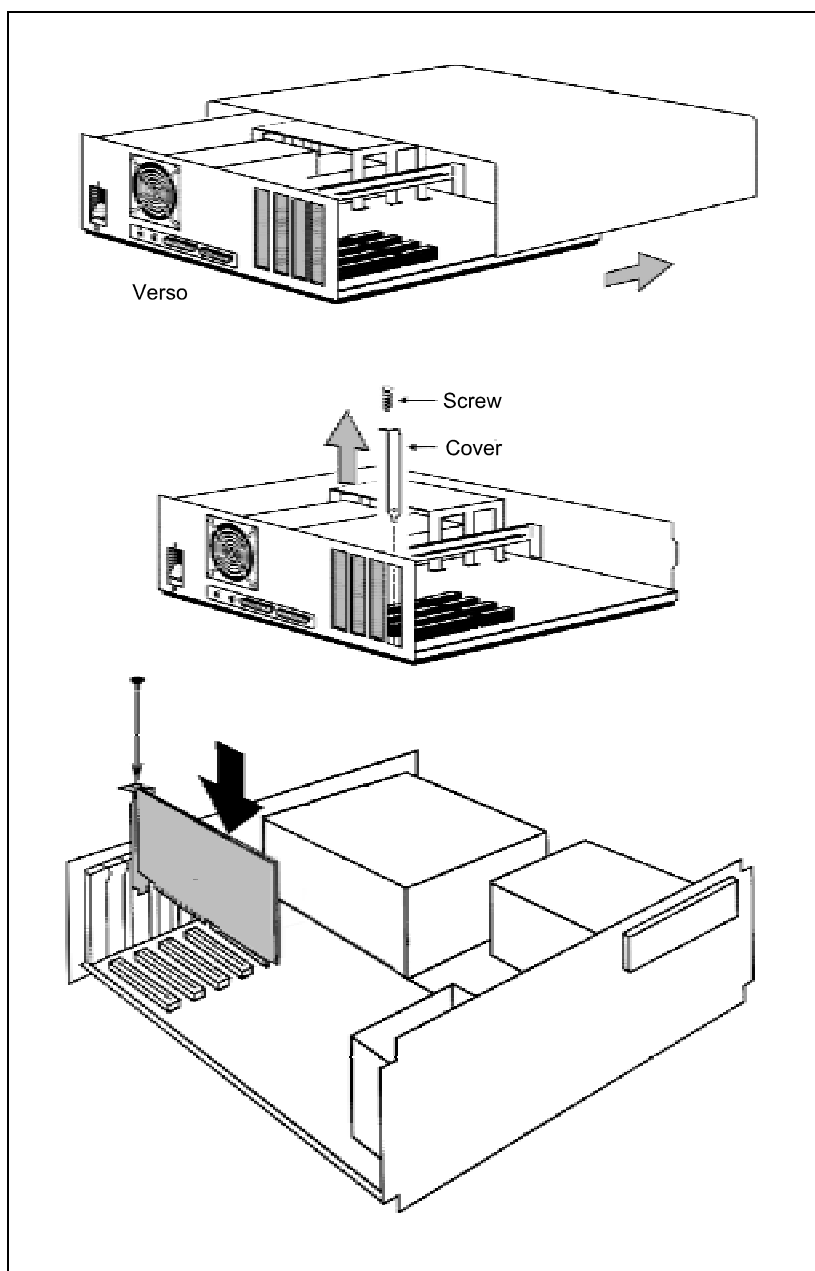


Fig. 3-5: Installation of the 3Com adapter

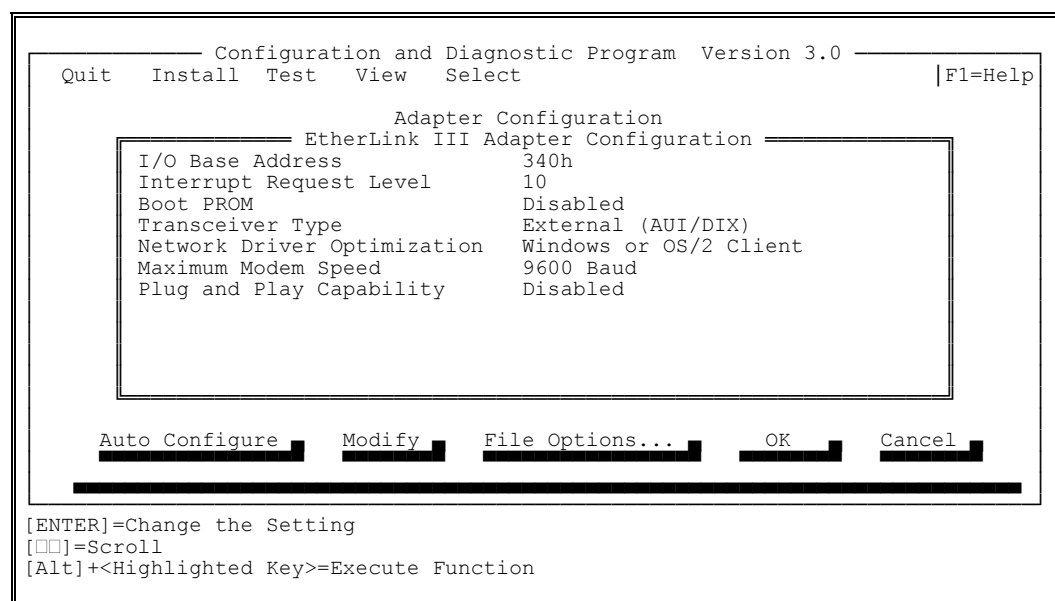
3.2.3.2 Default settings

The default settings supplied by VIPA guarantee a smooth installation of the software drivers. You should not have to modify these settings. If the default settings must be changed these modifications should only be made by qualified personnel.

The adapter is configured by means of software. Start the configuration program `3c5x9cfg.exe` located on disk 1/1 of "SW82C".

If the program does not operate properly you may find further information in the text files located on the disk.

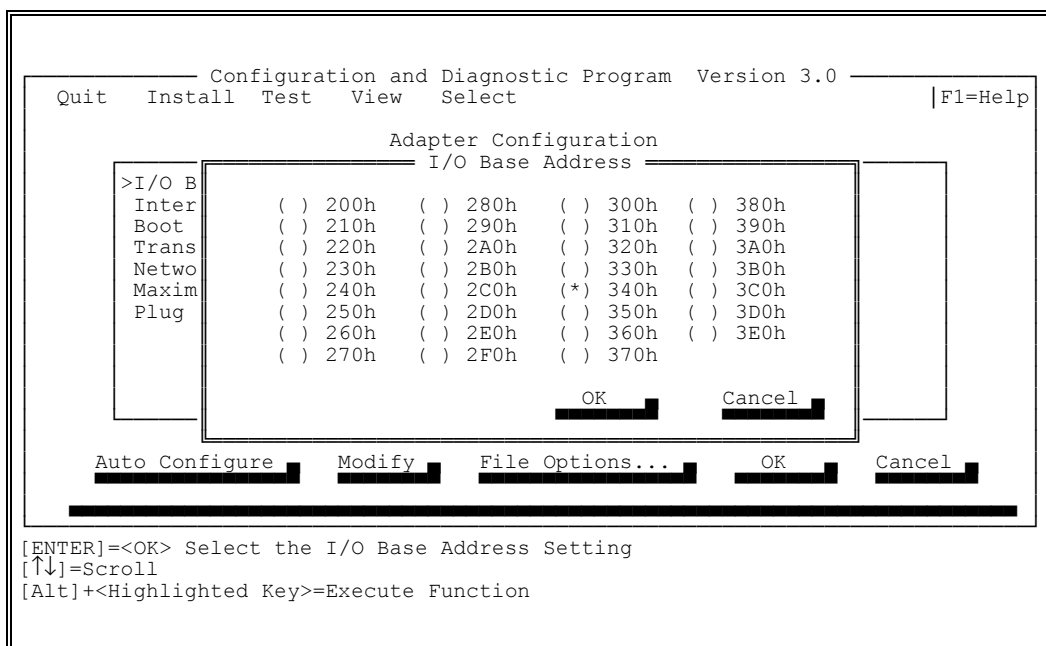
From the "Install" menu select the "Configure Adapter... (F4)" function. The following panel will be displayed:



All parameters may be accessed individually via this panel.

3.2.3.3 I/O base address selection

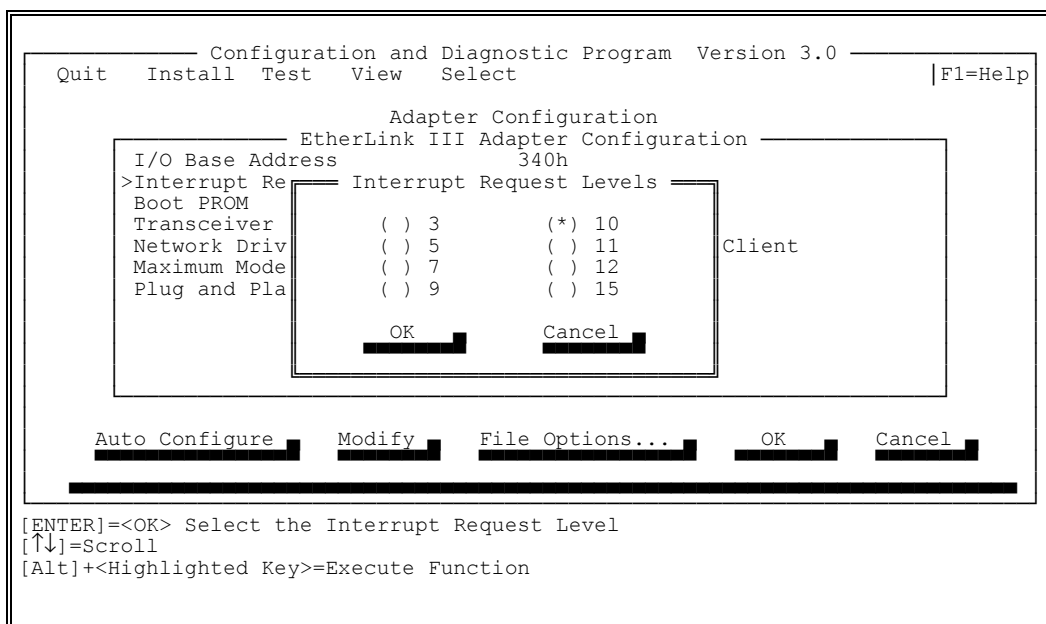
200h and 3E0h in steps of ten:



The selected base address may not be in use.

3.2.3.4 Selection of the Interrupt Request Level

The Interrupt Request Level may be selected from the following set of values:

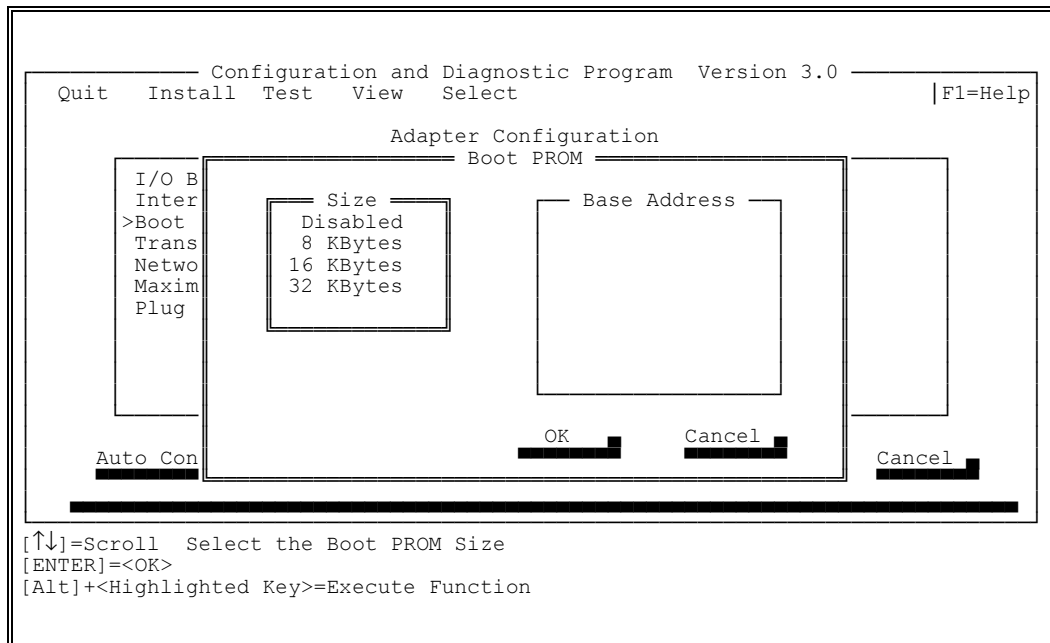


The selected Interrupt Request Level may not be in use.

3.2.3.5 Boot PROM settings

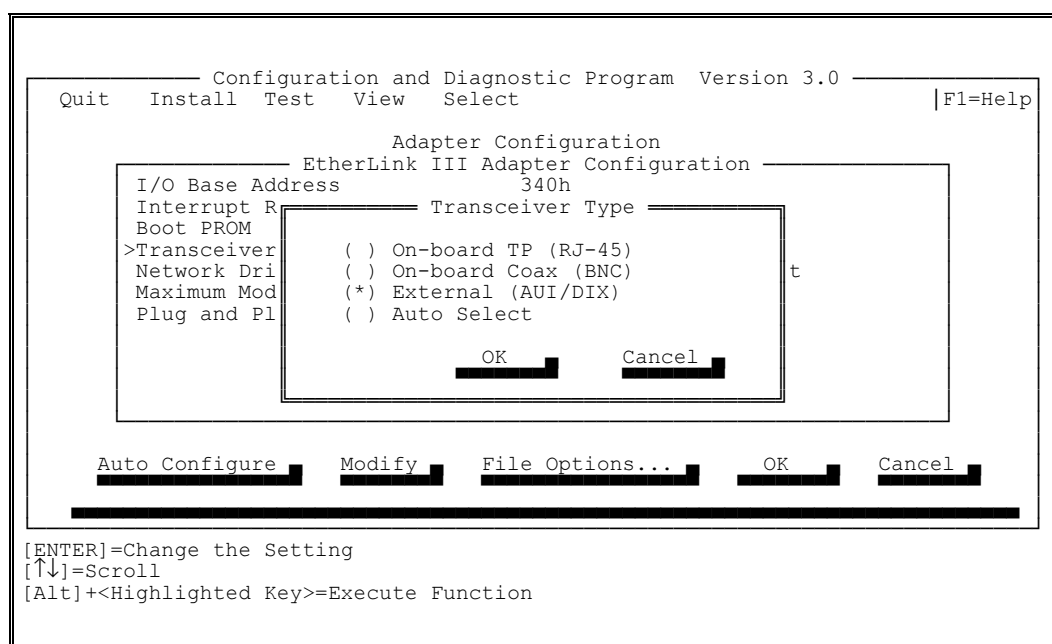
Where it is required that the workstation should be booted from the hard disk of the file server a boot PROM must be installed to enable the remote boot operation.

1. Carefully insert the PROM into the respective socket.
Make sure that the orientation of the PROM is correct
2. In the menu select “Boot PROM” to indicate the presence of the boot PROM.



3.2.3.6 Selecting the type of transceiver

Use the panel “Transceiver Type” to select the type of communication medium:



3.2.3.7 Network driver optimization

Use this panel to select whether the driver is used for DOS, Windows, OS/2 or for a server:

The screenshot shows the 'Configuration and Diagnostic Program Version 3.0' interface. The main menu includes 'Quit', 'Install', 'Test', 'View', and 'Select'. The 'Select' option is highlighted, leading to the 'Adapter Configuration' screen. Within this screen, the 'EtherLink III Adapter Configuration' window is open, showing 'I/O Base Address' as '340h'. The 'Network Driver Optimization' sub-window is active, displaying the prompt 'Optimize the network driver for a:' with three radio button options: '() DOS Client', '(*) Windows or OS/2 Client' (which is selected), and '() Server'. At the bottom of this sub-window are 'OK' and 'Cancel' buttons. Below the sub-window, the main window has buttons for 'Auto Configure', 'Modify', 'File Options...', 'OK', and 'Cancel'. At the very bottom, a legend explains the controls: '[ENTER]=<OK> Select the Network Driver Optimization Setting', '[↑↓]=Scroll', and '[Alt]+<Highlighted Key>=Execute Function'.

3.2.3.8 Setting the maximum communication speed for Modems

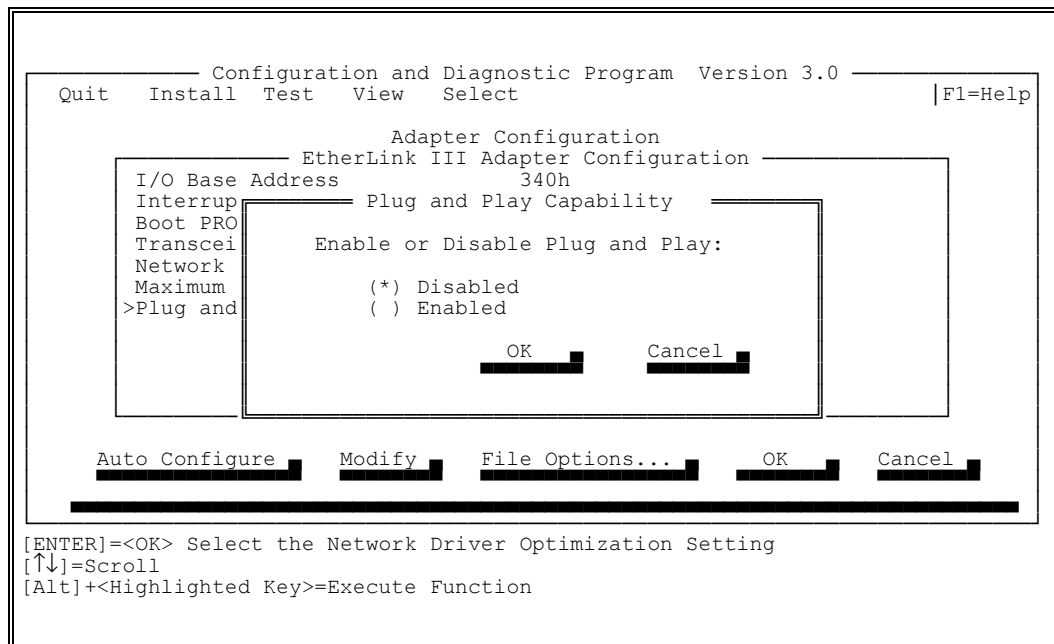
This panel may be used to determine the maximum speed at which a modem may operate on your PC:

The screenshot shows the same 'Configuration and Diagnostic Program Version 3.0' interface. The 'Select' option is highlighted, leading to the 'Adapter Configuration' screen. The 'EtherLink III Adapter Configuration' window is open, showing 'I/O Base Address' as '340h'. The 'Maximum Modem Speed' sub-window is active, displaying the prompt 'Maximum Modem Speed' with two columns of radio button options. The first column has '() No Modem', '() 1200 Baud', and '() 2400 Baud'. The second column has '(*) 9600 Baud' (which is selected), '() 19,200 Baud', and '() 38,400 Baud'. At the bottom of this sub-window are 'OK' and 'Cancel' buttons. Below the sub-window, the main window has buttons for 'Auto Configure', 'Modify', 'File Options...', 'OK', and 'Cancel'. At the very bottom, a legend explains the controls: '[ENTER]=<OK> Select the Modem Speed Setting', '[↑↓]=Scroll', and '[Alt]+<Highlighted Key>=Execute Function'.

This parameter is not required if you do not have a modem connected to your PC.

3.2.3.9 Plug and Play settings

You can select Plug and Play options by means of this panel. If you have selected "Enable" the adapter can be installed into systems supporting Plug and Play hardware without user intervention.



3.2.3.10 Network cabling

3.2.3.10.1 Thin-ethernet network

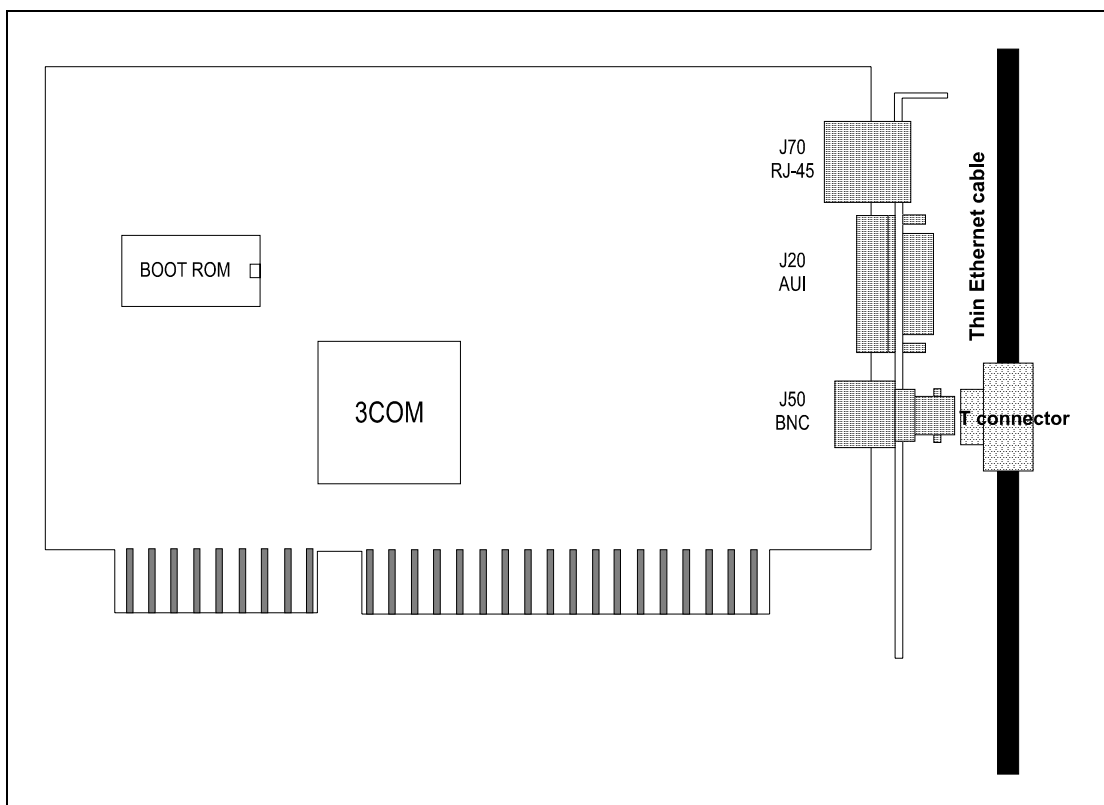


Fig. 3-6: Selecting "On-board Coax (BNC)"

3.2.3.10.2 Thick-ethernet network

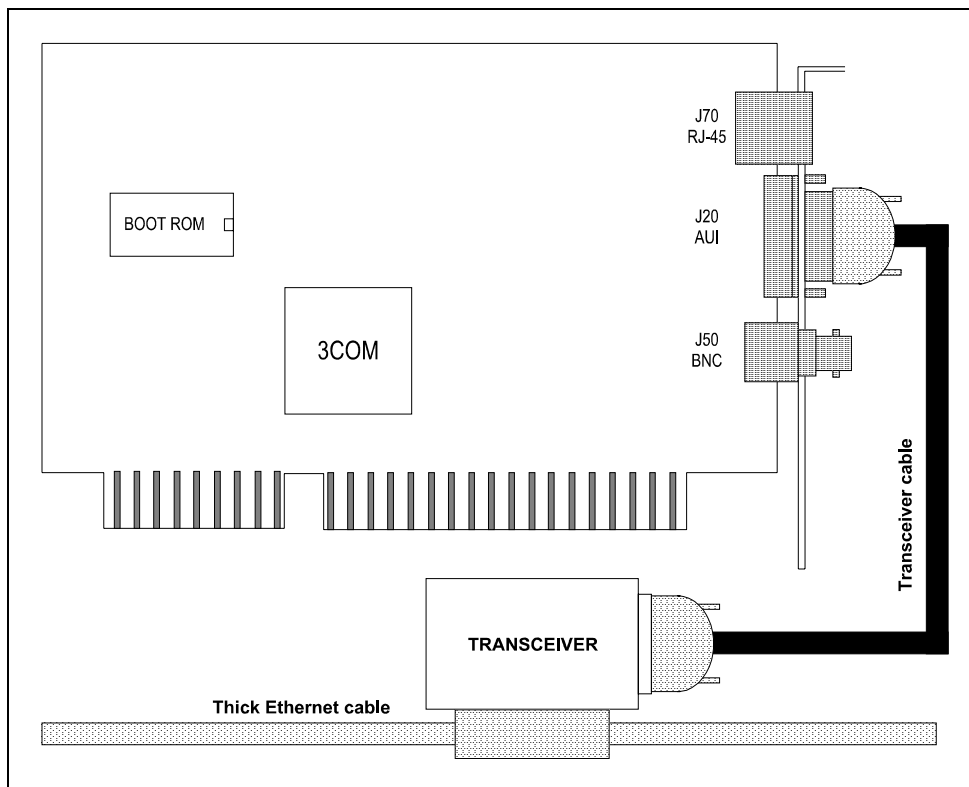


Fig. 3-7: Selecting "External (AUI/DIX)"

3.2.3.10.3 UTP-network

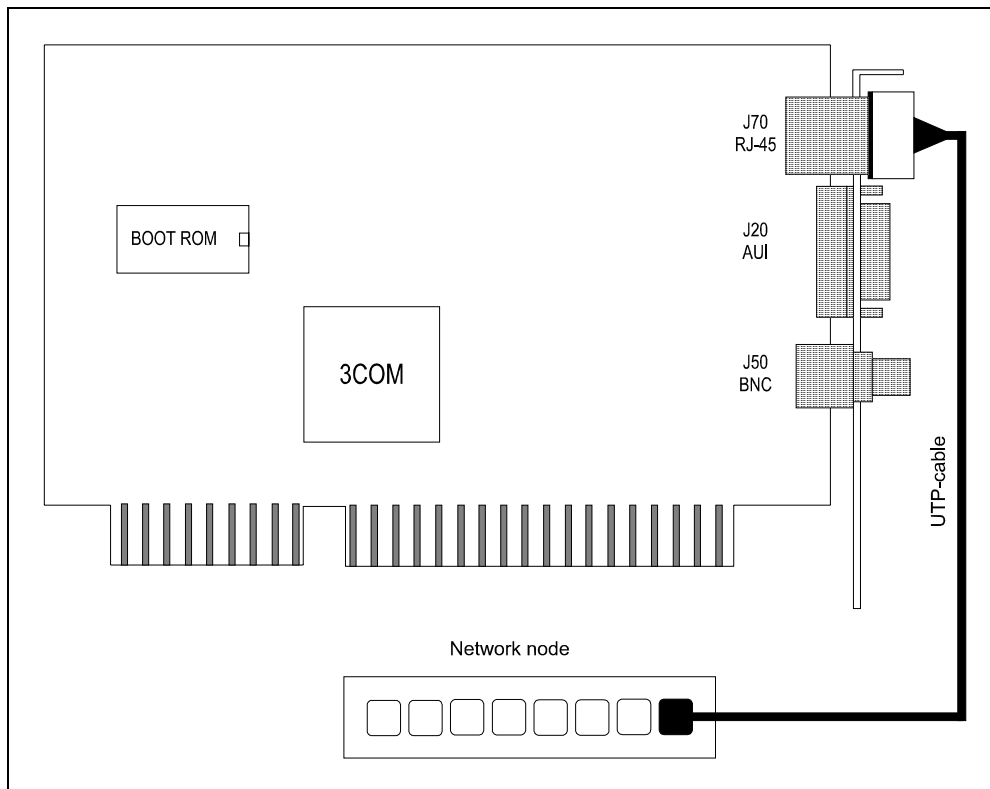


Fig. 3-8: Selecting "On-board TP (RJ-45) "

When you tick "Auto Select" the adapter will determine the type of cable automatically.

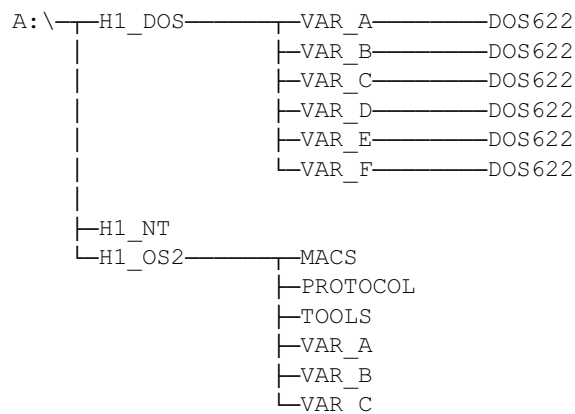
3.2.4 Software installation

The software installation depends on the configuration and on the operating system.

The selected combination depends on the operating system and other required protocols.

Below follow a number of sample installations for certain protocols and operating systems. Other protocols may be installed in a similar manner.

The installation files are located on the disk SW83C disk 1/2



Example of an installation

• MS-DOS installation

- | | |
|-----------|---|
| Version A | - H1 under MS-DOS without additional protocols |
| | - H1 under MS-DOS with Windows 3.1 (without Workgroups), without additional protocols |
| Version B | - H1 under MS-DOS with Windows for Workgroups |
| Version C | - H1 under MS-DOS with Windows for Workgroups and Novell IPX |
| Version D | - H1 under MS-DOS with IBM LAN Requester 4.0 |
| Version E | - H1 under MS-DOS with IBM LAN Requester 4.0 and Novell IPX |
| Version F | - H1 with 2 network adapters under MS-DOS without additional protocols |
| | - H1 with 2 network adapters under MS-DOS with Windows 3.1 (without Workgroups), without additional protocols |

• OS/2 installation

- | | |
|-----------|---|
| Version A | - H1 under OS/2 without additional protocols |
| Version B | - H1 under OS/2 with IBM LAN Requester 4.0 |
| Version C | - H1 under OS/2 with IBM LAN Requester 4.0 and Novell IPX |

• Windows NT installation

• Windows 95 installation

3.2.4.1 MS-DOS installation

3.2.4.1.1 MS-DOS Version A without additional protocols

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 under MS-DOS without additional protocols

Step 1

Install the adapter into your PC (see chapter 3.2.3.1)

Step 2

Set the required parameters, if necessary.

Step 3

Copy the files located in the subdirectory \H1_DOS\VAR_A of the disk SW83C 1/2 to the hard disk C: into the subdirectory \H1_DOS:

```
xcopy a:\h1_dos\var_a\*.* c:\h1_dos\
```

Step 4

Add the following lines to the C:\CONFIG.SYS and C:\AUTOEXEC.BAT files:

additional lines in C:\AUTOEXEC.BAT

```
      :  
      :  
C:\H1_DOS\NETBIND  
      :  
      :
```

additional lines in C:\CONFIG.SYS

```
      :  
      :  
DEVICE=C:\H1_DOS\PROTMAN.DOS /I:C:\H1_DOS  
DEVICE=C:\H1_DOS\ELNK3.DOS  
DEVICE=C:\H1_DOS\H1PROT.DOS  
      :  
      :
```

Step 5

Next you must please edit the file C:\H1_DOS\PROTOCOL.INI

1. Change the hardware settings in the section [ELNK3_NIF]

Please ensure that the IOADDRESS is not being used by another adapter.

2. Change the NETADDRESS entry in the section [H1PROT_NIF]

Please ensure that the network address is unique.

Excerpt from C:\H1_DOS\PROTOCOL.INI

```
[PROT_MAN]
```

```
DRIVERNAME = PROTMAN$
```

```
[IBMLXCFG]
```

```
ELNK3_NIF = ELNK3.NIF
```

```
H1PROT_NIF = H1PROT.NIF
```

```
[H1PROT_NIF]
```

```
DRIVERNAME = H1PROT$
```

```
BINDINGS = ELNK3_NIF
```

```
;Sample a station address (own station address)
```

```
; NETADDRESS = I0020D582FFFF
```

```
MAXVERBINDUNGEN = 10
```

```
MAXSENDEBUFFER = 10
```

```
MAXACKBUFFER = 10
```

```
MSDOSVEKTOR = 0X7D
```

↖ The leading "I" for the network address is mandatory!

```
[ELNK3_NIF]
```

```
DRIVERNAME = ELNK3$
```

```
IOADDRESS = 0X340
```

Step 6

Restart the PC and make sure that **all programs are loaded without error messages.**

3.2.4.1.2 MS-DOS Version B with Windows for Workgroups

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 with Windows for Workgroups (WfW) components

Step 1

Please install the VIPA CP1413plus H1 network adapter as described in chapter 3.2.3.1.

Step 2

Set the required parameters, if necessary.

Step 3

When Windows for Workgroups (WfW) is installed for the first time, the network configuration function will be executed automatically. Should you have an existing WfW installation you must please start the program "Network Setup" in the "Network" group of WfW. The chronological sequence of operations required to configure the network in Windows for Workgroups are depicted below.

Please select the following items in "Network-Setup":

```
->Network
    -> install the Microsoft Windows Network
    -> No additional network
->Network
    -> install the Microsoft Windows Network
-> More
-> Driver
    -> Network driver
    -> Add 3COM Etherlink III-Adapter
        -> Settings          (automatic or not used)
            Interrupt
            Base I/O-interface (please enter)
            Driver type
            Real-Mode NDIS Driver
        -> IPX/SPX Compatible Transport with NetBIOS -> remove
-> Microsoft NetBEUI          (save settings)
-> OK terminates setup.
```

Step 4

Please restart your PC. WfW should start and run correctly. Once WfW is running the network services are available. You can use the file manager to test the network services.

Step 5

Please copy the H1 directory for the Windows Multiprotocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_B\*.* C:\H1_DOS\
```

Step 6

Add the lines contained in the file C:\H1_DOS\CONFIG.SYS to the C:\CONFIG.SYS file

```
LASTDRIVE=p
device=c:\windows\protman.dos /I:c:\windows
device=c:\windows\ndishlp.sys
device=c:\h1_dos\elnk3.dos
device=c:\h1_dos\hlprot.dos
```

If the LASTDRIVE parameter exists, change it to read LASTDRIVE=P.

Step 7

Check that the C:\AUTOEXEC.BAT file contains the following line:

```
C:\WINDOWS\net start
```

If this line is not present, add it to the beginning of the file.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the file C:\WINDOWS\PROTOCOL.INI.

```
[H1Prot_nif]
DriverName = H1Prot$
Bindings = MS$ELNK3
MaxSendeBuffer = 10
MaxVerbindungen = 10
MaxAckBuffer = 10
MsDosVektor = 0x7d
;NetAddress = I0020D582FFFF
```

The leading "I" for the network address is mandatory!

The file C:\H1_DOS\PROTOCOL.WIN contains an example of a valid installation and may be used as is.

```
copy c:\h1_dos\protocol.win c:\windows\protocol.ini
```

Subsequently you must please change the NETADDRESS setting in the section [H1PROT_NIF] of the file C:\WINDOWS\PROTOCOL.INI.

Please ensure that the assigned network address is unique.

Step 9

Restart the PC and make sure that **all programs are loaded without error messages**. Also check for errors during the start up of Windows for Workgroups.

Note This installation requires that WINDOWS is installed on the C: drive in the directory C:\WINDOWS. If not, all references to WINDOWS must be changed in the configuration section.

The configuration was generated with MS-DOS 6.22, WfW 3.11.060.

The directory A:\H1_DOS\VAR_B\DOS622 contains an example of the AUTOEXEC.BAT, CONFIG.SYS, PROTOCOL.INI, WIN.INI and SYSTEM.INI files from a valid installation.

3.2.4.1.3 MS-DOS Version C with WfW and Novell IPX

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 with Windows for Workgroups (WfW) components
- Novell IPX

Step 1

Please install the VIPA CP1413plus H1 network adapter as described in chapter 3.2.3.1.

Step 2

Set the required parameters, if necessary.

Step 3

When Windows for Workgroups (WfW) is installed for the first time, the network configuration function will be executed automatically. Should you have an existing WfW installation you must please start the program "Network Setup" in the "Network" group of WfW. The chronological sequence of operations required to configure the network in Windows for Workgroups are depicted below.

Please select the following items in "Network-Setup":

```
->Network
    -> install the Microsoft Windows Network
    -> No additional network
->Network
    -> install the Microsoft Windows Network
-> More
-> Driver
    -> Network driver
    -> Add 3COM Etherlink III-Adapter
        -> Settings (automatic or not used)
            Interrupt
            Base I/O-interface (please enter)
            Driver type
            Real-Mode NDIS Driver
        -> IPX/SPX Compatible Transport with NetBIOS -> remove
-> Microsoft NetBEUI (save settings)
-> OK terminates setup.
```

Step 4

Please restart your PC. WfW should start and run correctly. Once WfW is running the network services are available. You can use the file manager to test the network services.

Step 5

Please copy the H1 directory for the Windows Multiprotocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_C\*.* C:\H1_DOS\
```

Step 6

Add the lines contained in the file C:\H1_DOS\CONFIG.SYS to the C:\CONFIG.SYS file

```
LASTDRIVE=p
device=c:\windows\protman.dos /I:c:\windows
device=c:\windows\ifshlp.sys
device=c:\h1_dos\elnk3.dos
device=c:\windows\ndishlp.sys
device=c:\h1_dos\msipx.sys
device=c:\h1_dos\hlprot.dos
```

If the LASTDRIVE parameter exists, change it to read LASTDRIVE=P.

Step 7

Check that the C:\AUTOEXEC.BAT file contains the following lines:

```
C:\WINDOWS\net start
C:\H1_DOS\MSIPX
.....\NETX
```

If any one of these lines are not present, add them to the beginning of the file as shown in the example above.

You can find the NETX program on your Novell installation disks.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the file C:\WINDOWS\PROTOCOL.INI.

```
[H1Prot_nif]
DriverName = H1Prot$
Bindings = MS$ELNK3
MaxSendeBuffer = 10
MaxVerbindungen = 10
MaxAckBuffer = 10
MsDosVektor = 0x7d
;NetAddress = I0020D582FFFF
```

The leading "I" for the network address is mandatory!

```
[msipx_nif]
DriverName = IPX$
MEDIATYPE = NOVELL/ETHERNET
BINDINGS = MS$ELNK3
```

The file C:\H1_DOS\PROTOCOL.WIN contains an example of a valid installation and may be used as is.

```
copy c:\h1_dos\protocol.win c:\windows\protocol.ini
```

Subsequently you must please change the NETADDRESS setting in the section [H1PROT_NIF] of the file C:\WINDOWS\PROTOCOL.INI.

Please ensure that the assigned network address is unique.

Step 9

Restart the PC and make sure that **all programs are loaded without error messages**.

Also check for errors during the start up of Windows for Workgroups.

Note This installation requires that WINDOWS is installed on the C: drive in the directory C:\WINDOWS. If not, all references to WINDOWS must be changed in the configuration section.

The configuration was generated with MS-DOS 6.22, WfW 3.11.060.

The directory A:\H1_DOS\VAR_C\DOS622 contains an example of the AUTOEXEC.BAT, CONFIG.SYS, PROTOCOL.INI, WIN.INI and SYSTEM.INI files from a valid installation.

3.2.4.1.4 MS-DOS Version D with IBM LAN Requester 4.0

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 under MS-DOS
- IBM LAN Requester 4.0

Step 1

Please install the VIPA CP1413plus H1 network adapter as described in chapter 3.2.3.1.

Step 2

Set the required parameters, if necessary.

Step 3

Install the IBM LAN Requester according to the instructions supplied by IBM. Install the "3Com Etherlink® III" network adapter to provide the ethernet interface.

Step 4

Please restart your PC. The network should start and run correctly.

To test the operation of the network you can send a message to another network client as follows:

```
net send [name of recipient] ["message"]
```

Step 5

Please copy the H1 directory for the Windows Multiprotocol onto your hard disk.

```
xcopy A:\H1_DOS\VAR_D\*.* C:\H1_DOS\
```

Step 6

Add the lines contained in the file C:\H1_DOS\CONFIG.SYS to the C:\CONFIG.SYS file

```
LASTDRIVE=p  
device=c:\net\protman.dos /I:c:\net  
device=c:\net\ndishlp.sys  
device=c:\h1_dos\elnk3.dos  
device=c:\h1_dos\h1prot.dos
```

If the LASTDRIVE parameter exists, change it to read LASTDRIVE=P.

Step 7

Check that the C:\AUTOEXEC.BAT file contains the following line:

```
C:\net\net start
```

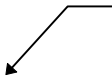
If this line is not present, add it to the beginning of the file.

Step 8

Append the contents of the file C:\H1_DOS\PROTOCOL.INI to the file C:\WINDOWS\PROTOCOL.INI.

```
[H1Prot_nif]
DriverName = H1Prot$
Bindings = IBMS$ELNK3
MaxSendeBuffer = 10
MaxVerbindungen = 10
MaxAckBuffer = 10
MsDosVektor = 0x7d
;NetAddress = I0020D582FFFF
```

The leading "I" for the network address is mandatory!



The file C:\H1_DOS\PROTOCOL.NET contains an example of a valid installation and may be used as is.

```
copy c:\h1_dos\protocol.net c:\net\protocol.ini
```

Subsequently you must please change the NETADDRESS setting in the section [H1PROT_NIF] of the file C:\WINDOWS\PROTOCOL.INI if you wish to change your own network address.

Please ensure that the assigned network address is unique.

Step 9

Restart the PC and make sure that **all programs are loaded without error messages.**

Note

This installation requires that the IBM LAN Requester is installed on the C: drive in the directory C:\NET. If not, all references to NET must be changed in the configuration section.

The configuration was generated with MS-DOS 6.22, IBM LAN Requester 4.0.

The directory A:\H1_DOS\VAR_D\DOS622 contains an example of the AUTOEXEC.BAT, CONFIG.SYS, PROTOCOL.INI, WIN.INI and SYSTEM.INI files from a valid installation.

3.2.4.1.5 MS-DOS Version E with IBM LAN Requester 4.0 and Novell IPX

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 under MS-DOS
- IBM LAN Requester 4.0
- Novell IPX

Install the IBM LAN Requester and Novell NetWare according to the manufacturers instructions.

Install the "3Com Etherlink[®] III" network adapter to provide the ethernet interface.

Now you must complete the installation according to the steps 4 to 9 for the MS-DOS Version C of chapter 3.2.4.1.3. Use the directory of the IBM LAN Requester instead of the Windows directory.

3.2.4.1.6 MS-DOS Version F with 2 network adapters, no additional protocols

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 with 2 network adapters under MS-DOS, without additional protocols

Step 1

Install the two network adapters in your PC (see chapter 3.2.3.1)

Step 2

Set the required parameters, if necessary.

Step 3

Start the configuration program 3c5x9cfg.exe from the disk SW82C disk 1/1.

The adapters must have different I/O base addresses and interrupt request levels.

Example:

Adapter 1: I/O-base address	0x340
Interrupt Request	10

Adapter 2: I/O-base address	0x300
Interrupt Request	15

Step 4

Copy the files from the directory \H1_DOS\VAR_F located on the disk SW83C 1/2 to your hard disk C: into the directory \H1_DOS:

```
xcopy a:\h1_dos\var_f\*.* c:\h1_dos\
```

Step 5

Add the following lines to your C:\CONFIG.SYS and C:\AUTOEXEC.BAT files:

additional lines in C:\AUTOEXEC.BAT

```
:
C:\H1_DOS\NETBIND
:
```

additional lines in C:\CONFIG.SYS

```
:
DEVICE=C:\H1_DOS\PROTMAN.DOS /I:C:\H1_DOS
DEVICE=C:\H1_DOS\ELNK3.DOS
DEVICE=C:\H1_DOS\ELNK3.DOS
DEVICE=C:\H1_DOS\H1PROT.DOS
:
```

Step 6

Next you must please edit the file C:\H1_DOS\PROTOCOL.INI

1. Change the hardware settings in sections [ELNK3_NIF] and [ELNK32_NIF]

Please ensure that the IOADDRESS is not being used by another adapter.

2. Change the NETADDRESS entry in the section [H1PROT_NIF]

Please ensure that the network address is unique.

Excerpt from C:\H1_DOS\PROTOCOL.INI

```
[ PROT_MAN ]
```

```
DRIVERNAME = PROTMAN$
```

```
[ IBMLXCFG ]
```

```
ELNK3_NIF = ELNK3.NIF
```

```
ELNK32_NIF = ELNK32.NIF
```

```
H1PROT_NIF = H1PROT.NIF
```

```
[ H1PROT_NIF ]
```

```
DRIVERNAME = H1PROT$
```

```
BINDINGS = ELNK3_NIF,ELNK32_NIF
```

```
; NETADDRESS = I0020D582FFFF,I0020D582FFFE
```

```
MAXVERBINDUNGEN = 10,10
```

```
MAXSENDEBUFFER = 10,10
```

```
MAXACKBUFFER = 10,10
```

```
MSDOSVEKTOR = 0X7D,0X7D
```

The leading "I" for the network address is mandatory!

```
[ ELNK3_NIF ]
```

```
DRIVERNAME = ELNK3$
```

```
IOADDRESS = 0X340
```

```
[ ELNK32_NIF ]
```

```
DRIVERNAME = ELNK32$
```

```
IOADDRESS = 0X300
```

Step 7

Restart the PC and make sure that **all programs are loaded without error messages.**

3.2.4.2 OS/2 installation

3.2.4.2.1 OS/2 Version A without additional protocols

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 under OS/2 without additional protocols

Step 1

Please install the VIPA CP1413plus H1 network adapter into your PC (see chapter 3.2.3.1).

Step 2

Restart the PC into DOS and set the required parameters, if necessary.

Boot OS/2 as the following steps refer to OS/2.

Step 3

Copy the files from the subdirectories \H1_OS2\MACS, H1_OS2\PROTOCOL, H1_OS2\TOOLS and H1_OS2\VAR_A located on the disk SW83C 1/2 into the subdirectory \H1_OS2 on your hard disk C::

```
xcopy a:\h1_os2\MACS c:\h1_os2\  
xcopy a:\h1_os2\PROTOCOL c:\h1_os2\  
xcopy a:\h1_os2\TOOLS c:\h1_os2\  
xcopy a:\h1_os2\VAR_A c:\h1_os2\
```

Step 4

Add the following lines to the CONFIG.SYS file:

additional lines in C:\CONFIG.SYS

```
:  
:  
DEVICE=C:\H1_OS2\PROTMAN.OS2 /I:C:\H1_OS2  
DEVICE=C:\H1_OS2\ELNK3.OS2  
DEVICE=C:\H1_OS2\H1PROT.OS2  
RUN=C:\H1_OS2\NETBIND.EXE  
:  
:
```

Step 5

Change the NETADDRESS setting in section [H1PROT_NIF] of the file C:\H1_OS2\PROTOCOL.INI.

Please ensure that the network address is unique.

Excerpt from the file PROTOCOL.INI

```
[PROT_MAN]

DRIVERNAME = PROTMAN$

[IBMLXCFG]

ELNK3_NIF = ELNK3.NIF
H1PROT_NIF = H1PROT.NIF

[H1PROT_NIF]

DRIVERNAME = H1PROT$
BINDINGS = ELNK3_NIF
;Example for a station address (own station address)
; NETADDRESS = I00001C011010
MAXVERBINDUNGEN = 32
MAXSENDEBUFFER = 32
MAXACKBUFFER = 64

[ELNK3_NIF]

DRIVERNAME = ELNK3$
IOADDRESS = 0X340
```

— The leading "I" for the network address is mandatory!

Step 6

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages.**

3.2.4.2.2 OS/2 Version B with IBM LAN Requester 4.0

Refer to chapter 3.2.3 for hardware installation instructions.

This installation includes the following components:

- H1 under OS/2
- IBM LAN Requester 4.0

Note you must have the IBM LAN Requester!

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Restart the PC into DOS and Set the required parameters, if necessary.

Boot OS/2 as the following steps refer to OS/2.

Step 3

Install the IBM LAN Requester 4.0 according to the following procedure:

```

:
In the window "Network Adapters"
-> Other adapters (select)
    Insert the disk SW83C 1/2
    -> A:\H1_OS2\MACS (enter)
        the files will be copied
In the window "Network Adapters"
    VIPA CP 1413plus (3Com Etherlink III) Adapter (select)
    -> Add
In the window "Current Configuration"
-> Edit
    enter the required hardware settings
In the window "Protocols"
-> other Protocols (select)
    -> A:\H1_OS2\PROTOCOL (enter)
    -> OK
        (the driver is copied to your hard disk)
        VIPA H1 Protokoll (select)
        -> Add
        select any other protocol that
        must also be installed.
    -> OK
```

Step 4

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages**.

Note The directory A:\H1_OS2\VAR_B contains the CONFIG.SYS and PROTOCOL.INI files of a valid installation. You may refer to them if you require assistance.

3.2.4.2.3 OS/2 Version C with IBM LAN Requester 4.0 and Novell IPX

This installation includes the following components:

- H1 under OS/2
- IBM LAN Requester 4.0
- Novell IPX

Note you must have the IBM LAN Requester!

Step 1

Install the VIPA CP1413plus H1 network adapter as described in chapter 3.1.3.

Step 2

Restart the PC into DOS and Set the required parameters, if necessary.

Boot OS/2 as the following steps refer to OS/2.

Step 3

Install the IBM LAN Requester 4.0 if it has not already been installed. Select the "IBM NetWare Requester Support" setting for "Protocols".

Step 4

Once you have restarted the PC and verified that the network is operating properly, execute the "install" program located in the directory "ibmcom".

Use the following procedure:

```
-> Configure
-> select LAN adapters and protocols
   ->configure
       for "Protocols" select
       -> other Protocols
           -> A:\H1_OS2\PROTOCOL                      (enter)
               ->OK
                   (the driver will be copied to the hard disk)
       select VIPA H1 Protokoll
       -> Add
           -> OK
-> close
```

You must also install the IPX drivers and bind them via NDIS.

Step 5

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages.**

Note The directory A:\H1_OS2\VAR_C contains the CONFIG.SYS and PROTOCOL.INI files of a valid installation. You may want to refer to them if you require assistance.

3.2.4.3 Windows NT installation

Installing H1prot.sys for WindowsNT

Step 1

Refer to chapter 3.2.3 for hardware installation instructions.

Note you may need to modify the configured settings. We recommend that you use an address of 340h if this has not been allocated.

Step 2

Install the driver for the network adapter as follows:

- Start "Control Panel" in the "Main" group.
- select "Network" in "Control Panel".
- select "Add Network Adapter" in "Network Setup".
- select the "3COM Ethernetlink" from the list.
- Enter the correct "Address" and "Interrupt".

Step 3

Install the H1_Prot driver as follows:

1. Copy the driver "**h1prot.sys**" from the directory \H1_NT of the disk SW83C 1/2 to your hard disk C: into the installation directory of Windows NT \WINNT35 :

```
copy a:\H1_NT\H1prot.sys  
c:\winnt35\system32\drivers
```

2. Enter the required settings into the "Registry".

These settings are available from the file "H13COM35.ini" located on the VIPA installation disk 1/2 in the directory "\H1_NT".

Use the "Regini.exe" tool to make the entries:

```
a:\H1_NT\regini a:\H1_NT\H13com35.ini
```

Step 4

Execute a system shutdown.

Restart the PC and make sure that **all programs are loaded without error messages.**

Installation for experienced users

A script file exists to install the H1 driver for Windows NT. This script file is processed when the H1 driver is installed for the first time.

Proceed as follows:

Step 1

Install the MAC driver for the network adapter

Step 2

Insert the H1 driver disk into drive A: and select "Other Protocol" from the options. Another panel will be displayed.

Step 3

Select "A:\H1_NT". The disk will be read and the panel will display "VIPA H1-Protokoll". Select this protocol.

Step 4

Where a number of different network adapters have been installed, the binding of the H1 protocol driver to non-VIPA CP1413plus-adapters must be removed. Otherwise the H1 driver will not operate properly.

3.2.4.4 Windows 95 installation

Step 1

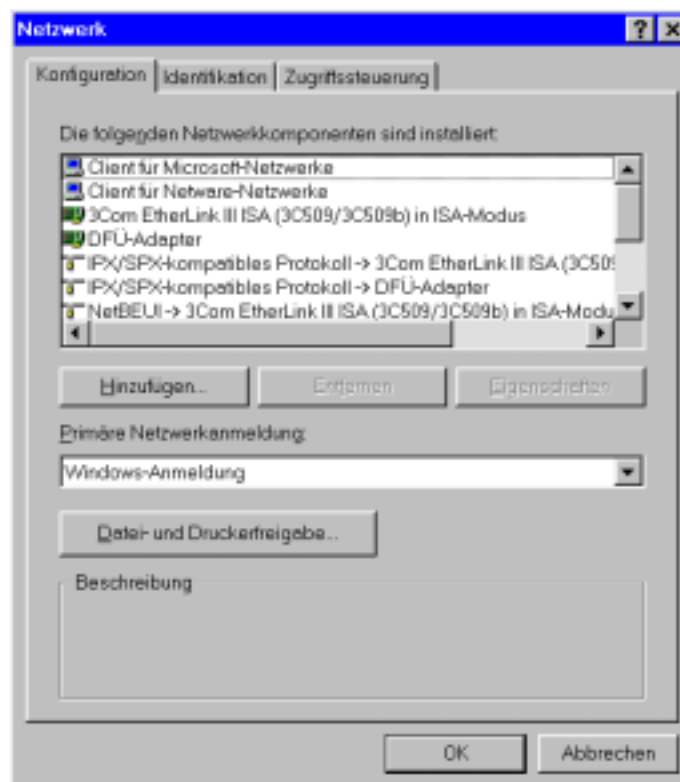
Refer to chapter 3.2.3 for hardware installation instructions.

Step 2

Start Windows. Go to “Settings” on the Start Menu. Select the “Control Panel”. The *Control Panel* window will be displayed.

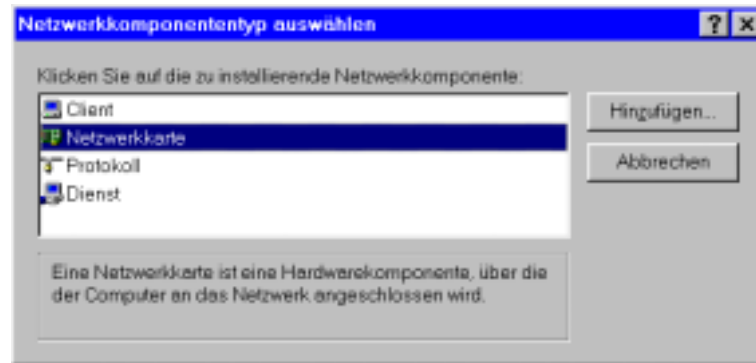


Execute the “Network” program from this window. The *Network* window will be displayed.



Step 3

Click on **[Add]**. The window *Select Network Component Type* will be displayed. You may enter the VIPA network adapter and the VIPA-H1 protocol driver. First you must specify what network adapter you will use. Select "Adapter" and click on **[Add]**.

**Step 4**

Select "3Com" as the manufacturer. A list containing all available adapters made by 3Com will be displayed. Select the following adapter from the list and click **[OK]**:

"3Com EtherLink III ISA (3C509/3C509b) in ISA-Mode"



This concludes the installation for the adapter.

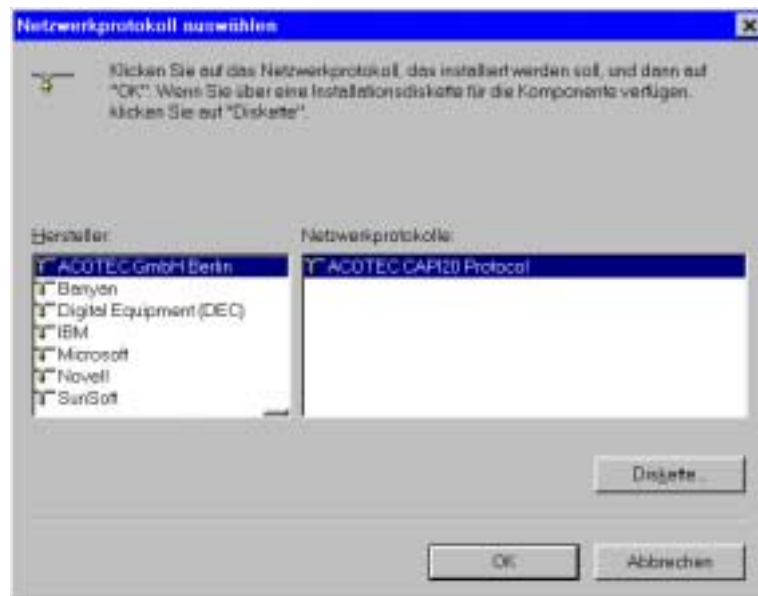
Step 5

You need to install the H1 protocol to work with the H1 system. Select "3Com" from the *Network* window and click on **[Add]**.

The window *Select Network Component Type* will be displayed next. Enter the H1 protocol by selecting "Protocol" and clicking on **[Add]**. The *Network Protocol* window will appear.

Step 6

Insert the disk containing the H1 driver and click on **[Have Disk...]**.



Click on **[Browse...]** in the *Install From Disk* window.

In the window *Open* select the file "H1PROT.INF" from the directory A:\H1_WIN95 and click on **[OK]**.

The window *Copy From Disk* now contains the entry "A:\H1_WIN95". Click on **[OK]**. The selected protocol will be installed.

Select the "VIPA H1 protocol" in the window *Select Network Protocol* and click on **[OK]**.

Click **[OK]** in the *Network* window. This concludes the installation of the adapter and the H1 protocol.



If you have installed network adapters that are not VIPA CP1413plus adapters, you must remove the binding between the H1-protocol driver and all non-VIPA adapters. Otherwise the H1-driver will not operate properly.

3.3 Entry into protocol file [H1PROT_NIF]

Once the installation into MS-DOS, Windows 3.x and Windows 95 is complete, the section [H1PROT_NIF] is inserted into the Protokoll.ini file.

The following are optional parameters. Default settings will be used wherever you decide not to enter a value.

MAXVERBINDUNGEN	<p>specifies the maximum number of connections that may be open simultaneously.</p> <p>Range 2 <= x <= 128</p> <p>Note every connection requires 100 bytes.</p> <p>Default DOS 10; OS/2 30; NT 64</p>
MAXSENDEBUFFER	<p>are the internal storage areas for data during SEND and RECEIVE. These buffers are occupied when:</p> <ol style="list-style-type: none">1. SEND was started but the receiving station has not accepted any data.2. RECEIVE "active". <p>The size of the send buffer is 1100 byte. The system will slow down if the pool of send buffers has been exhausted. As it is normal for all connections to "hang" the number of SENDEBUFFERS may be less than the number of connections.</p> <p>Range 2 <= x <= 128</p> <p>Default DOS 10; OS/2 25; NT 64</p>
MAXACKBUFFER	<p>are used for internal H1 handshakes. The number of MAXACKBUFFERS should be exactly the same as the number of connections if this connection is often used for data transfers. The number may be less if the system load is low.</p> <p>Range 2 <= x <= 128</p> <p>Default DOS 10; OS/2 30; NT 64</p>
MSDOSVEKTOR	<p>ONLY FOR MS-DOS AND WINDOWS</p> <p>Access to this driver is subject to a software interrupt in MS-DOS. As the use of vectors by programs is not subject to any rules, the vector must be specified. This may not be in use by another program.</p> <p>Range 0x78 <= x <= 0x7F</p> <p>Default is 0x7A</p>

Note If this vector is changed you must use the function `H1SetzeVektor (int vector)` in those programs that will use the H1.

Certain versions of Novell use the vector 0x7A.

NETADDRESS

Contains the local station address.

The station address of some network adapters can not be changed after the system has been started.. In this case the station address must be specified here.

Default ROM adapter address with a leading "I". The "I" is mandatory.

PROTOKOLLIEREN

specifies whether the driver is placed in Promiscuous Mode or not. It is necessary to change to Promiscuous Mode if the network must operate with "Station Restrictions".

Range Ja / Nein (Yes/No)

Default Nein (No)

4 Programming

4.1 General information on programming	4-1
4.1.1 Summary, operation - procedure	4-3
4.1.2 Description of the H1-parameters	4-6
4.1.3 Error messages from .Fehler	4-8
4.1.4 Determining the ethernet address	4-10
4.2 H1 Layer 4 program interface	4-11
4.2.1 General information on the H1 program interface	4-11
4.2.2 General H1 Layer 4 functions	4-13
4.2.3 Specific H1 Layer 4 functions	4-29
4.3 PLC Layer 7 program interface	4-41
4.3.1 General information on the PLC program interface	4-41
4.3.2 General Layer 7 functions	4-42
4.3.3 File functions Layer 7	4-47
4.3.4 Specific Layer 7 functions	4-61

4 Programming

4.1 General information on programming

Layer 4

Layer		
7	APPLICATION	H1DriverOpen H1DriverClose H1GetVersion
6	PRESENTATION	H1StartConnect H1StartConnectCard
5	SESSION	H1StopConnect H1StopConnectAll
4	TRANSPORT	H1TestStatus H1GetLineCharacteristics H1ListDefinedConnections
3	NETWORK	H1GetStationAddress H1GetStationAddressCard H1SetStationAddress H1SetStationAddressCard
2	DATA LINK	H1GetStandardvalues H1SetStandardvalues H1SetVector
1	PHYSICAL	H1SendData H1SendDataEx H1StartSend H1CheckSend H1ReadData H1ReadDataEx H1StartRead H1StartReadEx H1CheckRead H1CheckReadEx

The Layer 4 functions correspond to the "SEND DIRECT" and "RECEIVE DIRECT" functions of the VIPA CP143 plus PLC-interface.

Layer 7 (Application)

Layer

7	APPLICATION
6	PRESENTATION
5	SESSION
4	TRANSPORT
3	NETWORK
2	DATA LINK
1	PHYSICAL

S5StartConnection
 S5StartConnectionCard
 S5StopConnection
 S5StopAllConnections
 S5GetRevision
 S5SetNetfileName
 S5GetNetfileName
 S5GetConnectionparameter
 S5GetConnectionCard
 S5PutConnectionparameter
 S5WriteConnectionCard
 S5ListConnections
 S5ListNetConnections
 S5SetVector
 S5SetStationaddress
 S5SetStationaddressCard
 H1ReadParameter
 H1WriteParameter
 S5GetParameter
 S5PutParameter
 S5ReadFromPLC
 S5StartRead
 S5PollRead
 S5WriteToPLC
 S5StartWrite
 S5PollWrite
 S5FetchPassiv
 S5WritePassiv

The AP-functions correspond to the "FETCH ACTIVE" and "WRITE ACTIVE" functions of the VIPA CP143 plus PLC-interface.

4.1.1 Summary, operation - procedure

This summary contains a listing of all the functions along with the respective configuration procedure.

Defining a connection

PC-side	CP-function	PLC-side
<u>in the file "net.net"</u> <i>[Connection_1]</i> 1. <i>EthernetStation = 0020d582000f</i> (Ethernet address remote station) 2. <i>EthernetLocalTsap = 11223344</i> (2 to 16 bytes, TSAP's of both stations overlap) 3. <i>EthernetRemoteTsap = 22334455</i> (2 to 16 bytes) 4. <i>EthernetPriority = Prio2</i> 5. <i>EthernetLineType = Normal,Active</i> <i>e</i> ("Normal,Passive" is also possible depending on the type of connection) 6. optional: <i>EthernetAdapter = 1</i> (Adapter number 1 to n)	none	FB SendAll FB ReceiveAll

Tab. 4-1: Procedure define connection

Layer 4: Read data from PLC

PC-side	CP-functions	PLC-side
<u>RECEIVE</u> H1ReadData () H1StartRead () H1CheckRead () <u>in the file „net.net“</u> <i>[Connection_1]</i> <i>EthernetLineType = Normal,Passive</i>	Send, Active (default), Read Write = No (default)	FB Send FB Control FB SendAll (absolute)

Tab. 4-2: Procedure Read data from PLC

Layer 4: Write data to PLC

PC-side	CP-function	PLC-side
SEND H1SendData () H1StartSend () H1CheckSend () <u>in the file „net.net“</u> <i>[Connection_2]</i> <i>EthernetLineType = Normal,Active</i>	Receive Read Write = No (default)	FB Receive FB Control FB ReceiveAll (absol.)

Tab. 4-3: Procedure Write data to PLC

Layer 4 Dual connection: write data to PLC or read data from PLC via TSAP

PC-side	CP-function	PLC-side
You must specify parameters for 2 tasks: <u>1. SEND</u> H1SendData () H1StartSend () H1CheckSend () <u>in the file „net.net“</u> <i>[Connection_3]</i> <i>EthernetLineType = Normal,Active or Normal,Passive</i> in the sequence used when configuring the PLC <u>2. RECEIVE</u> H1ReadData () H1StartRead () H1CheckRead () <u>in the file „net.net“</u> <i>[Connection_4]</i> <i>EthernetLineType = ... (same as SEND)</i>	Send, Active (default), Read Write = No (default) Receive Read Write = No (default) Send - Rec: Ethernet Active PC: Normal,Passive Rec - Send Ethernet Passive PC: Normal,Active Please observe sequence!	FB Send FB Control FB SendAll (absol.) FB Receive FB Control FB ReceiveAll (absol.)

Tab. 4-4: Procedure for a dual connection

Layer 7: Read a DB from the PLC

PC-side	CP-function	PLC-side
<u>FETCH</u> S5ReadFromPLC() S5StartRead() S5CheckRead() <u>in the file „net.net“</u> <i>[Connection_5]</i> <i>EthernetLineType = Normal,Active</i>	Fetch Passive, Read Write = Yes	FB SendAll (absolute) FB ReceiveAll (absol.)

Tab. 4-5: Procedure read DB from PLC

Layer 7: write DB to PLC

PC-side	CP-function	PLC-side
<u>WRITE</u> S5WriteToPLC() S5StartWrite() S5PollWrite() <u>in the file „net.net“</u> <i>[Connection_6]</i> <i>EthernetLineType = Normal,Active</i>	Receive Passive, Read Write = Yes	FB SendAll (absolute) FB ReceiveAll (absol.)

Tab. 4-6: Procedure write DB to PLC

4.1.2 Description of the H1-parameters

Excerpt from H1Def.h

```
typedef struct _CONNECT_PARAMS
{
    unsigned char Priority;           /* priority */
    unsigned char ConnectType;       /* type of connection */
    unsigned char LenDestAddr;       /* Length of destination address */
    unsigned char DestAddr[6];       /* destination address */
    unsigned char Multicast;         /* multicast circuit number */
    unsigned char LenNSAP;           /* Long destination-NSAP_ID */
    unsigned char NSAP[12];          /* destination NSAP-ID */
    unsigned char LenDestTSAP;       /* Long destination TSAP_ID */
    unsigned char DestTSAP[16];      /* destination TSAP-ID */
    unsigned char LenOwnTSAP;        /* Long own TSAP_ID */
    unsigned char OwnTSAP[16];       /* own TSAP-ID */
    unsigned char LenConnParams;     /* length of additional connect params */
    unsigned char ConnParams[16];    /* additional connect params */
} CONNECT_PARAMS;
#endif // _CONN_PARAMS_DEFINED
#endif // NORMAL_LINE
```

H1-parameter

Priority

The line priority can have any value between 0 (highest priority) to 4 (lowest priority).

0 and 1 are so-called express priorities, **2 and 3** are normal priorities. **Prio 4** will not be used very often as it causes the connection to be reestablished with every send operation. On the other hand, this priority will place far less of a load on the network when the connection is not used very often, as the line does not have to be supervised (as the connection is terminated after each send operation). Please note that the data transfer rate for express priorities is no higher than that for normal priorities. However, some controllers use interrupts to transfer data to memory when **priority 0** is selected. As a result the overall data transfer rate may be higher. The maximum data for **priorities 0 and 1** may be 16 bytes.

Connection type

Active / Passive determines whether the local station actively establishes a connection or whether it expects the remote station to establish the connection. This value may not be the same on the two sides of a connection.

Destination address

Ethernet Addressing parameters have binary values which may range from hex 0 to hex FF. The remote station, which may consist of a PLC is identified by means of the address. This type of ethernet address always has 6 characters. The first three characters determine the manufacturer of the destination system. These bytes are provided by the central IEEE committee. Where the manufacturers code has not been assigned it is important to ensure that the value of the first byte is even, i.e. it must be divisible by two. The remaining three bytes can have any value. It is not possible to have multiple stations with the same ethernet address in one network.

Multicast circuit-No.

Multicast connections are connections that are not directed at all clients on the network but only at those with the same multicast circuit number. This number may range from 0 to 63. If the line type is not multicast, then the value for multicast circuit is irrelevant.

NSAP

The NSAP (Network Service Access Point) determines the network number for linked networks. This value is hardly ever used as many types of network have no provisions for the NSAP.

Remote TSAP

The remote TSAP determines the connection address of the other system. Before a connection can be established, the own TSAP must correspond to the remote TSAP. This condition can easily be satisfied if the own TSAP and the remote TSAP are set to the same value.

The length of such a TSAP is often 8 bytes, however, it may range from 1 to 16 bytes. Industrial Ethernet systems use 20 as a value for the first 3 bytes.

Own TSAP

The value for own TSAP (Transport Service Access Point) determines the connection address for the local system that is used for data transfer. The TSAP's on the H1-level are comparable with ports on the TCP/IP-level.

4.1.3 Error messages from .Fehler

No.	Message	Description
0	OK	no error, command will be executed
1	H1_BAD_CR_PARAMS	illegal CR parameter
2	H1_NO_SLOT	maximum number of connections are active
3	H1_WAIT_CONNECT	connection has been severed or interrupted
4	H1_NOT_IMPLEMENTED	function is not implemented
5	H1_BAD_LINE	illegal connection number
6	H1_WAIT_DATA	no data available
7	H1_WAIT_SEND	data has not been sent
8	H1_INTERNAL_ERROR	internal error
9	H1_NO_REQUEST	polled task does not exist
10	H1_NO_DRIVER	no driver
11	H1_UEBERLAST	task could not be executed
12	H1_BLOCKED_DATA	arrival of blocked data
13	H1_NO_ADAPTER	the specified adapter does not exist
14	H1_ALREADY_RUNNING	task is already running

Tab. 4-7: Error messages from .Fehler

Description

H1_BAD_CR_PARAMS	The connection parameters are invalid or incorrect..
H1_NO_SLOT	The maximum number of connections has already been started. Please terminate existing connections before you start another connection.
H1_WAIT_CONNECT	This is not a true error message. The message indicates that the connection has been initiated but not yet established.
H1_NOT_IMPLEMENTED	This message is generated when you call a function that has not yet been implemented on your system..
H1_BAD_LINE	This message may be due to two reasons: - one of the connection numbers is incorrect. - the driver was never opened.
H1_WAIT_DATA	Data has not yet been received. The timer has expired. The timer of the driver has been set to 60 minutes. You may determine the current condition of a connection by means of H1CheckRead .
H1_WAIT_SEND	Data has not yet been sent. You may determine the current state of transmission by means of H1CheckSend .
H1_INTERNAL_ERROR	An error has occurred in the driver. Please contact the VIPA-Hotline.
H1_NO_REQUEST	Currently no active send or read task.
H1_NO_DRIVER	No driver exists or the driver has not been defined correctly.

H1_UEBERLAST	<p>The maximum number of available buffers has been exceeded. Please wait until another connection releases its buffer.</p> <p>The error message H1_UEBERLAST (overload) is generated when MaxSendebuffer or. MaxAckbuffer has not been set to 2 x MaxVerbindungen (max. connections).</p>
H1_BLOCKED_DATA	<p>Indicates the arrival of blocked data. A 0 is issued as an acknowledgement once blocking has been terminated. Please use blocking as soon as you want to transfer more than 1024 bytes. Set bit 15 in the task number to initiate blocking.</p>
H1_NO_ADAPTER	<p>This may be caused by one of the following:</p> <ul style="list-style-type: none">- adapter does not exist or is faulty- driver does not exist or incorrectly configured <p>check the adapter number (adapter 1 =0) did the system recognize the adapter?</p> <p>Netbind executes without errors in MS-DOS and in Windows 3.11. Windows 95 has not detected any hardware conflicts. The "Event Viewer" of Windows NT may not have any entries. No error messages may occur during the boot phase of OS/2.</p>
H1_ALREADY_RUNNING	<p>the task is already running.</p>

4.1.4 Determining the ethernet address

The ethernet address (6 bytes) has the following components:

- byte 1, 2, 3: 0x00 0x20 0xD5 (VIPA network designator)
- byte 4, 5, 6: these 3 bytes are located on the CP1413plus.

Z'nyx-adapter

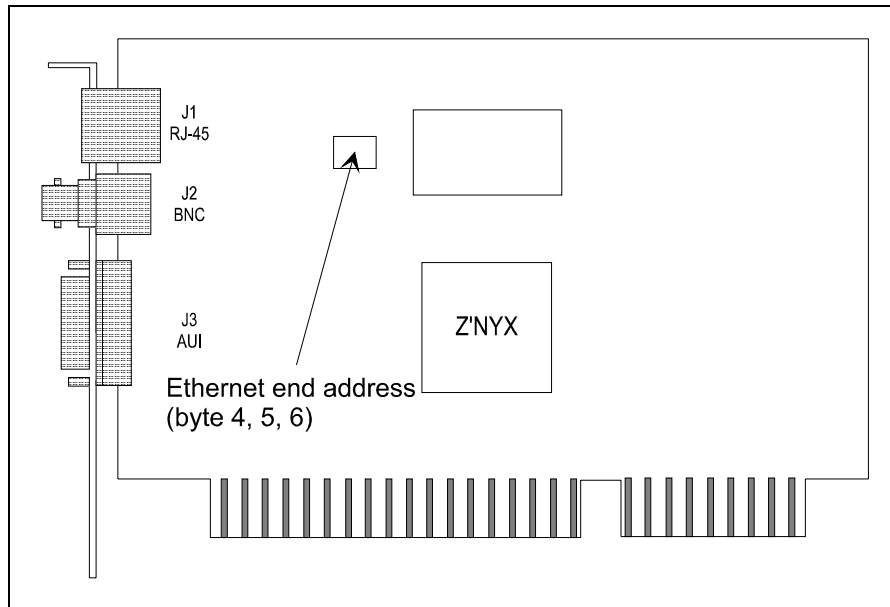


Fig. 4-1: The ethernet end-address of the Z'nyx-adapter (byte 4, 5, 6)

3Com-adapter

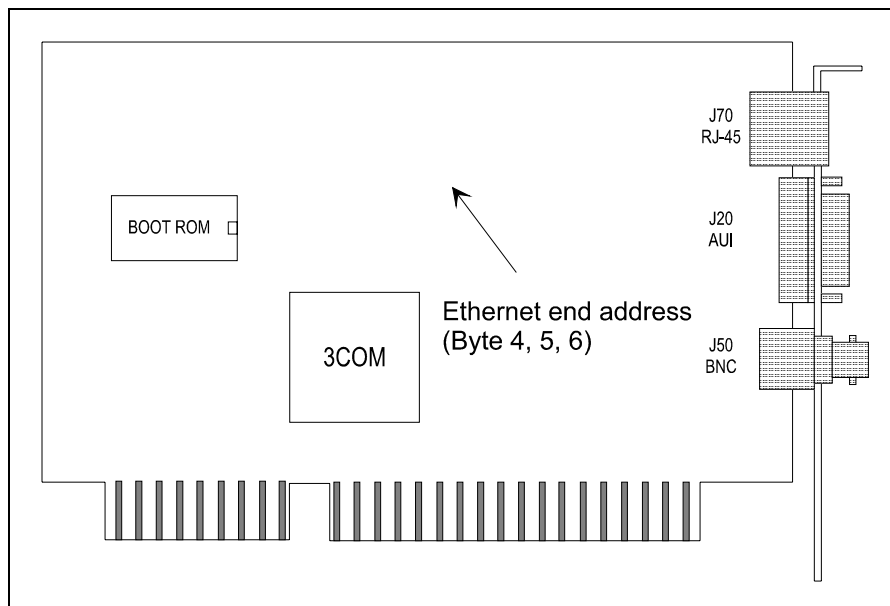


Fig. 4-1: The ethernet end-address of the 3Com-adapter (byte 4, 5, 6)

4.2 H1 Layer 4 program interface

4.2.1 General information on the H1 program interface

The H1 program interface provides direct access to the functions of level 4 A of the ISO 7-layer model.

Data is transferred in its present form, i.e. no further conversion takes place.

To access any station it is necessary to first establish a connection to the destination system. This connection may then be used to transfer the data to and from the destination system.

It is possible to establish multiple connections to different systems or to a single system.

Any connection that is no longer required should be terminated.

MS-DOS

The MS-DOS based access functions are available as “med” and “large” memory model from a library for Microsoft C6.0.

The respective file names are:

- Layer4	H1Zugrif.c	bzw.	xH1LibR.lib	(x = m for "med" Model, l for "large" Model)
- AP	S5AP.c	bzw.	xS5Ap.lib	(x = m for "med" Model, l for "large" Model)
- Netdatei			xS5AP.lib, xwmr.lib	(x = m for "med" Model, l for "large" Model)

The library was tested with MSC 6.0A

OS/2

All functions are supplied in the form of dynamic link libraries (.dll).

- H1LibP.dll, S5AP.dll

The following export libraries are available for linking custom-written programs:

- H1LibP.lib, S5AP.lib

The libraries were tested with:

- IBM Cset++ 2.1, 3.0
- BorlandC/C++ 1.5, 2.1
- WatcomC/C++ 10.0, 10.5

Windows NT

All functions are supplied in the form of dynamic link libraries (.dll).

- H1DIINT.dll, S5APNT.dll

The following export libraries are available for linking custom-written programs:

- H1DIINT.lib, S5APNT.lib

These libraries were tested with VisualC++ 1.5 and 2.0

Windows

All functions are supplied in the form of dynamic link libraries (.dll).

- H1LibWin.dll, S5APWin.dll

The following export libraries are available for linking custom-written programs:

- H1LibWin.lib, S5APWin.lib

These libraries were tested with MSC 6.0A and VisualC++ 1.5

The connection parameters determine the type of connection. These parameters are provided in a central connection file; however, the user they may also supply individual values where required. The recommended method is to use the connection file as this provides for better documentation.

Connection files may be modified by means of an editor. You may use any ASCII editor for DOS and in OS/2, WINDOWS and Windows NT any one of the common windows based text editors will suffice. Here you may also use an ASCII editor as the connection files contain plain ASCII data.

A description of the file format is contained in chapter 5.

4.2.2 General H1 Layer 4 functions

The following conditions apply to all functions of the layer 4 interface:

- All functions for the layer 4 interface have an H1 prefix.
- The return value of the function is the return value as defined for the respective operating system for the `DeviceIOctl` function. Please refer to the documentation of the respective operating system for a list of possible return values.
- On a PC user programs may access layer 4 tasks directly. The file `H1DEF.H` contains a listing of all the functions.

4.2.2.1 Driver open

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1DriverOpen(void)
```

The `H1DriverOpen` function is used to initialize access to the VIPA H1 driver.

Parameters

None.

Return value

0
`H1_NO_DRIVER`
`H1_NO_ADAPTER`

Note

This function must be the first one that is executed. The driver can be closed by executing the `H1DriverClose` function.

See also

`H1DriverClose`
`H1StopConnectAll`

4.2.2.2 Driver close

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1DriverClose(void)
```

The **H1DriverClose** function releases any allocated resources.
Connections that were assigned to the current task (for OS/2, Win95, WinNT) are terminated.

Parameters

None

Return value

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This function must be the last one that is executed. Once this function has completed the driver can only be accessed again after another **H1DriverOpen**. All connections assigned to the current task are terminated.

In MS-DOS the connections are not terminated automatically. This can be achieved by executing the **H1StopConnectAll** function.

See also

H1DriverOpen
H1StopConnectAll

4.2.2.3 Determining the driver version

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1GetVersion(unsigned short W_POINTER  
version)
```

This function returns the version level of the H1 software.

Parameters

version Pointer to an unsigned short variable that contains the decimal version number of the H1-driver.

Example: a return containing 124 indicates version 1.24

Return value

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

The version number is defined as follows: The first digit indicates the main version number and the second and third digit indicate the sub-version number.

Thus a version number 112 means version 1.12.

The driver must first have been opened by means of **H1DriverOpen**.

See also

H1DriverOpen

H1DriverClose

4.2.2.4 Initiate connection

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StartConnect (H1_CONNECT_PARAMS W_POINTER  
cr)
```

The function **H1StartConnect** is used to initiate connections.

Parameters

cr Pointer to a structure of the type **H1_CONNECT_PARAMS**. Connection parameters must be declared before this function is executed.

Possible values that may be returned by **cr.Fehler**:

```
0  
H1_BAD_CR_PARAMS  
H1_WAIT_CONNECT  
H1_NO_SLOT  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Return value

< > 0 The operating system has detected an error.
0 The **cr.Fehler** element contains one of the above values.

Note

If the function completes without errors a connection number will be returned in the structure. This number is valid until the connection is terminated by means of **H1StopConnect**.

The driver must previously have been opened by means of **H1DriverOpen**.

See also

```
H1DriverClose  
H1DriverOpen  
H1StopConnect  
H1StartConnectCard
```

4.2.2.5 Initiate connections for multiple adapters

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StartConnectCard(
    H1_CONNECT_PARAMS_LINE W_POINTER cr)
```

The **H1StartConnectCard** is used to initiate connections. You must use this function if your hardware contains more than one adapter. You may also use this function when you are only using a single adapter.

Parameters

cr Pointer to a structure of the type **H1_CONNECT_PARAMS_LINE**. The connection number and the adapter number must have been previously assigned.

Possible values returned by **cr.Fehler**:

```
0
H1_WAIT_CONNECT
H1_BAD_CR_PARAMS
H1_NO_SLOT
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

Return value

< > 0 The operating system has detected an error.

0 The **cr.Fehler** element contains one of the values above.

Note

The structure will return a connection number if the function was completed without errors. This number is valid until the connection is terminate by means of **H1StopConnect**.

The driver must have been opened by means of **H1DriverOpen**.

The adapter or card number starts from 0, where 0 represents adapter 1.

See also

```
H1DriverOpen
H1DriverClose
H1StopConnect
H1StartConnect
```

4.2.2.6 Terminate a connection

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StopConnect(unsigned short connection  
number)
```

You can terminate a connection by means of the **H1StopConnect** function.

Parameters

connection number The connection number of a valid connection.

Return value

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

Note

When this function has been executed, the connection number is no longer valid.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been initiated by **H1StartConnect** or **H1StartConnectCard**.

See also

H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard

4.2.2.7 Testing the status of a connection

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1TestStatus(H1_REC_PARAMS W_POINTER rec)
```

The function **H1TestStatus** returns the status of a connection.

Parameters

rec a structure containing line parameters. You must assign a value to the **Vnr** parameter.

Possible values returned by **rec.Fehler**:

```
0  
H1_WAIT_CONNECT  
H1_BAD_LINE  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Return value

< > 0 The operating system has detected an error.
0 The **rp.Fehler** element contains one of the values shown above.

Note

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been initiated by **H1StartConnect** or **H1StartConnectCard**.

See also

```
H1DriverOpen  
H1DriverClose  
H1StartConnect  
H1StartConnectCard
```

4.2.2.8 Get line parameter

```
#include <H1Def.h>
```

```
Get line parameter unsigned short WENTRY_C H1GetLineparameter(H1_LINEVAL  
W_POINTER val)
```

This function returns the currently active parameters for an H1 connection.

Parameters

val a structure containing line parameters. You must assign a value to the **Vnr** parameter.

Possible values returned by **val.Fehler**:

```
0  
H1_WAIT_CONNECT  
H1_BAD_LINE  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Return value

<> 0 The operating system has detected an error.
0 The **val.Fehler** element contains one of the above mentioned values.

Note

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been initiated by **H1StartConnect** or **H1StartConnectCard**.

See also

```
H1DriverOpen  
H1DriverClose  
H1StartConnect  
H1StartConnectCard
```

4.2.2.9 List Connections

```
#include <DrvListe.h>
```

```
unsigned short WENTRY_C H1ListAssignedConnections(All_H1_Anzeige  
W_Pointer array)
```

The function **H1ListAssignedConnections** returns a list containing all connections assigned to the specified driver.

The structure contains the number of valid entries. The list contains the relevant data for each connection:

```
unsigned short Vnr;           \\Connection number
unsigned short SourceReference; \\ Own reference for ethernet
unsigned short DestReference;  \\ Reference for ethernet destin.
unsigned short OwnSequNummer;  \\ Own sequence number
unsigned short DestSequNummer; \\ Destination sequence number
```

DestReference	If DestReference is 0 the connection was not established.
OwnSequNummer	contains the number of telegrams that were sent by the local (own) station since the connection was established..
DestSequNummer	contains the number of telegrams that were sent by the destination station since the connection was established..

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been initiated by **H1StartConnect** or **H1StartConnectCard**.

Note

On systems running OS/2 and Windows NT the connection number may belong to a different task. These connection numbers can not be used.

See also

```
H1DriverOpen  
H1DriverClose  
H1StartConnect  
H1StartConnectCard  
H1TestStatus
```

4.2.2.10 Get station address

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1GetStationAddress(unsigned char W_POINTER address)
```

This function determines the station address of the local (own) station.

Parameters

address Pointer to a memory location containing the current station parameters.

Byte 0,1 = length of station address

Byte 2..7 = current station name

Byte 8..13 = ROM station address

Return value

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

The station parameter consists of two sets of ethernet addresses. These are the permanent address which is located in the ROM of the adapter, and the current address. This address will be used on the ethernet LAN.

See also

H1DriverOpen

H1DriverClose

H1GetStationAddressCard

H1SetStationAddress

H1SetStationAddressCard

4.2.2.11 Get station address for multiple adapters

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1GetStationAddressCard (unsigned char W_POINTER address,  
unsigned short card)
```

This function determines the station address of the local (own) station. This function is required in an environment where more than one adapter is used. You may also use this function when your hardware contains only a single adapter.

Parameters

address Pointer to a memory location containing the current station parameters.

Byte 0,1 = length of station address

Byte 2..7 = current station name

Byte 8..13 = ROM station address

card Number assigned to the adapter. 0 represents adapter 1

Return value

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

The station parameter consists of two sets of ethernet addresses. These are the permanent address which is located in the ROM of the adapter, and the current address. This address will be used on the ethernet LAN.

See also

```
H1DriverOpen  
H1DriverClose  
H1GetStationAddress  
H1SetStationAddress  
H1SetStationAddressCard
```

4.2.2.12 Set station address

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SetStationAddress(unsigned char  
W_POINTER address)
```

This function modifies the local (own) station address.

Parameters

address Pointer to a memory location containing the modified station parameters. The remaining 6 bytes contain the station address.

For an H1 environment the highest byte may not contain a broadcast- or multicast-address.

Return value

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

The station address must be unique, otherwise it is possible that collisions occur on the network.

The station address is independent of other protocols, e.g. IPX, if the setting “Protokollieren = Ja” (log protocol = yes) has been specified. On some LAN’s the **MAC**-driver may not be set to operate in **promiscuous mode**. In this case you must set “Protokollieren = Nein” (log protocol = no). Other systems refuse to cooperate when the station address is modified at run time, i.e. the setting is “Protokollieren = Ja” (log protocol = yes). The system is loaded drastically by the promiscuous mode. Please refer to the **technical reference** for your network for further information in this regard.

Example

```
unsigned char address [6];  
    station[0] = 0x00;  
    station[1] = 0x20;  
    station[2] = 0xd5;  
    station[3] = 0x80;  
    station[4] = 0x02;  
    station[5] = 0x01;  
H1SetStationAddress(address);
```

See also

H1DriverOpen
H1DriverClose
H1GetStationAddress
H1GetStationAddressCard
H1SetStationAddressCard

4.2.2.13 Set station address for multiple adapters

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SetStationAddressCard(unsigned  
    char W_POINTER address, unsigned short card)
```

This function modifies the station address of the local (own) station. This function is required in an environment where more than one adapter is used. You may also use this function when your hardware contains only a single adapter.

Parameters

address	Pointer to a memory location containing the modified station parameters. The remaining 6 bytes contain the station address. For an H1 environment the highest byte may not contain a broadcast- or multicast-address.
card	The number of the adapter. 0 represents adapter 1

Return value

```
0  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Note

See note on **H1SetStationAddress** in chapter 4.2.2.12

Example

```
unsigned char address [6];  
    station[0] = 0x00;  
    station[1] = 0x20;  
    station[2] = 0xd5;  
    station[3] = 0x80;  
    station[4] = 0x02;  
    station[5] = 0x01;  
H1SetStationAddressCard(address, card);
```

See also

```
H1DriverOpen  
H1DriverClose  
H1GetStationAddress  
H1GetStationAddressCard  
H1SetStationAddress
```

4.2.2.14 Get H1 system values

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1GetStandardvalues(  
    H1_Initvalues W_Pointer init)
```

This function returns the H1system values.

Parameters

`init` Pointer to a memory location containing the system values.

Return value

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

These settings apply to all connections. If you decide to modify H1 system values by means of **H1SetStandardvalues** while connections are active, the new parameters will apply with immediate effect. This may have the undesired result that a connection is subject to old and new parameters.

4.2.2.15 Write H1 system values

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SetStandardvalues(  
    H1_Initvalues W_Pointer init)
```

This function modifies H1 standard values.

Parameters

`init` Pointer to a memory location that contains the new system values.

Return value

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

These settings apply to all connections. If you decide to modify H1 system values by means of **H1SetStandardvalues** while connections are active, the new parameters will apply with immediate effect. This may have the undesired result that a connection is subject to old and new parameters.

H1 system values

TimeoutAck	An Ack will be issued when this timer has expired.
TimeoutCrSchnell	A new CR will be issued “quickly” when this timer has expired.
TimeoutCrLangsam	A new CR will be issued “slowly” when this timer has expired.
TimeoutSend	When this timer expires the flag “cannot send data” is valid.
TimeoutRec	When this timer expires the flag “no data read” is valid.
TimeoutLive	When this timer expires a non-existent connection is recognized.
TimeoutRetrySend	When this timer expires the send operation is repeated.
TimeoutNewSend	When this timer expires before a confirmation is received the respective send operation is repeated.
NoCrKurz	Specifies the number of quick CR’s before slow CR’s are issued.
NoRetrySend	Specifies the number of retries when a send operation is not followed by an answer.
MaxCredit	Maximum credit value.
TPDUSize	Maximum value for length of data, in H1 format.
ClassOptions	Standard or extended mode.
ProtOption	Checksum, expedited data transfer.
TimeoutWait	Timeout for Send Rec with Semaphore. -1 = wait always.
res[12]	must be 0.

4.2.2.16 Set MS-DOS entry vector

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SetVector(unsigned short Vector)
```

SetThis function modifies the MS-DOS entry vector. This should only be changed if the system can not operate with the default value.

Parameters

Vector From 78h .. 7Fh. (default: 7Ah)

Return value

alter Vector
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This function is only available for MS-DOS.

This vector must correspond to the entry in PROTOCOL.INI, section:

[H1PROT_NIF] MSDOSVEKTOR = XX

H1SetVector must be executed before **H1DriverOpen**, as it is impossible to execute **H1SetVector** otherwise.

See also

H1DriverOpen
H1DriverClose

4.2.3 Specific H1 Layer 4 functions

4.2.3.1 Send data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SendData(H1_SENDFPARAMS W_POINTER send)
```

This function sends data via an existing connection. When the function is executed it will only return when send data has been successful or when a timeout occurs.

Parameters

send Pointer to a structure containing send parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.
If the telegram should form part of a blocked transmission bit 15 in the element **Vnr** must be set. On the receiving side this causes that H1_BLOCKED_DATA is entered into the Fehler element. Bit 15 may not be set in the last telegram of a blocked transmission.

Possible return values from `send.Fehler`:

```
0
H1_WAIT_CONNECT
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
```

Return value

`< > 0` The operating system has detected an error.
`0` The `send.Fehler` element contains one of the values mentioned above

Note

This function is normally used in multitasking systems, as it only returns to the caller once the send operation has been successful or when a timeout occurs. It is often executed from within a thread. It may be started by means of **H1StartSend** and its status can be determined (polled) by means of **H1CheckSend**.

You may not mix the **H1SendData** and **H1StartSend** / **H1CheckSend** tasks for a single connection. The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1StartSend
H1CheckSend
```

4.2.3.2 Send expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SendDataEx(H1_SENDPARAMS W_POINTER send)
```

This function sends expedited data via an existing connection provided that the connection is capable of this mode. Expedited Data is data that has priority over other data, i.e. high-speed data.

Parameters

send Pointer to a structure containing send parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.
If the telegram should form part of a blocked transmission bit 15 in the element **Vnr** must be set. On the receiving side this causes that H1_BLOCKED_DATA is entered into the Fehler element. Bit 15 may not be set in the last telegram of a blocked transmission.

Possible return values from `send.Fehler`:

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NOT_SUPPORTED
H1_NO_ADAPTER
```

Return value

`< > 0` The operating system has detected an error.
0 The `send.Fehler` element contains one of the values mentioned above.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1StartSend
H1CheckSend
```


4.2.3.3 Start Send

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StartSend(H1_SENDPARAMS W_POINTER send)
```

This function starts the send operation. The function **H1CheckSend** must be executed in a loop until it returns a value indicating that the send operation was successful.

Parameters

send	Pointer to a structure containing send parameters. Parameters Vnr and DataLen must contain the respective values. Ensure that the buffer at the end of the structure is large enough. If the telegram should form part of a blocked transmission bit 15 in the element Vnr must be set. On the receiving side this causes that H1_BLOCKED_DATA is entered into the Fehler element. Bit 15 may not be set in the last telegram of a blocked transmission.
------	--

Possible return values from send.Fehler:

```
0
H1_WAIT_CONNECT
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_ALREADY_RUNNING
```

Return value

< > 0	The operating system has detected an error.
0	The send.Fehler element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. It starts a send operation. The result of the function can be determined by means of **H1CheckSend**. You may not mix the **H1SendData** and **H1StartSend** / **H1CheckSend** tasks for a single connection. The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1SendData
H1CheckSend
```

4.2.3.4 Check send status

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1CheckSend(H1_SENDFPARAMS W_POINTER send)
```

This function determines whether a send task that was initiated by means of **H1StartSend** has completed.

Parameters

send Structure containing send parameters.

Possible return values from send.Fehler:

```
0
H1_WAIT_CONNECT
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NO_REQUEST
```

Return value

< > 0 The operating system has detected an error.

0 The send.Fehler element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. It starts a send operation. You may not mix the **H1SendData** and **H1StartSend** / **H1CheckSend** tasks for a single connection. The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**. When a value of **H1_WAIT_SEND** is returned the **H1CheckSend** function must be repeated (Polling).

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1SendData
H1StartSend
```

The following page contains a flowchart that explains how you should use the functions in your programs.

The send data example in chapter 6 operates according to this flowchart.

4.2.3.5 Flowchart "send a telegram"

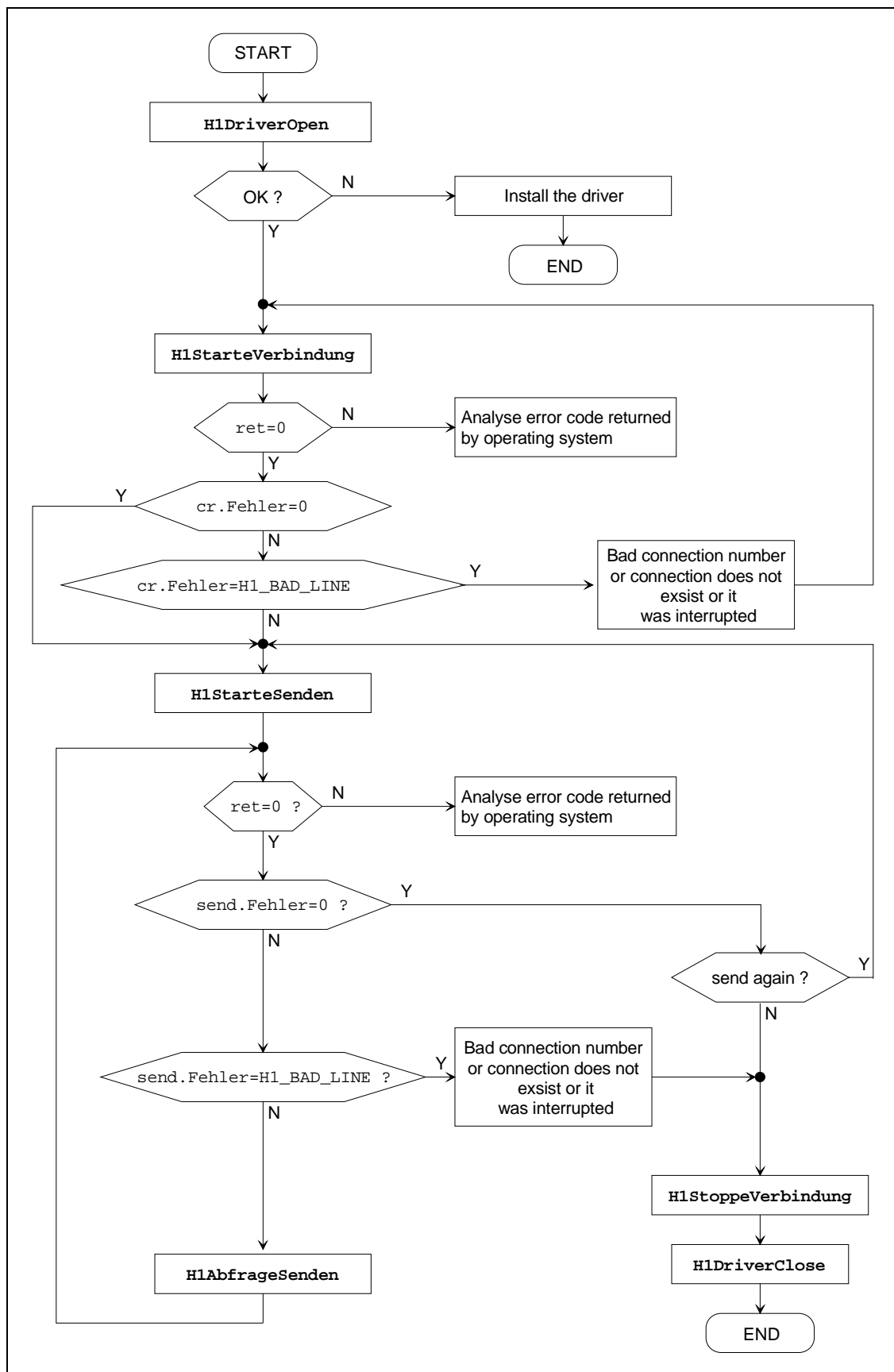


Fig. 4-2: Flowchart for "Send a telegram"

4.2.3.6 Read data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1ReadData(H1_REC_PARAMS W_POINTER rec)
```

This function is used to read data. It will only return to the caller once data has been received or after a timeout has occurred.

Parameters

rec Pointer to a structure containing receive parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.

Possible values returned by `rec.Fehler`:

```
0
H1_WAIT_CONNECT
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
```

Return value

`< > 0` The operating system has detected an error.

0 The `rec.Fehler` element contains one of the values mentioned above.

Note

This function is normally used in multitasking systems, as it only returns to the caller once the receive operation has been successful or when a timeout occurs. It is often executed from within a thread. It may be started by means of **H1StartRead** and its status can be determined (polled) by means of **H1CheckRead**.

You may not mix the **H1SendRead** and **H1StartRead** / **H1CheckRead** tasks for a single connection.

The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1StartRead
H1CheckRead
```

4.2.3.7 Read expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1ReadDataEx(H1_REC_PARAMS W_POINTER rec)
```

This function initiates a read task for expedited data. Expedited data consist of priority data or urgent data.

Parameters

rec	Pointer to a structure containing receive parameters. Parameters Vnr and DataLen must contain the respective values. Ensure that the buffer at the end of the structure is large enough. Possible values returned by <code>rec.Fehler</code> : <div style="margin-left: 40px;"> 0 H1_WAIT_CONNECT H1_BLOCKED_DATA H1_BAD_LINE H1_WAIT_DATA H1_NO_DRIVER H1_NO_ADAPTER H1_ALREADY_RUNNING </div>
-----	--

Return value

< > 0	The operating system has detected an error.
0	The <code>rec.Fehler</code> element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. It starts a read operation. You may determine the result of the function by means of **H1CheckReadEx**. You may not mix the **H1ReadDataEx** and **H1StartReadEx** / **H1CheckReadEx** tasks for a single connection.

The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

When a value of `H1_WAIT_SEND` is returned the **H1StartReadEx** function must be repeated.

See also

```

H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1StartReadEx
H1CheckReadEx

```

4.2.3.8 Start read

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StartRead(H1_REC_PARAMS W_POINTER rec)
```

This function starts a read task.

Parameter

rec Pointer to a structure containing receive parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.

Possible values returned by **rec.Fehler**:

```
0
H1_WAIT_CONNECT
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_ALREADY_RUNNING
```

Return value

< > 0 The operating system has detected an error.

0 The **rec.Fehler** element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. It starts a read operation. You may determine the result of the function by means of **H1CheckRead**. You may not mix the **H1ReadData** and **H1StartRead**/ **H1CheckRead** tasks for a single connection.

The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

When a value of **H1_WAIT_SEND** is returned the **H1StartRead** function must be repeated.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1ReadData
H1CheckRead
```

4.2.3.9 Start read expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StartReadEx(H1_RECPARAMS W_POINTER rec)
```

This function initiates a read task for expedited data. Expedited data consists of priority data or urgent data.

Parameter

rec Pointer to a structure containing receive parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.

Possible values returned by **rec.Fehler**:

```
0
H1_WAIT_CONNECT
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_ALREADY_RUNNING
```

Return value

< > 0 The operating system has detected an error.

0 The **rec.Fehler** element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. It starts a read operation. You may determine the result of the function by means of **H1CheckReadEx**. You may not mix the **H1ReadDataEx** and **H1StartReadEx** / **H1CheckReadEx** tasks for a single connection.

The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**.

When a value of **H1_WAIT_SEND** is returned the **H1StartReadEx** function must be repeated.

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1ReadDataEx
H1CheckReadEx
```

4.2.3.10 Check read

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1CheckRead(H1_REC_PARAMS W_POINTER rec)
```

This function determines whether a read task that was initiated by means of **H1StartRead** is still active.

Parameters

rec Pointer to a structure containing receive parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.

Possible values returned by `rec.Fehler`:

```
0
H1_WAIT_CONNECT
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NO_REQUEST
```

Return value

`< > 0` The operating system has detected an error.

`0` The `rec.Fehler` element contains one of the values mentioned above.

Note

This function is normally executed in singletasking systems. You may not mix the **H1ReadData** and **H1StartRead** / **H1CheckRead** tasks for a single connection. The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**. When a value of `H1_WAIT_SEND` is returned the **H1CheckRead** function must be repeated (polling).

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1ReadData
H1StartRead
```

The following page contains a flowchart that shows how you should use these functions in your programs.

The example for a read operation shown in chapter 6 operates according to this flowchart.

4.2.3.11 Check read expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1CheckReadEx(H1_REC_PARAMS W_POINTER rec)
```

This function determines whether a read task for expedited data that was initiated by **H1StartReadEx** is still active. Expedited data consists of priority data or urgent data.

Parameter

rec Pointer to a structure containing receive parameters. Parameters **Vnr** and **DataLen** must contain the respective values. Ensure that the buffer at the end of the structure is large enough.

Possible values returned by `rec.Fehler`:

```
0
H1_WAIT_CONNECT
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NO_REQUEST
```

Return value

`< > 0` The operating system has detected an error.

`0` The `rec.Fehler` element contains one of the above mentioned values.

Note

This function is normally executed in singletasking systems. You may not mix the **H1ReadDataEx** and **H1StartReadEx** / **H1CheckReadEx** tasks for a single connection. The driver must first be initiated by means of **H1DriverOpen** and a connection must also be established by executing **H1StartConnect** or **H1StartConnectCard**. When a value of `H1_WAIT_SEND` is returned the **H1CheckReadEx** function must be repeated (polling).

See also

```
H1DriverOpen
H1DriverClose
H1StartConnect
H1StartConnectCard
H1ReadDataEx
H1StartReadEx
```

4.2.3.12 Flowchart "Read a message"

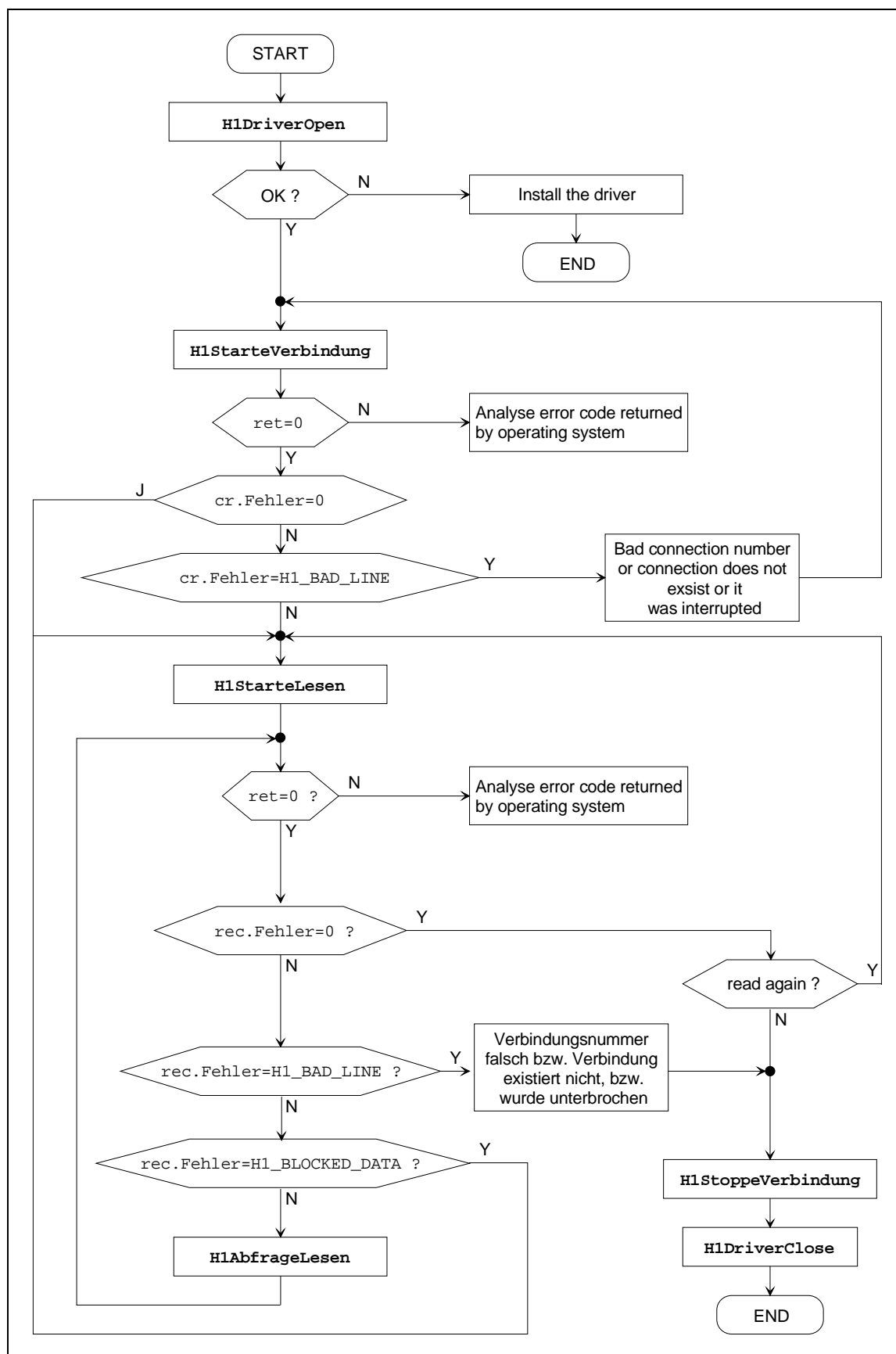


Fig. 4-3: Flowchart for "Read a telegram"

4.3 PLC Layer 7 program interface

4.3.1 General information on the PLC program interface

The PLC access functions are required to access controllers of the Industrial Ethernet (H1) network.

On a controller it is possible to access all areas e.g. data modules, function modules, system modules etc. You may, for instance, alter individual data items or load and overwrite complete modules. The different controllers on a network are addressed by means of connection parameters. It is also possible that multiple connections to a destination system exist.

The PLC functions represent the active part of a connection. Data is retrieved from a passively configured controller (only `SendAll` and `RecAll`). The respective parameters are specified individually for every station. All the functions defined on the H1 level are supported.

The functions `S5FetchPassiv` and `S5WritePassiv` provide a basis for a simple PLC-simulation.

The PLC program interface does not depend on the operating system, i.e. all PLC functions are available irrespective of the type of operating system.

Before a station becomes accessible you must first establish a connection with the destination system. This connection may then be used to exchange data with the destination system.

It is possible to maintain a number of connections between different systems or even to a single system.

When a connection is no longer required it should be closed down.

The connection parameters determine the type of connection. These parameters are provided in a central connection file; however, the user may also supply individual values where required. The recommended method is to use the connection file as this provides for better documentation. This standard parameter file is called `Net.net`.

Connection files may be modified by means of an editor. You may use any ASCII editor for DOS and in OS/2, WINDOWS and Windows NT any one of the common windows based text editors will suffice. Here you may also use an ASCII editor as the connection files contain plain ASCII data.

A description of the file format is contained in chapter 5.

4.3.2 General Layer 7 functions

4.3.2.1 Start connection

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5StartConnection (
    H1_CONNECT_PARAMS W_POINTER cr)
```

The **S5StartConnection** function is used to define a connection to another system on the network.

This function must be executed before a connection can be used.

Parameters

cr Pointer to a structure containing connection parameters. These parameters are read by means of **S5GetConnectionParameter**.

The value **Vnr** is set if the function completes without error. This represents the connection number. However, this number will only be returned if

- all drivers have been installed.
- the connection parameters are correct.
- the remote station acknowledges the connection if required by the connection parameters.

A valid **Fehler! Textmarke nicht definiert**.connection number does not imply that the connection to the partner exists. This also depends on the connection parameters.

Return value

0
H1_BAD_CR_PARAMS
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StartConnectionCard
S5StopConnection

4.3.2.2 Start a connection for multiple adapters

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5StartConnectionCard (
    H1_CONNECT_PARAMS_LINE W_POINTER cr)
```

The **S5StartConnection** function is used to define a connection to another system on the network.

This function must be executed before a connection can be used.

Parameter

cr	Pointer to a structure containing connection parameters. These parameters are read by means of S5GetConnectionParameterCard .
----	--

The value **Vnr** is set if the function completes without error. This represents the connection number. However, this number will only be returned if

- all drivers have been installed.
- the connection parameters are correct.
- the remote station acknowledges the connection if required by the connection parameters.

A valid connection number does not imply that the connection to the partner exists. This also depends on the connection parameters.

Return value

0
H1_BAD_CR_PARAMS
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StartConnection
S5StopConnection

4.3.2.3 Terminate a connection

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5StopConnection (unsigned short
connectionnumber)
```

The **S5StopConnection** function is used to terminate a defined connection. It must be executed even if the connection has never truly existed. When this function completes all the internal memory locations in the library and in the drivers are released.

Parameter

connectionnumber	Here you must supply a valid connection number of a connection that was started by means of S5StartConnection and that has not yet been terminated by S5StopConnection .
------------------	--

Return value

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StartConnection
S5StartConnectionCard

4.3.2.4 Get revision levels

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5GetRevision(CP_REVISION W_POINTER cprev)
```

This function returns the revision levels of all installed drivers and libraries in a structure called **cprev**.

In your program you should always make sure that the revision levels of installed drivers match or exceed a minimum level. It is possible that some functions contained in this description are not implemented fully if they have an earlier revision level than required.

Parameters

cprev	Pointer to the structure CP_REVISION. The element cb must be contain the length of the structure. The easiest manner to achieve this is by means of sizeof(CP_REVISION)
--------------	---

Return value

0	OK.
1	Incorrect Length of structure.

See also

S5StartConnection
S5StartConnectionCard
S5StopConnection

4.3.2.5 Set NET.NET-file name

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5SetNetfileName(char W_POINTER Filename);
```

This function assigns a new value to the name and the path for the file NET.NET.

Parameters

Filename	Pointer to a memory location containing the new file name and path. All net-file functions where the filename is not explicitly specified retrieve the new file name from this location.
----------	--

Return value

0	OK, name was saved
1	File name too long

Note

The specified path must exist. If the new file does not exist it will be created. A file with the same file name will be replaced.

4.3.2.6 Read NET.NET-file name

```
#include <S5ACCESS.H>
```

```
void WENTRY_C S5GetNetfileName(char W_POINTER filename);
```

This function returns the current file name of the Net.Net-file.

Parameters

Filename	Pointer to a memory location containing the new file name and path.
----------	---

4.3.3 File functions Layer 7

The following functions refer to the parameter file that is described in the appendix. These functions may be divided into two groups:

- File functions that refer to the standard parameter file. This is the `Net.net` file.
- File functions that refer to a file with a specified name.

File functions for the standard parameter file

The standard parameter file is the file `Net.net` which defaults to the root directory of drive C:. The name and path of the file may be changed. You can do this as follows:

1. Declare the variable `CONNETFILE` in your system. This variable can then be used to change the name and the path of the file.

Example: The file is called `H1.net` and it is located in the directory `C:\H1`

```
SET CONNETFILE = C:\H1\H1.net
```

2. Execute the following function:

```
Set NetFileName (char * Filename)
```

File function for a file with a specified name

The file name is supplied to the parameter file together with the function call.

4.3.3.1 Get connection parameter from standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5GetConnectionParameter (
    S5_VERBINDUNGSDATEN W_POINTER s5data)
```

Certain parameters are required when assigning a connection by means of the function **S5StartConnection**. The function **S5GetConnectionParameter** is available to retrieve these parameters from a file.

The respective file may be created and modified by means of the same program that was used to configure the module VIPA CP143 plus (/QP). It is also possible to use another text editor that does not write control characters to the data is located in a pure text file. Chapter 5 contains the exact format of this file.

Parameters

s5data is a pointer to the structure **S5_VERBINDUNGSDATEN**. The element **Verbindungsname** must contain the connection name.

Return value

0	OK. The data was entered into the structure.
TEXT_NO_NET_FILE	The connection file does not exist.
TEXT_NO_SECTION	The connection name could not be found.
TEXT_NO_MEM	There is not enough memory available for processing this function.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

```
S5StartConnection
S5StartConnectionCard
S5StopConnection
S5GetConnectionCard
S5PutConnectionParameter
S5WriteConnectionCard
S5ListConnections
```

4.3.3.2 Get connection parameters for multiple adapters from standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5GetConnectionCard(
    S5_VERBINDUNGSDATEN_KARTE W_POINTER s5data)
```

A number of parameters are required to define a connection by means of, for example, the function **S5StartConnectionCard**. To allow for external configuration these parameters may be retrieved from a file using the function **S5GetConnectionCard**.

The respective file may be created and modified by means of the same program that was used to configure the module VIPA CP143 plus (/QP). It is also possible to use another text editor that does not write control characters to the data is located in a pure text file. Chapter 5 contains the exact format of this file.

Parameters

s5data is a pointer to the structure **S5_VERBINDUNGSDATEN_KARTE**. The element **Verbindungsname** must contain the connection name.

Return value

0	OK. The data was entered into the structure.
TEXT_NO_NET_FILE	The connection file does not exist.
TEXT_NO_SECTION	The connection name could not be found.
TEXT_NO_MEM	There is not enough memory available for processing this function.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

```
S5StartConnection
S5StartConnectionCard
S5StopConnection
S5GetConnectionParameter
S5PutConnectionParameter
S5WriteConnectionCard
S5ListConnections
```

4.3.3.3 Put connection parameters to standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5PutConnectionParameter (
    S5_VERBINDUNGSDATEN W_POINTER s5data)
```

Any program that provides a user interface for the configuration of the H1 connection may save the entered data by means of the **S5PutConnectionParameter** function. This function uses the same file as previously described for **S5GetConnectionParameter**.

Parameters

s5data is a pointer to the structure S5_VERBINDUNGSDATEN. All elements must contain the required values.

Return value

0	OK. The data was entered into the file.
TEXT_NO_MEM	There is not enough memory available for processing this function.
TEXT_DISK_FULL	A general error occurred when the file was written to disk.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5StartConnection
S5StartConnectionCard
S5StopConnection
S5GetConnectionParameter
S5GetConnectionCard
S5WriteConnectionCard
S5ListConnections

4.3.3.4 Write connection parameters for multiple adapters into standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5WriteConnectionCard (
    S5_VERBINDUNGSDATEN_KARTE W_POINTER s5data)
```

Any program that provides a user interface for the configuration of the H1 connection may save the entered data by means of the **S5WriteConnectionCard** function. This function uses the same file as previously described for **S5GetConnectionParameterCard**.

Parameters

s5data is a pointer to the structure S5_VERBINDUNGSDATEN_KARTE.
All elements must contain the required values.

Return value

0	OK. The data was entered into the file.
TEXT_NO_MEM	There is not enough memory available for processing this function.
TEXT_DISK_FULL	A general error occurred when the file was written to disk.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5StartConnection
S5StartConnectionCard
S5StopConnection
S5GetConnectionParameter
S5GetConnectionCard
S5PutConnectionParameter
S5ListConnections

4.3.3.5 Get a list of connections from the standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5ListConnections ( unsigned short len,
      char W_POINTER mem)
```

The **S5ListConnections** function returns a list containing all the names of the connections that are currently defined in the file. This may be used to check whether a certain connection exists.

Parameters

len	specifies the length of the memory location.
mem	is a pointer to the memory location. The length of this memory location must be at least as long as specified by speicherlen.

Return value

0	OK. The connection names are located in the memory area, each one separated from the previous entry by a null and a single byte. Bit 0: a 1 indicates that the connection is in use. Bit 1: always 0 Bit 2: a 1 indicates an H1-connection Bit 3: a 1 indicates a TCP/IP-connection. Bit 4: a 1 indicates a send connection (SEND DIRECT) Bit 5: a 1 indicates a receive connection (RECEIVE DIRECT) The list is terminated by two consecutive nulls
TEXT_NO_NET_FILE	The connection file does not exist.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

```
S5StartConnection
S5StartConnectionCard
S5StopConnection
S5GetConnectionParameter
S5GetConnectionCard
S5PutConnectionParameter
S5WriteConnectionCard
```

4.3.3.6 Get list of connections from specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>
```

```
unsigned short WENTRY_C S5ListNetConnections ( char W_POINTER
        filename, short len, char mem)
```

The **S5ListNetConnections** function returns a list containing all the names of the connections that are currently defined in the file. This may be used to check whether a certain connection exists.

Parameter

filename	the name of the Net file, containing the list of connections.
len	specifies the length of the memory location.
mem	is a pointer to the memory location. This area of memory must have a minimum length of <code>speicherlen</code> .

Return value

0	<p>OK. The connection names are located in the memory area, each one separated from the previous entry by a null and a single byte.</p> <p>Bit 0: a 1 indicates that the connection is in use. Bit 1: always 0 Bit 2: a 1 indicates an H1-connection Bit 3: a 1 indicates a TCP/IP-connection. Bit 4: a 1 indicates a send connection (SEND DIRECT) Bit 5: a 1 indicates a receive connection (RECEIVE DIRECT)</p> <p>The list is terminated by two consecutive nulls</p>
TEXT_NO_NET_FILE	The connection file does not exist.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

```
H1ReadParameter
H1WriteParameter
S5ReadParameter
S5WriteParameter
```

4.3.3.7 Set MS-DOS vector

```
#include <S5Access.h>
```

```
unsigned short WENTRY_C S5SetVector(unsigned short Vector)
```

This function changes the MS-DOS vector. This value should only be changed if the system can not operate with the default.

Parameters

Vector range 78h .. 7Fh. (default: 7Ah)

Return value

old Vector
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This function is only available for MS-DOS.

This vector must correspond to the value specified in PROTOCOL.INI section:

[H1PROT_NIF] MSDOSVEKTOR = XX

4.3.3.8 Set Station address

```
#include <S5Access.h>
```

```
unsigned short WENTRY_C S5SetStationAddress(unsigned char  
W_POINTER address)
```

This function changes the local (own) station address.

Parameter

address	Pointer to a memory location containing the modified station parameters. The remaining 6 bytes contain the station address. For an H1 environment the highest byte may not contain a broadcast- or multicast-address.
---------	--

Return value

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

The station address must be unique, otherwise it is possible that collisions occur on the network.

The station address is independent of other protocols, e.g. IPX, if the setting “Protokollieren = Ja” (log protocol = yes) has been specified. On some LAN’s the **MAC**-driver may not be set to operate in **promiscuous mode**. In this case you must set “Protokollieren = Nein” (log protocol = no). Other systems refuse to cooperate when the station address is modified at run time, i.e. the setting is “Protokollieren = Ja” (log protocol = yes). The system is loaded drastically by the promiscuous mode. Please refer to the **technical reference** for your network for further information in this regard.

Example

```
unsigned char address [6];  
    station[0] = 0x00;  
    station[1] = 0x20;  
    station[2] = 0xd5;  
    station[3] = 0x80;  
    station[4] = 0x02;  
    station[5] = 0x01;  
    S5SetStationAddress(address);
```

See also

S5SetStationAddressCard

4.3.3.9 Set station address for multiple adapters

```
#include <S5Access.h>
```

```
unsigned short WENTRY_C S5SetStationAddressCard(unsigned  
    char W_POINTER address, unsigned short card)
```

This function modifies the local (own) station address. The function must be used on systems where multiple adapters are installed. The function may also be used if the system contains only a single adapter.

Parameters

address Pointer to a memory location containing the modified station parameters. The remaining 6 bytes contain the station address.

For an H1 environment the highest byte may not contain a broadcast- or multicast-address.

card adapter number. 0 is adapter 1

Return value

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

See note for **S5SetStationAddress** chapter 4.3.3.8

Example

```
unsigned char address [6];  
    station[0] = 0x00;  
    station[1] = 0x20;  
    station[2] = 0xd5;  
    station[3] = 0x80;  
    station[4] = 0x02;  
    station[5] = 0x01;  
S5SetStationAddressCard(address,card);
```

See also

S5SetStationAddress

4.3.3.10 Read a record of H1 parameters from the specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>

unsigned short WENTRY_C H1ReadParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    H1_CONNECT_PARAMS_LINE W_POINTER cr)
```

Reads one record of H1 parameters from the specified Net file.

Parameters

netfile	name of .Net file.
vname	name of the respective connection.
cr	structure where parameters are saved.

Return value

0	OK. The data is located in the structure.
TEXT_NO_NET_FILE	The connection file does not exist.
TEXT_NO_SECTION	The name of the connection could not be found.
TEXT_NO_MEM	Insufficient memory for processing this function.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5ListNetConnections
H1WriteParameter
S5ReadParameter
S5WriteParameter

4.3.3.11 Write a record of H1 parameters to specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>

unsigned short WENTRY_C H1WriteParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    H1_CONNECT_PARAMS_LINE W_POINTER cr)
```

Write a record of H1 parameters to the specified .Net file.

Parameters

netfile	Name of the .Net file.
vname	Name of the respective connection.
cr	Structur which is used to transfer the parameters.
	All parameters must have been defined.

Return value

0	OK. The data has been entered into the structure.
TEXT_NO_MEM	Insufficient memory for processing this function.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5ListNetConnections
H1ReadParameter
S5ReadParameter
S5WriteParameter

4.3.3.12 Read a record of PLC parameters from specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>

unsigned short WENTRY_C S5ReadParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    S5_ANSCHALTUNG W_POINTER s5)
```

Reads one record of H1parameters from the specified .Net file.

Parameter

netfile	Name of the .Net file.
vname	Name of the respective connection.
s5	Structure where retrieved parameters are stored.

Return value

0	OK. The data has been entered into the structure.
TEXT_NO_NET_FILE	The connection file does not exist.
TEXT_NO_SECTION	The name of the connection could not be found.
TEXT_NO_MEM	Insufficient memory for processing this function.

Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5ListNetConnections
H1ReadParameter
H1WriteParameter
S5WriteParameter

4.3.3.13 Write a record of PLC parameters to specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
#include <NETFILE.H>

unsigned short WENTRY_C S5WriteParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    S5_ANSCHALTUNG W_POINTER s5)
```

Writes a record of H1 parameters into the specified .Net file.

Parameter

netfile	Name of the .Net file.
vname	Name of the respective connection
s5	Structur which is used to transfer the parameters.
	All parameters must have been defined.

Return value

0	OK. The data has been entered into the structure.
TEXT_NO_MEM	Insufficient memory for processing this function.

Return value Note

As this function uses files it is not possible to execute it from within drivers.

See also

S5ListNetConnections
H1ReadParameter
H1WriteParameter
S5ReadParameter

4.3.4 Specific Layer 7 functions

4.3.4.1 Fetch active

4.3.4.1.1 Read from

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5ReadFromPLC (      S5_PARAMS      W_POINTER
s5,
      unsigned short speicherlen, void W_POINTER speicher,
      unsigned short W_POINTER s5fehler)
```

The **S5ReadFromPLC** function is used to read data from a PLC when the operation must not be interrupted by external events. This is normally only required for multitasking operating systems like OS/2 or Windows NT.

Parameters

s5	Pointer to a structure of the type S5_PARAMS. The values: Kennung , DB , DW and Len must contain valid entries. The structure s5 must contain a valid connection number that was started by means of the function S5StartConnection and that has not yet been terminated by the S5StopConnection function.
speicherlen	defines the length (in bytes) of the memory area. This memory area must be large enough to accommodate all values that must be retrieved, otherwise the function must be executed more than once to read all data. When S5StartRead is executed again any data that was previously retrieved is overwritten.
speicher	is the pointer to the memory area where the retrieved values are stored. Its length must at least equal speicherlen.
s5Fehler	is a pointer to an integer variable. If the return value of the function is 0 then the element s5Fehler contains the respective error code returned from the other station. A number of error codes are located in the table on page 4-71. This indicates that a 0 in s5Fehler means that data was transferred without error.

Return value

```
0
H1_BAD_LINE
H1_WAIT_CONNECT
H1_NO_DRIVER
H1_NO_ADAPTER
```

Note

This function is usually used in a multitasking environment. The function returns control only when the data is complete or when an error is detected. This may require some time (up to app. 15 min.) depending on the connection parameters as well as the responses from the other station. This function can not be interrupted.

See also

```
S5StartRead
S5CheckRead
```

4.3.4.1.2 Start Read

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5StartRead (S5_PARAMS W_POINTER s5)
```

The **S5StartRead** function initializes a read operation.

The requested data is not returned immediately. You must first issue the function **S5CheckRead**. You must not issue multiple start read operations for a connection without retrieving returned data by means of **S5CheckRead**.

Parameter s

s5 Pointer to a structure of the type S5_PARAMS. The values: **Kennung**, **DB**, **DW** and **Len** must contain valid entries.

The structure s5 must contain a valid connection number that was started by means of the function **S5StartConnection** and that has not yet been terminated by the **S5StopConnection** function.

Return value

0
H1_BAD_LINE
H1_WAIT_CONNECT
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This function is usually used in conjunction with **S5CheckRead** in non-multitasking environments that employ polling techniques. In multitasking systems it is advisable to use the function **S5ReadFromPLC** in a task or a thread.

See also

S5LeseAusSps
S5CheckRead

4.3.4.1.3 Check a read operation

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5CheckRead(unsigned short speicherlen,  
    void W_POINTER speicher, unsigned short W_POINTER  
    s5Fehler, unsigned short connectionnumber)
```

The **S5CheckRead** function checks whether the data that was previously requested by means of a **S5StartRead** function is available. This function must be repeated until the data becomes available or until an error message indicating a faulty connection is received.

Parameters

speicherlen	specifies the length of the memory area in bytes. This area must have enough capacity to accept all expected data, otherwise it is necessary to repeat the function until all received data has been retrieved. Any unread data is erased if the function S5StartRead is repeated.
speicher	is a pointer to the area where retrieved data must be stored. This must have a minimum length of <code>speicherlen</code> .
s5Fehler	is a pointer to an integer variable. If the function returns a value of 0 then <code>s5Fehler</code> contains the error code received from the remote station. Some of the possible error codes are located in the table on page 4-71. A 0 in <code>s5Fehler</code> indicates that the data transfer was terminated without errors.
connectionnumber	is the connection that was started by means of the function S5StartConnection and that has not yet been terminated by a S5StopConnection .

Return value

```
0  
H1_BAD_LINE  
H1_WAIT_DATA  
H1_WAIT_CONNECT  
H1_NO_DRIVER  
H1_NO_REQUEST
```

Note

This function is usually used in conjunction with **S5CheckRead** in non-multitasking environments that employ polling techniques. In multitasking systems it is advisable to use the function **S5ReadFromPLC** in a task or a thread.

See also

```
S5LeseAusSps  
S5StartRead
```

4.3.4.1.4 Flowchart "Read from the PLC"

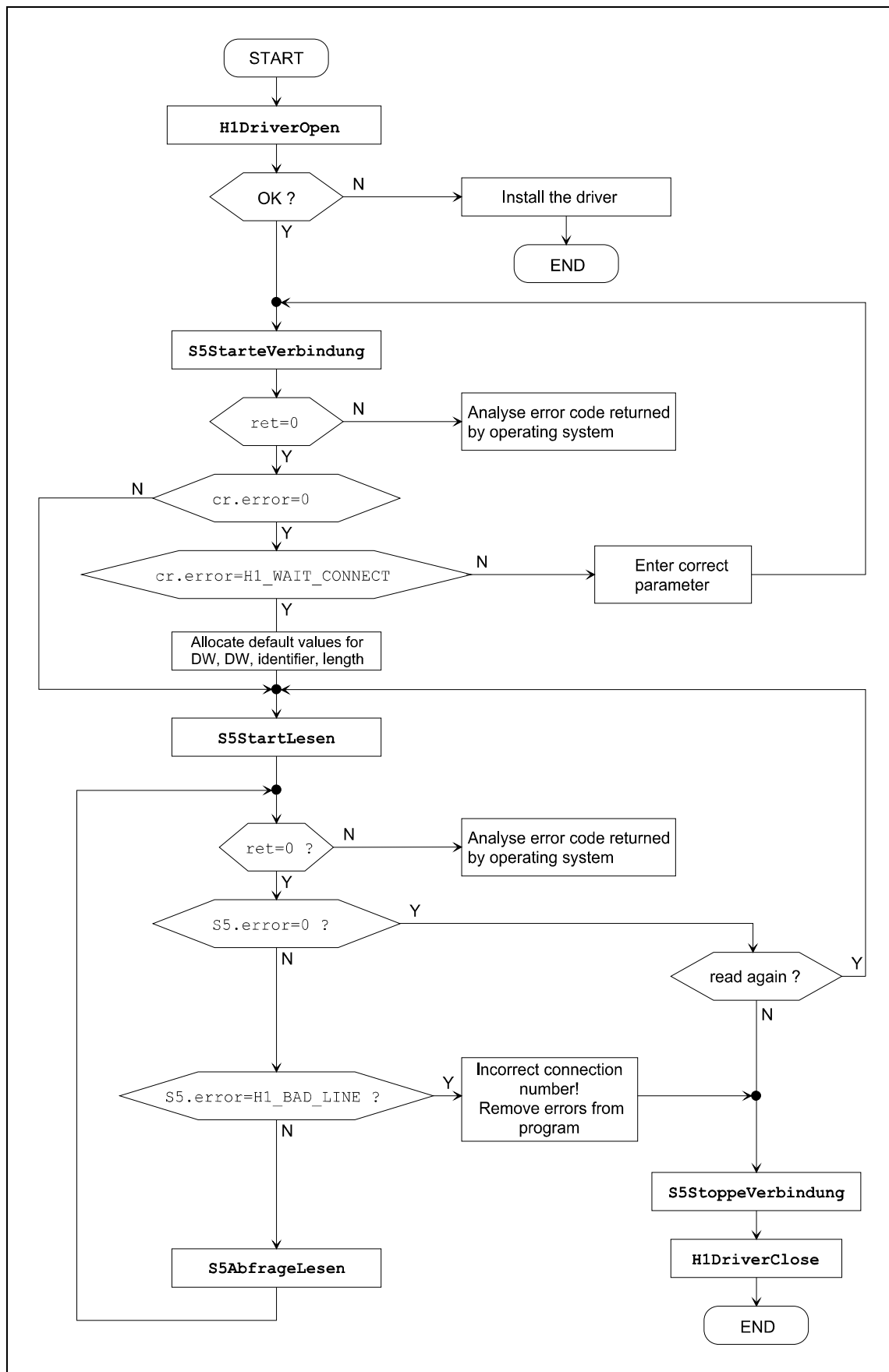


Fig. 4-4: Flowchart "Read from the PLC"

4.3.4.2 Write Active

4.3.4.2.1 Write to the PLC

```
#include <S5ACCESS.H>
```

```
unsigned short WENTRY_C S5WriteToPLC (S5_PARAMS W_POINTER s5,  
    void W_POINTER data, ushort W_POINTER s5Fehler)
```

The function **S5WriteToPLC** is used when data must be written into the PLC and when the function will not be interrupted by another event. This is normally only required for multitasking operating systems like OS/2 or Windows NT.

Parameters

s5	Pointer to a structure of the type S5_PARAMS. The values: Kennung , DB , DW and Len must contain valid entries. The structure s5 must contain a valid connection number that was started by means of the function S5StartConnection and that has not yet been terminated by the S5StopConnection function.
data	is the pointer to the data area which contains the values that must be sent. The length of this storage space is determined by the specifications contained in s5.
s5Fehler	is a pointer to an integer variable. If the function returns a value of 0 then s5Fehler contains the error code received from the remote station. Some of the possible error codes are located in the table on page 4-71. A 0 in s5Fehler indicates that the data transfer was terminated without errors.

Return value

```
0  
H1_BAD_LINE  
H1_WAIT_CONNECT  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Note

This function is usually used in multitasking operating systems. The function returns control only after the received data has been processed completely or when an error occurs. This may require some time, depending on the connection parameters and the reaction of the remote station. It is for this reason that this function can not be interrupted.

See also

```
S5StartConnection  
S5StartConnectionCard  
S5StartWrite  
S5PollWrite  
S5StopConnection
```

4.3.4.2.2 Start write operation

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5StartWrite(
    S5_PARAMS W_POINTER s5,
    void W_POINTER data)
```

The function **S5StartWrite** initializes a write operation.

You may not start more than one start write operations for a single connection.

Parameters

s5	Pointer to a structure of the type S5_PARAMS. The values: Kennung , DB , DW and Len must contain valid entries. The structure s5 must contain a valid connection number that was started by means of the function S5StartConnection and that has not yet been terminated by the S5StopConnection function.
data	is the pointer to the data area which contains the values that must be sent. The length of this storage space is determined by the specifications contained in s5.

Return value

0
H1_BAD_LINE
H1_WAIT_CONNECT
H1_NO_DRIVER
H1_ALREADY_RUNNING
H1_NO_ADAPTER

Note

This function is usually used in conjunction with **S5PollWrite** in non-multitasking environments that employ polling techniques. In multitasking systems it is advisable to use the function **S5WriteToPLC** in a task or a thread.

See also

S5StartConnection
S5StartConnectionCard
S5WriteToPLC
S5PollWrite
S5StopConnection

4.3.4.2.3 Poll a write operation

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5PollWrite(
    ushort W_POINTER s5fehler,
    ushort connectionnumber)
```

The **S5PollWrite** function is used to check whether data that was previously transferred by means of an **S5StartWrite** function has been sent or not.

This function must be repeated until the data has been sent successfully or until an error message indicating a faulty connection is received.

Parameters

s5Fehler	is a pointer to an integer variable. If the function returns a value of 0 then s5Fehler contains the error code received from the remote station. Some of the possible error codes are located in the table on page 4-71. A 0 in s5Fehler indicates that the data transfer was terminated without errors.
connectionnumber	is the connection that was started by means of the function S5StartConnection and that has not yet been terminated by a S5StopConnection .

Return value

```
0
H1_BAD_LINE
H1_WAIT_DATA
H1_WAIT_CONNECT
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_REQUEST
H1_NO_ADAPTER
```

See also

```
S5StartConnection
S5StartConnectionCard
S5WriteToPLC
S5StartWrite
```

4.3.4.2.4 Flowchart "Write to the PLC"

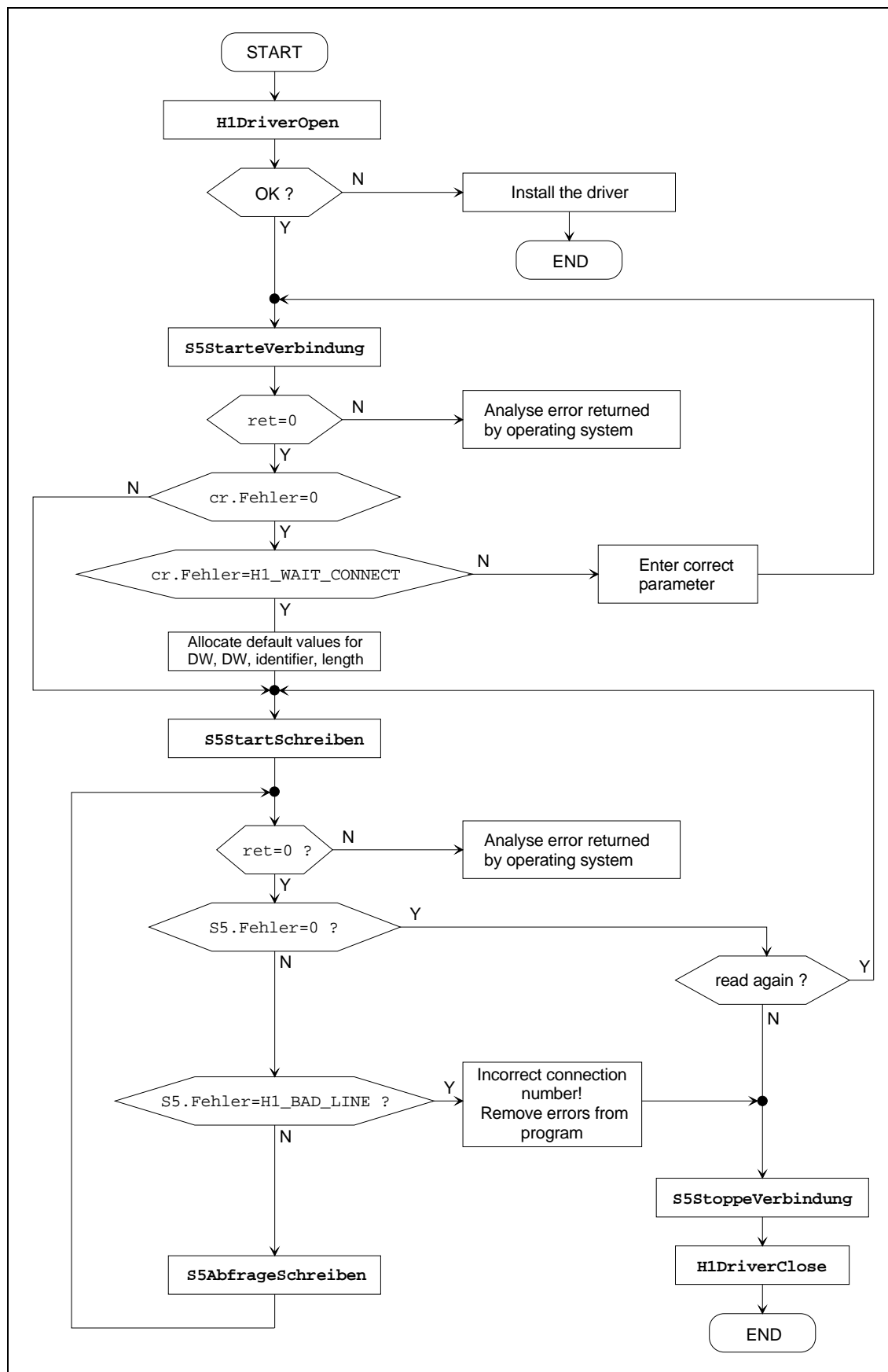


Fig. 4-5: Flowchart "Write to the PLC"

4.3.4.3 Fetch Passive

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5FetchPassiv(unsigned short
connectionnumber,
    short(WENTRY_C *RetCall)(
        short DB,
        short DW,
        short Type, short Len, short W_POINTER S5Err,
        void W_POINTER W_POINTER Data)
    void W_POINTER W_POINTER retptr)
```

The function **S5FetchPassiv** sends the data that a remote station requested by means of fetch.

Parameters

connectionnumber	is the connection that was started by means of the function S5StartConnection and that has not yet been terminates by a S5StopConnection .
RetCall	Pointer to the return call function. The requested data is supplied to this function as soon as the request from the remote station is received. This function will not be executed if the request from the remote station contains error. In this case the function S5FetchPassiv is terminated with an error 1. If the return call function returns 0 the answer is transferred to the fetch operation. Here the error code will be returned if *S5Err contains a value other than 0, otherwise the data will be returned. *Daten must contain the pointer to the data that must be transferred. The caller must provide the data according to the values contained in DB, DW and Type. The length of data may be calculated from Type and Len.
retptr	Pointer to the memory area that was transferred to the function RetCall. This may be used to release memory without the use of global variables.

Return value

```
0
H1_BAD_LINE
H1_WAIT_CONNECT
H1_NO_DRIVER
H1_NO_ADAPTER
```

Note:

This function is only available in the version that is intended for multitasking systems. It may also be used on systems running DOS but it will wait until a request is received from the remote station or until a connection error occurs.

4.3.4.4 Write Passive

```
#include <S5ACCESS.H>

unsigned short WENTRY_C S5WritePassiv(
    unsigned short connectionnumber,
    short(WENTRY_C *RetCall)(
        short DB,
        short DW,
        short Type,
        short Len,
        short W_POINTER S5Err,
        void W_POINTER W_POINTER Data)
    void W_POINTER W_POINTER retptr)
```

The **S5WritePassiv** function accepts the data which was sent by a write from the remote station.

Parameters

connectionnumber	is the connection that was started by means of the function S5StartConnection and that has not yet been terminated by a S5StopConnection .
RetCall	Pointer to the return call function. The requested data is supplied to this function as soon as the request from the remote station is received. This function will not be executed if the request from the remote station contains errors. In this case the function S5WritePassiv is terminated with an error 1. If the return call function returns 0 the answer is transferred to the fetch operation. Here the error code will be returned if *S5Err contains a value other than 0, otherwise the data will be returned. *Daten must contain the pointer to the data that must be transferred. The caller must provide the data according to the values contained in DB, DW and Type. The length of data may be calculated from Type and Len.
retptr	Pointer to the memory area that was transferred to the function RetCall. This may be used to release memory without the use of global variables.

Return value

```
0
H1_BAD_LINE
H1_WAIT_CONNECT
H1_NO_DRIVER
H1_NO_ADAPTER
```

Note

This function is only available in the version that is intended for multitasking systems. It may also be used on systems running DOS but it will wait until a request is received from the remote station or until a connection error occurs.

4.3.4.5 Transport error codes

When the value a function returns is 0 then `s5Fehler` contains the error code returned by the remote station. The following table contains some of these error codes.

Code	Description
0	no error
1	bad Q/ZTYP at the handler
2	PLC-area not available (DB not present)
3	PLC-area too small
4	QVZ-error occurred in PLC
5	error in display word (ANZW)
6	invalid ORG format
7	no data buffers available
8	no unused transport connections
9	error at the remote station
A	connection error (connection was terminated or could not be established)
B	message error (firmware error)
C	initialization error (e.g. RECEIVE to SEND)
D	termination after RESET
E	task with READ/WRITE (no initialization from PLC possible)
F	task does not exist
FF	system error

Tab. 4-8: Transport error codes

5 Files, constants and structures

5.1 Parameter file description	5-1
5.2 Constants and structures	5-4
5.2.1 H1 protocol driver definitions	5-4
5.2.2 Layer 7 interface definition	5-7
5.2.3 Definition of calling codes for drivers	5-10
5.2.4 Definition of call types	5-12

5 Files, constants and structures

5.1 Parameter file description

The default parameter file is named `Net.net`. This contains multiple sections that contain settings grouped together according to their function. The following format applies to the sections and the settings in the `Net.net` file.

[Section-name]

Key-name=value

In this example the **[Section-name]** refers to the name of a section.

The names must be enclosed in square brackets ([]) and the left hand bracket must be located at the left margin.

The declaration **Key-name=value** defines the value assigned to each and every setting. The key-name is the name of a setting. This name can consist of any combination of letters and numbers and it must be followed immediately by the equals sign (=). Depending on the type of setting, the value may consist of a string or a string enclosed in inverted commas. Most sections contain more than one setting.

You may insert remarks into the initialization file. You must start every remark with a semicolon (;).

[EigeneAdresse]	Section-name containing the station parameters.
KachelBasis=	Own station tile-basis address.
Stationsname=	Own station name
Adresse=	Own station address

The following sections of the file `Net.net` are connection names.

[Name der Verbindung]	Section-name of a connection containing the connection parameters.
------------------------------	--

The following variables may be used in a connection section:

Leitungstyp= Ethernet	The connection is provided by an ethernet H1 connection.
EthernetKarte=	The number of the adapter that you wish to address. This number may range from 1 up to the number of adapters you have installed.

EthernetStation= aabbccddeeff	<p>The ethernet address of the destination station.</p> <p>A station address consists of 6 bytes, i.e. the entry must consist of 12 digits.</p> <p>The entry is in HEX</p> <p><i>Example: EthernetStation=080006010001</i></p>
EthernetNSAP=	<p>NSAP of the destination station. NSAP has not been implemented on Siemens H1 systems.</p>
EthernetEigenerTSAP=	<p>Own TSAP.</p> <p>The length of a TSAP can be from 2 to 16 digits.</p> <p>The entry is in HEX</p> <p><i>Example: EthernetEigenerTSAP=2020202030202031</i></p>
EthernetFremderTSAP=	<p>Remote TSAP. The length of a TSAP can be from 2 to 16 digits.</p> <p>The entry is in HEX</p> <p><i>Example: EthernetFremderTSAP=2020202030202031</i></p>
EthernetStartParameter=	<p>Additional connection parameters.</p> <p>These parameters are used internally by the destination system.</p> <p>Their effect depends on the destination system.</p> <p>The entry is in HEX</p> <p><i>Example: EthernetStartParameter=20303436</i></p>
EthernetLeistungsart=	<p>Type of line.</p> <p>This parameter may contain the strings "Normal", "Normal, Aktiv", "Normal, Passiv", "Broadcast", "Multicast" and "Datagramm".</p> <p>"Normal" on its own stands for "Normal, Aktiv".</p> <p><i>Example: EthernetLeistungsart=Normal,Aktiv</i></p>

EthernetPriorität=

Connection priority.

Industrial Ethernet accepts:

Prio0, Prio1 are the "Express"-priorities. These can accept up to 16 bytes of data,

Prio2, Prio3 are the normal priorities and

Prio4 is the lowest priority.

Example: EthernetPriorität=Prio2

EthernetMulticastkreis=

This contains the multicast circuit number for a multicast connection.

It is ignored for any other type of connection.

Example: EthernetMulticastkreis=2

5.2 Constants and structures

5.2.1 H1 protocol driver definitions

The `H1Def.h` file contains the necessary definitions when the driver is called from level 4A.

```
// File H1Def.h
// Definitions for the data structures of the H1 protocol driver
// This file contains all the definitions required for calling the driver from level 4A
// Version 1.3
// Modified 1.4.1994
// Extended 14.08.95: Default driver values
//      30.11.95:Timeout for Send and Rec
//      06.01.96:Further English variable names

#ifndef H1DEF_H_INTERN // locking
#define H1DEF_H_INTERN
#include <H1Engl.h> // All variables and functions in English
// #include <H1Fra.h> // All variables and functions in French
// #include <H1Spa.h> // All variables and functions in Spanish
#include <WMKTypes.h> // Pointers and calls independent of compiler
#include <DrvFCall.h> // Driver calling codes

// Values for the element -->Fehler
#define H1_BAD_CR_PARAMS      1 // bad CR parameters
#define H1_NO_SLOT           2 // maximum number of connections are active
#define H1_WAIT_CONNECT      3 // connection not established or interrupted
#define H1_NOT_IMPLEMENTED   4 // function not implemented
#define H1_BAD_LINE          5 // bad connection number
#define H1_WAIT_DATA         6 // no data available
#define H1_WAIT_SEND         7 // data has not yet been sent
#define H1_INTERNAL_ERROR    8 // should not occur
#define H1_NO_REQUEST        9 // you have polled a task that does not exist
#define H1_NO_DRIVER         10 // a call to the H1 driver has been detected, but no MAC driver or no H1 driver exists
#define H1_UEBERLAST         11 // the command could not be executed (send was repeated 20 times)
#define H1_BLOCKED_DATA      12 // blocked data has arrived
#define H1_NO_ADAPTER        13 // the specified adapter does not exist
#define H1_ALREADY_RUNNING   14 // task is already active
#define H1_NOT_SUPPORTED     15 // the function is not supported. Normally this refers to
// expediated data, but the connection does not support this.
#define H1_TRY_AGAIN         16 // For WIN 3.x the call was issued recursively in the HW Int. Start again at a later stage
#define H1_NO_MEMORY         17 // May occur for Open Driver under Win 3.x

// Task types
#define TYP_SEND_DIREKT      1 // Send direct
#define TYP_REC_DIREKT       2 // Rec direct
#define TYP_SEND_ALL         3 // Send ALL
#define TYP_REC_ALL          4 // Read ALL
#define TYP_SEND_FETCH       5 // Fetch Send (Write)
#define TYP_REC_FETCH        6 // Fetch Read

// Connection data
#ifndef NORMAL_LINE          // These parameters are also located in OS2Ethernet.h

/* Connection types (ConnectType) */
#define NORMAL_LINE          1
#define DATAGRAMM_LINE       2
#define MULTICAST_LINE       4
#define BROADCAST_LINE       8
#define PASSIVE_LINE         0x80
#define ACTIVE_LINE          0

/* Priorities (Priority) */
#define EXPRESS_PRIORITY_0    1
#define EXPRESS_PRIORITY_1    2
#define PRIORITY_2           4
#define PRIORITY_3           8
#define PRIORITY_4          16
```



```

/* Structures */
#pragma pack(1)

#ifndef _CONN_PARAMS_DEFINED
#define _CONN_PARAMS_DEFINED

typedef struct _CONNECT_PARAMS
{
    unsigned char Priority;           /* Priority */
    unsigned char ConnectType;       /* Connection type */
    unsigned char LenDestAddr;       /* Length of destination address */
    unsigned char DestAddr[6];       /* Destination address */
    unsigned char Multicast;         /* Multicast circuit number */
    unsigned char LenNSAP;           /* Length of destination NSAP_ID */
    unsigned char NSAP[12];          /* Destination NSAP-ID */
    unsigned char LenDestTSAP;       /* Length of destination TSAP_ID */
    unsigned char DestTSAP[16];      /* Destination TSAP-ID */
    unsigned char LenOwnTSAP;        /* Length of own TSAP_ID */
    unsigned char OwnTSAP[16];       /* Own TSAP-ID */
    unsigned char LenConnParams;     /* Length of additional connect param's */
    unsigned char ConnParams[16];    /* Additional connect param's */
}CONNECT_PARAMS;

#endif // _CONN_PARAMS_DEFINED

#endif // NORMAL_LINE

typedef struct _H1_INITVALUES {
    unsigned short Cb;               // size of this structure in bytes including cb
    unsigned short TimeoutAck;        // an Ack is sent when this timer expires
    unsigned short TimeoutCrSchnell;  // a fast CR is issued when this timer expires
    unsigned short TimeoutCrLangsam;  // a slow CR is issued when this timer expires
    unsigned short TimeoutSend;       // when this timer expires the flag for data can not be sent is raised
    unsigned short TimeoutRec;        // when this timer expires the flag for data could not be read is set.
    unsigned short TimeoutLive;       // when this timer expires the connection is flagged as bad
    unsigned short TimeoutRetrySend;  // when this timer expires the connection is flagged as bad
    unsigned short TimeoutNewSend;    // when this timer expires the transmission is repeated if a confirmation is not
                                     // received
    unsigned short NoCrKurz;          // number of fast CR's, followed by slow CR's
    unsigned short NoRetrySend;       // retry counter when no answer is received upon a send
    unsigned short MaxCredit;         // maximum credit value
    unsigned short TPDUSize;          // maximum data length, in H1 format
    unsigned short ClassOptions;      // standard or Extended Mode
    unsigned short ProtOption;        // checksum, expediated data transfer
    unsigned long TimeoutWait;        // timeout for send rec with semaphore. -1 = wait always
    unsigned char res[12];            // must be 0
}H1_INITVALUES;

typedef struct H1_LINEVAL {
    short Cb;                         // the size of this structure in bytes including cb
    unsigned short Vnr;               // connection number
    unsigned short Fehler;            // error code during processing
    short MaxFrameLen;               // maximum length of telegram
    short OwnCredit;                 // own credit
    short DestCredit;               // credit of the destination station
    unsigned short ClassOptions;      // standard or extended mode
    unsigned short ProtOption;        // checksum, expediated data transfer
    unsigned long TimeoutErr;         // timeout value for connection error
    unsigned short BytesWaitSending;  // quantity of data that is ready to be sent
    unsigned short BytesWaitReceiving; // quantity of data that is ready for retrieveing
}H1_LINEVAL;

typedef struct {
    CONNECT_PARAMS CrParams;          // connection parameters
    unsigned short Vnr;               // connection number
    unsigned short Fehler;            // error code when establishing connection
}H1_CONNECT_PARAMS;

```

```

typedef struct {
    CONNECT_PARAMS CrParams;           // connection params
    unsigned short Vnr;                 // connection number
    unsigned short Fehler;              // error code when establishing connection
    unsigned short Karte;              // 0 = adapter 1
    long Timeout;                      // timeout for Send and Rec. -1 = wait until data is available
}H1_CONNECT_PARAMS_LINE;

typedef struct {
    unsigned short Vnr;                 // connection number
    unsigned short DataLen;            // data quantity
    unsigned short Fehler;             // error code when sending
    unsigned long reserved;            // reserved for internals
    unsigned char Daten[1];            // send data
}H1_SENDFPARAMS;

typedef struct {
    unsigned short Vnr;                 // connection number
    unsigned short DataLen;            // maximum data length
    unsigned short RecLen;             // actual length of data received
    unsigned short Fehler;             // error code when receiving
    unsigned long reserved;            // reserved for internals. Must be 0
    unsigned char Daten[1];            // data memory
}H1_RECPARAMS;

#pragma pack()

// initiate the function
#ifdef __cplusplus
extern "C" {
#endif
// calls for C
unsigned short WENTRY_C H1GetVersion(unsigned short W_POINTER version);
unsigned short WENTRY_C H1DriverOpen(void);
unsigned short WENTRY_C H1DriverClose(void);
unsigned short WENTRY_C H1GetStationAddress(unsigned char W_POINTER address);
unsigned short WENTRY_C H1SetStationAddress(unsigned char W_POINTER address);
unsigned short WENTRY_C H1GetStationAddressCard(unsigned char W_POINTER address,unsigned short card);
unsigned short WENTRY_C H1SetStationAddressCard(unsigned char W_POINTER address,unsigned short card);
unsigned short WENTRY_C H1StartConnect(H1_CONNECT_PARAMS W_POINTER cr);
unsigned short WENTRY_C H1StartConnectCard(H1_CONNECT_PARAMS_LINE W_POINTER cr);
unsigned short WENTRY_C H1StopConnect(unsigned short connectionnumber);
unsigned short WENTRY_C H1StopConnectAll(void);
unsigned short WENTRY_C H1StopConnect(unsigned short connectionnumber);
unsigned short WENTRY_C H1StartSend(H1_SENDFPARAMS W_POINTER send);
unsigned short WENTRY_C H1CheckSend(H1_SENDFPARAMS W_POINTER send);
unsigned short WENTRY_C H1SendData(H1_SENDFPARAMS W_POINTER send);
unsigned short WENTRY_C H1SendDataEx(H1_SENDFPARAMS W_POINTER send);
unsigned short WENTRY_C H1StartRead(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1CheckRead(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1ReadData(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1StartReadEx(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1CheckReadEx(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1ReadDataEx(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1TestStatus(H1_RECPARAMS W_POINTER rec);
unsigned short WENTRY_C H1GetStandardvalues(H1_INITVALUES W_POINTER init);
unsigned short WENTRY_C H1SetStandardvalues(H1_INITVALUES W_POINTER init);
unsigned short WENTRY_C H1GetLineCharacteristics(H1_LINEVAL W_POINTER val);
#ifdef DEBUG
unsigned short WENTRY_C H1ReadDebugBuffer(char W_POINTER buffer);
unsigned short WENTRY_C H1ReadDebugFrame(unsigned char W_POINTER buffer);
#endif
unsigned short WENTRY_C NetSendFrame(unsigned char W_POINTER buffer);
#ifdef OS2_FUNCTIONS_INCLUDED && NT_FUNCTIONS_INCLUDED
unsigned short WENTRY_C H1SetzeVektor(unsigned short vector);
#endif
#ifdef __cplusplus
}
#endif
// H1DEF_H_INTERN

```

5.2.2 Layer 7 interface definition

The S5Access.h file defines the layer 7 interface.

```
// File S5Access.h
// Version 1.1 dated 3.5.94
// Version 1.2 dated 1.4.95
// Version 1.3 dated 7.9.95

#ifndef S5ACCESS_INTERN // locking
#define S5ACCESS_INTERN
#if defined (WINDOWS_FUNCTIONS_INCLUDED) // Only for Win 3.x
#define SPEZIALFALL_AP_IM_INT
#endif

#include <H1Def.h>
// Values for Kennung (identifier) and OrgKennung (Org identifier)
#define KENNUNG_BAUSTEIN 1
#define KENNUNG_MERKER 2
#define KENNUNG_EINGANG 3
#define KENNUNG_AUSGANG 4
#define KENNUNG_PERIPHERIE 5
#define KENNUNG_ZAEHLER 6
#define KENNUNG_TIMER 7
#define KENNUNG_SYSTEMDATEN 8
#define KENNUNG_ABSOLUT 9
#define KENNUNG_ERW_BAUSTEIN 10
#define KENNUNG_EXTMEM 16
#define KENNUNG_EXT_PERIPHERIE 17
#pragma pack(1)
typedef struct {
short s5; // "S5"
unsigned char Headerlen; // 0x10
char KennungOpcode; // 1
char LenKennungOpcode; // 3
char Opcode; // Send = 3, 4 = acknowledgement
union {
struct {
char OrgBlock; // Send: 3, acknowledgement: F
char LenOrgblock; // Send: 8, acknowledgement: 3
char OrgKennung; // Send: 1, Rec: ?, acknowledgement acknowledgement: error number
char DB; // Send: module number, acknowledgement: FF
short DW; // Start DW, acknowledgement (byte) 7
short Len; //
unsigned char Voidblock; // ff
unsigned char LenVoidblock; // 2
}Start;
struct {
unsigned char Ackblock; // f
unsigned char LenABlock; // 3
unsigned char Error; // 0 = no error
unsigned char Voidblock; // ff
unsigned char LenVoidblock; // 7
}Acknowledgement;
struct {
unsigned char PgBlock; // 3
unsigned char LenPgBlock; // 6
unsigned char Startblock; // 2 = more, 1 = last
unsigned char Continblock; // 2 = first block, 0 = continuation block
unsigned char Opcode; // AS511 Opcode
unsigned char TermCode; // AS511 Termcode
unsigned char t3; // 4
unsigned char t4; // 2 oder 4
unsigned char rt1; //
unsigned char rt2; //
}Pg;
}p;
}SIENEC_AP;
typedef struct _S5_PARAMS {
```

```

    unsigned short connectionnumber;
    unsigned short Kennung;           // parameter identifier
    unsigned short DB;                // module number
    unsigned short DW;                // data word number
    unsigned short Len;              // length of data
}S5_PARAMS;

typedef struct {
    unsigned short Auftragsnummer;    // task number.
    short Offset;                    // Offset from base tile
    unsigned short Auftragsart;       // defines from task types
    short Benutzt;                    // use the connection automatically from start
}S5_ANSCHALTUNG;

typedef struct _S5_VERBINDUNGSDATEN {
    char Verbindungsname[32];        // ASCII connection name
    S5_ANSCHALTUNG S5Params;          // task no, ..
    CONNECT_PARAMS CrParams;         // TSAP, NSAP, ..
}S5_VERBINDUNGSDATEN;

typedef struct _S5_VERBINDUNGSDATEN_KARTE {
    char Verbindungsname[32];        // ASCII connection name
    S5_ANSCHALTUNG S5Params;          // task no, ..
    CONNECT_PARAMS CrParams;         // TSAP, NSAP, ..
    unsigned short Karte;            // 0 = adapter 1
}S5_VERBINDUNGSDATEN_KARTE;

typedef struct _CP_REVISION {
    unsigned short cb;                // length of the structure in bytes
    unsigned short SerNo;             // serial number of the CP
    unsigned short KachelNo;          // number of the selected tile
    unsigned short KachelRev;         // Revision of the tile driver
    unsigned short H1Rev;             // Revision of the H1 driver
    unsigned short IpRev;             // Revision of the TCP/IP driver
    unsigned short TermRev;           // Revision of the terminal
    unsigned char StationAdr[6];      // Ethernet ROM station address
}CP_REVISION;

#pragma pack()

// Prototypes: access functions for .DLL or .LIB
#ifdef __cplusplus
extern "C" {
#endif
unsigned short WENTRY_C S5SetStationAddress(unsigned char W_POINTER address);
unsigned short WENTRY_C S5SetStationAddressCard(unsigned char W_POINTER address,unsigned short card);
unsigned short WENTRY_C S5StartConnection(H1_CONNECT_PARAMS W_POINTER cr);
unsigned short WENTRY_C S5StartConnectionCard(H1_CONNECT_PARAMS_LINE W_POINTER cr);
unsigned short WENTRY_C S5StopConnection(unsigned short connectionnumber);
unsigned short WENTRY_C S5StopConnectionAll(void);
unsigned short WENTRY_C S5ReadFromPLC(S5_PARAMS W_POINTER s5,unsigned short memorylen,void W_POINTER memory,unsigned short W_POINTER s5error);
unsigned short WENTRY_C S5StarRead(S5_PARAMS W_POINTER s5);
unsigned short WENTRY_C S5CheckRead(unsigned short memorylen,void W_POINTER memory,unsigned short W_POINTER s5error,unsigned short connectionnumber);
unsigned short WENTRY_C S5WriteToPLC(S5_PARAMS W_POINTER s5,void W_POINTER data,unsigned short W_POINTER s5error);
unsigned short WENTRY_C S5StartWrite(S5_PARAMS W_POINTER s5,void W_POINTER data);
unsigned short WENTRY_C S5CheckWrite(unsigned short W_POINTER s5error,unsigned short connectionnumber);
unsigned short WENTRY_C S5FetchPassiv(unsigned short connectionnumber,short(WENTRY_C *RetCall)(short DB,short DW,short Typ,short Len,short W_POINTER S5Err,void W_POINTER W_POINTER Data),void W_POINTER W_POINTER retptr);
unsigned short WENTRY_C S5WritePassiv(unsigned short connectionnumber,short(WENTRY_C *RetCall)(short DB,short DW,short Typ,short Len,short W_POINTER Err,void W_POINTER W_POINTER Data),void W_POINTER W_POINTER retptr);
#ifdef OS2_FUNCTIONS_INCLUDED | WIN95_FUNCTIONS_INCLUDED | NT_FUNCTIONS_INCLUDED
unsigned short WENTRY_C S5SetVector(unsigned short vector);
#endif

```

```
// S5 INI file access functions
unsigned short WENTRY_C S5GetConnectionParameter(S5_VERBINDUNGSDATEN W_POINTER s5data);
unsigned short WENTRY_C S5PutConnectionParameter(S5_VERBINDUNGSDATEN W_POINTER s5data);

unsigned short WENTRY_C S5DeleteConnection(char W_POINTER connectionname);
unsigned short WENTRY_C S5ListConnections(unsigned short len,char W_POINTER mem);

unsigned short WENTRY_C S5ListNetConnections(char W_POINTER filename,unsigned short len,char W_POINTER mem);
unsigned short WENTRY_C S5WriteConnection(char W_POINTER netfile,char W_POINTER vname,S5_ANSCHALTUNG W_POINTER s5A);
unsigned short WENTRY_C S5ReadConnection (char W_POINTER netfile,char W_POINTER vname,S5_ANSCHALTUNG W_POINTER s5A);
unsigned short WENTRY_C SortConnectionEnabled(char W_POINTER buffer,S5_ANSCHALTUNG W_POINTER s5);
char W_POINTER WENTRY_C CreateTaskName(unsigned short tasktype);

unsigned short WENTRY_C H1GetLineCatacteristics (char W_POINTER netfile,char W_POINTER vname,H1_CONNECT_PARAMS_LINE W_POINTER cr);
unsigned short WENTRY_C S5GetConnectionParameters (char W_POINTER netfile,char W_POINTER vname,S5_ANSCHALTUNG W_POINTER s5);
unsigned short WENTRY_C H1WriteParameter (char W_POINTER netfile,char W_POINTER vname,H1_CONNECT_PARAMS_LINE W_POINTER cr);
unsigned short WENTRY_C S5PutParameter(char W_POINTER netfile,char W_POINTER vname,S5_ANSCHALTUNG W_POINTER s5);
unsigned short WENTRY_C ReadParameter(char W_POINTER netfile,char W_POINTER vname,H1_CONNECT_PARAMS_LINE W_POINTER cr,S5_ANSCHALTUNG W_POINTER s5);
unsigned short WENTRY_C WriteParameter(char W_POINTER netfile,char W_POINTER vname,H1_CONNECT_PARAMS_LINE W_POINTER cr,S5_ANSCHALTUNG W_POINTER s5);

unsigned short WENTRY_C S5ReadConnectionCard(S5_VERBINDUNGSDATEN_KARTE W_POINTER s5data);
unsigned short WENTRY_C S5WriteConnectionCard(S5_VERBINDUNGSDATEN_KARTE W_POINTER s5data);

unsigned short WENTRY_C S5SetNetfileName(char W_POINTER Dateiname);

unsigned char W_POINTER S5ApMalloc(unsigned short len);
void S5ApFree(void W_POINTER ptr);

#ifdef __cplusplus
}
#endif

#endif // S5ACCESS_INTERN lock
```

5.2.3 Definition of calling codes for drivers

The codes for driver calls are defined in the file `DrvFCall.h`.

```
// File DrvFCall.h
// Version 1.0
// Author: W.M. W.K.
// Date: 10.9.94
// Last alteration: 09.01.96

// Definition of the driver access codes (for DevIOCtl and INTxx)

#ifndef DRVFUNCTIONCALLS
#define DRVFUNCTIONCALLS

// Für DosDevIOCtl
#define LANMAN_PROTOKOLL      0x81

// Entrypoints
#if defined (NT_FUNCTIONS_INCLUDED) || defined (WIN95_FUNCTIONS_INCLUDED)

// NT numbers must be located on an even byte boundary, else an INTERNAL ERROR IN NT OPERATING SYSTEM will occur
#define H1_GET_DEBUGBUFFER      2048 // Only for test and debugging
#define SNIF_GET_DEBUGBUFFER    2048 // Only for test and debugging
#define H1_GET_DEBUGFRAME      2056 // Only for test and debugging
#define H1_GET_FRAMEBUFFER      2064 // Network analyzer !! Immediate !!
#define SNIF_GET_FRAMEBUFFER    2064 // Network analyzer !! Immediate !!
#define H1_SET_FILTER           2072 // Network analyzer: convert logging mode
#define SNIF_SET_FILTER         2072 // Netzwerkanalyser: Protokolliermodus umsetzen
#define H1_FRAME_WAIT_SEM      2080 // Wait for FrameSem with Timeout
#define SNIF_FRAME_WAIT_SEM     2080 // Wait for FrameSem with Timeout
#define H1_ALLOCATE_FRAMEBUFFER 2088 // Allocate or release the drivers internal (ring)buffer
#define SNIF_ALLOCATE_FRAMEBUFFER 2088 // Allocate or release the drivers internal (ring)buffer

#define LANMAN_ENTRY            2080 // For internal use only

#define H1_GET_ETHERNET_ADDRESS 2120 // Pointer to 14 Bytes. Byte 1 and 2 = len
#define H1_SET_ETHERNET_ADDRESS 2128 // Pointer to 6 Bytes
#define H1_CONNECT_REQUEST      2136 // Pointer to H1_CONNECTPARAMS
#define H1_DISCONNECT_REQUEST   2144 // Pointer to connectionnumber
#define H1_SEND_DATA            2152 // Pointer to H1_SENDPARAMS
#define H1_RECEIVE_DATA         2160 // Pointer to H1_RECPARAMS
#define H1_GET_LINEPARAMS       2168 // Pointer to H1_RECPARAMS
#define H1_START_SEND           2176 // Pointer to H1_SENDPARAMS
#define H1_START_RECEIVE        2184 // Pointer to H1_RECPARAMS
#define H1_CHECK_SEND           2192 // Pointer to H1_SENDPARAMS
#define H1_CHECK_RECEIVE        2200 // Pointer to H1_RECPARAMS
#define H1_BIND_TO_S5           2208 // Pointer to S5_BINDPARAMS
#define H1_SEND_FRAME           2216 // Network analyzer: send any frame
#define SNIF_SEND_FRAME         2216 // Network analyzer: send any frame
#define H1_DRIVER_RESET         2224 // PlugAndPlay: reset driver internally
#define SNIF_DRIVER_RESET       2224 // PlugAndPlay: reset driver internally
#define H1_GET_VERSION          2232 // Pointer to general version number
#define SNIF_GET_VERSION        2232 // Pointer to general version number
#define SNIFFER_GET_VERSION     2240 // Pointer to version number of network analyzer
#define H1_SET_ETHERNET_ADDRESS_LINE 2248 // Pointer to 6 bytes for the address, 2 bytes for the adapter
#define SNIF_SET_ETHERNET_ADDRESS 2248 // Pointer to 6 bytes for the address, 2 bytes for the adapter
#define H1_CONNECT_REQUEST_LINE 2256 // Pointer to H1_CONNECTPARAMS
#define H1_GET_ETHERNET_ADDRESS_LINE 2264 // Pointer to 14 bytes. Byte 1 and 2 = len
#define SNIF_GET_ETHERNET_ADDRESS 2264 // Pointer to 14 bytes. Byte 1 and 2 = len
#define H1_GET_INITVALUES       2272 // Pointer to H1_INITVALUES
#define H1_SET_INITVALUES       2280 // Pointer to H1_INITVALUES
#define H1_SEND_EX              2288 // Pointer to H1_SENDPARAMS
#define H1_RECEIVE_DATA_EX      2296 // Pointer to H1_RECPARAMS
#define H1_START_RECEIVE_EX     2304 // Pointer to H1_RECPARAMS
#define H1_CHECK_RECEIVE_EX     2312 // Pointer to H1_RECPARAMS
#define H1_GET_LINE_VALUES      2320 // Pointer to H1_LINEVAL
#define H1_GET_ALL_DEBUG        2328 // All indicators and conditions
```

```

#else // NT_FUNCTIONS_INCLUDED
// For Unix, OS/2, WIN3.x and Dos UCHAR
#define H1_GET_DEBUGBUFFER          0x10 // Only for test and debugging
#define SNIF_GET_DEBUGBUFFER        0x10 // Only for test and debugging
#define H1_GET_DEBUGFRAME          0x11 // Only for test and debugging
#define H1_GET_FRAMEBUFFER         0x12 // Network analyzer only for test purposes
#define SNIF_GET_FRAMEBUFFER        0x12 // Network analyzer only for test purposes
#define H1_SET_FILTER               0x13 // Network analyzer: set filter
#define SNIF_SET_FILTER             0x13 // Network analyzer: set filter
#define H1_FRAME_WAIT_SEM           0x14 // Wait for FrameSem with Timeout
#define SNIF_FRAME_WAIT_SEM        0x14 // Wait for FrameSem with Timeout
#define H1_ALLOCATE_FRAMEBUFFER     0x15 // Allocate or release the drivers internal (ring)buffer
#define SNIF_ALLOCATE_FRAMEBUFFER   0x15 // Allocate or release the drivers internal (ring)buffer

#define LANMAN_ENTRY                0x58 // For internal purposes
#define H1_GET_ETHERNET_ADDRESS     0x20 // Pointer to 14 bytes. Byte 1 and 2 = len
#define H1_GET_ETHERNET_ADDRESS_LINE 0x21 // Pointer to 14 bytes. Byte 1 and 2 = len
#define SNIF_GET_ETHERNET_ADDRESS   0x21 // Pointer to 14 bytes. Byte 1 and 2 = len
#define H1_SET_ETHERNET_ADDRESS_LINE 0x22 // Pointer to 6 bytes
#define SNIF_SET_ETHERNET_ADDRESS    0x22 // Pointer to 6 bytes
#define H1_SET_ETHERNET_ADDRESS     0x30 // Pointer to 6 bytes
#define H1_CONNECT_REQUEST          0x31 // Pointer to H1_CONNECTPARAMS
#define H1_DISCONNECT_REQUEST       0x32 // Pointer to connectionnumber
#define H1_SEND_DATA                0x33 // Pointer to H1_SENDPARAMS
#define H1_RECEIVE_DATA             0x34 // Pointer to H1_RECPARAMS
#define H1_GET_LINEPARAMS           0x35 // Pointer to H1_CONNECTPARAMS
#define H1_START_SEND               0x36 // Pointer to H1_SENDPARAMS
#define H1_START_RECEIVE            0x37 // Pointer to H1_RECPARAMS
#define H1_CHECK_SEND               0x38 // Pointer to H1_SENDPARAMS
#define H1_CHECK_RECEIVE            0x39 // Pointer to H1_RECPARAMS
#define H1_BIND_TO_S5               0x3A // Pointer to S5_BINDPARAMS
#define H1_SEND_FRAME               0x3b // Send any frame
#define SNIF_SEND_FRAME             0x3b // Send any frame
#define H1_DRIVER_RESET             0x3c // Internal driver reset
#define SNIF_DRIVER_RESET           0x3c // Internal driver reset
#define H1_CONNECT_REQUEST_LINE     0x3d // Pointer to H1_CONNECTPARAMS
#define H1_GET_VERSION              0x40 // Get version number
#define SNIF_GET_VERSION            0x40 // Get version number
#define H1_START_SEND_2             0x41 // Pointer to H1_SENDPARAMS_2
#define H1_START_RECEIVE_2          0x42 // Pointer to H1_RECPARAMS_2
#define H1_CHECK_SEND_2             0x43 // Pointer to H1_SENDPARAMS_2
#define H1_CHECK_RECEIVE_2          0x44 // Pointer to H1_RECPARAMS_2
#define SNIFFER_GET_VERSION         0x45 // Get version number
#define H1_GET_ALL_DEBUG            0x46 // All indicators and conditions
#define GENERAL_OUTPORT             0x47 // Debug: set port
#define GENERAL_INPORT              0x48 // Debug: read from port
#define GENERAL_MEMREAD             0x49 // Debug: read from memory
#define GENERAL_MEMWRITE            0x4a // Debug: write to memory
#define H1_GET_INITVALUES           0x4b // Pointer to H1_INITVALUES
#define H1_SET_INITVALUES           0x4c // Pointer to H1_INITVALUES
#define H1_SEND_EX                  0x4d // Pointer to H1_SENDPARAMS
#define H1_RECEIVE_DATA_EX          0x4e // Pointer to H1_RECPARAMS
#define H1_START_RECEIVE_EX         0x4f // Pointer to H1_RECPARAMS
#define H1_CHECK_RECEIVE_EX         0x50 // Pointer to H1_RECPARAMS
#define H1_GET_LINE_VALUES          0x51 // Pointer to H1_LINEVAL
#define H1_LAMPEN                   0x52 // Lamp control associated with telegrams
#define H1_BROADCAST                0x53 // Return for all broadcasts != H1 (FEFE in identifier)

#define GATEWAY_SET                 0x60 // Pointer to gateway entry point

#endif // NT_FUNCTIONS_INCLUDED

#endif // DRVFUNCTIONCALLS

```

5.2.4 Definition of call types

The file `WMKTypes.h` defines the types of access for different compilers and operating systems.

```
// File WMKTypes.h
// Definition of access types for different compilers and operating systems
// Version 1.0
// Version dated 6.2.95
// Revision 1 dated 10.2.95: change of name in WMKTypes.h, because NT WTypes and WKTypes are used

#ifndef WMKTYPES_H_INTERN // lock
#define WMKTYPES_H_INTERN

#if defined (NT_FUNCTIONS_INCLUDED)
#define WENTRY_C pascal
// #define WENTRY_C __stdcall
#define W_POINTER *
#elif defined (OS2_FUNCTIONS_INCLUDED)
#if defined (OS2_32BIT)
#if defined (__BORLANDC__)
#define WENTRY_C pascal
#elif defined (__IBMC__) || defined (__IBMCPP__)
#define WENTRY_C _Pascal
#elif defined (__WATCOMC__)
#define WENTRY_C _Pascal
#else
#define WENTRY_C pascal
#endif
#define W_POINTER *
#else
#define WENTRY_C pascal far
#define W_POINTER far *
#endif
#elif defined (WINDOWS_FUNCTIONS_INCLUDED)
#if defined (__32BIT__)
#define WENTRY_C pascal
#define W_POINTER *
#else
#define WENTRY_C _loadds pascal far
#define W_POINTER far *
#endif
#elif defined (WIN95_FUNCTIONS_INCLUDED)
#define WENTRY_C __stdcall
#define W_POINTER *
#else // Dos
#define WENTRY_C _loadds pascal far
#define W_POINTER far *
#endif

#endif // WMKTYPES_H_INTERN
```


6 Examples

6.1 Simple H1 transmission example	6-1
6.2 PLC reception example using a parameter file	6-3
6.2.1 Source listing	6-3
6.2.2 Source diagrams	6-5
6.2.3 The respective PLC program	6-8
6.2.4 The respective CP143plus parameter settings	6-9
6.2.5 The respective parameter file Net.net	6-10

6 Examples

6.1 Simple H1 transmission example

This program establishes a connection to another system with the address 080006010001 and a TSAP (own and remote) 20202020202020 with priority 2, Normal, Aktiv. This connection is then used to send 100 bytes of 0's.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <H1Def.h>
#pragma optimize("",off)
int dummy(int dummyvalue)
{
    return dummyvalue * 3;
}
void delay(void)
{
    unsigned long wait;
    for (wait = 600000L ; wait ; wait--)
    {
        dummy(1);
    }
}
#pragma optimize("",on)
int main(int argn,char *argv[])
{
    int num,err;
    H1_CONNECT_PARAMS cr;
    H1_SENDFPARAMS *sp;
    printf("H1 Testprogram. (C) VIPA, W. Krings\n");
    if (argn > 1)
    {
        num = atoi(argv[1]);
        if (num) // Vector
        {
            printf("Softint %X, -> press any key to continue ..",num);
            H1SetVector(num);
            getch();
            printf("\n");
        }
    }
    cr.CrParams.Priority = PRIORITY_2;
    cr.CrParams.ConnectType = NORMAL_LINE;
    cr.CrParams.LenDestAddr = 6;
    cr.CrParams.DestAddr[0] = 8;
    cr.CrParams.DestAddr[1] = 0;
    cr.CrParams.DestAddr[2] = 6;
    cr.CrParams.DestAddr[3] = 1;
    cr.CrParams.DestAddr[4] = 1;
    cr.CrParams.DestAddr[5] = 2;
    cr.CrParams.LenNSAP = 0;
    cr.CrParams.LenDestTSAP = 8;
    strcpy(cr.CrParams.DestTSAP,"...");
    cr.CrParams.LenOwnTSAP = 8;
    strcpy(cr.CrParams.OwnTSAP,"...");
    cr.CrParams.LenConnParams = 0;
    if (sp = (H1_SENDFPARAMS *)malloc(sizeof(H1_SENDFPARAMS) + 100))
    {
        if (!(err = H1StartConnect(&cr)))
        {
```

```

printf("Connection started, err %d\n",cr.Fehler);
delay();
if (!cr.Fehler)
{
    sp->Vnr = cr.Vnr;
    sp->DataLen = 100;
    memset(sp->Daten,0,100);
    printf("Start send now ..");
    if (1) // POLL mode or wait
    {
        if (!(err = H1StartSend(sp)))
        {
            if (sp->Fehler != H1_WAIT_CONNECT)
            {
                printf("Send started, err %d\n",err);
                delay();
                for (num = 0 ; num < 10 ; num++ , delay())
                {
                    H1CheckSend(sp);
                    if (!sp->Fehler)
                    {
                        printf("Data was sent\n");
                        break;
                    }
                    else
                    {
                        printf("Check send, err %d\n",sp->Fehler);
                    }
                }
            }
            else
            {
                printf("No connection, err %d\n",sp->Fehler);
            }
        }
        else
        {
            printf("Start read, err %d\n",err);
        }
    }
    else
    {
        err = H1SendData(sp);
        printf("Send ret %d, H1err %d\n",err,sp->Fehler);
    }
    printf("End, err %d\n",err);
    H1StopConnect(cr.Vnr);
}
else
{
    printf("CR error, err %d\n",err);
}
}
else
{
    printf("H1 Drv not available or bad interrupt vector, err %d\n",err);
}
free(sp);
}
return 0;
return 0;
}

```

6.2 PLC reception example using a parameter file

This program retrieves H1 connection parameters from the file C:\Net.net. Then it starts the first connection in the file.

The program then proceeds to read a word via the connection from the selected PLC from DB10-DW11.

6.2.1 Source listing

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <fcntl.h>
#include <share.h>
#include <malloc.h>
#include <NetFile.h>
#include <S5Access.h>
#pragma optimize("",off)
int dummy(int dummyvalue)
{
    return dummyvalue * 3;
}
void delay(void)
{
    unsigned long wait;
    for (wait = 600000L ; wait ; wait--)
    {
        dummy(1);
    }
}
#pragma optimize("",on)
int main(int argn,char *argv[])
{
    char *names,*ptr;
    int num,err;
    H1_CONNECT_PARAMS cr;
    S5_VERBINDUNGSDATEN s5daten;
    S5_PARAMS s5;
    int s5fehler;
    printf("H1 AP Library test program. (C) WKS\n/s = send, else AP read\n");
    if (argn > 1)
    {
        num = atoi(argv[1]);
        if (num) // Vector
        {
            printf("Softint %X, -> any key to continue ..",num);
            S5SetVector(num);
            getch();
            printf("\n");
        }
    }
    if ((names = (char *)malloc(5000)) != 0)
    {
        if (!(err = S5ListConnections(5000,names)))
        {
            for (ptr = names , num = 0 ; *ptr ; ptr += strlen(ptr) + 1)
            {
                printf("[%s]\n",ptr);
                num++;
            }
        }
        else
        {
            printf("No connection, err %d\n",err);
        }
        strcpy(s5daten.Verbindungsname,names);
        free(names);
    }
    else
    {
        s5daten.Verbindungsname[0] = 0;
    }
    if (s5daten.Verbindungsname[0]) // is a name available?
    {
        if (!(err = S5GetConnectionParameter(&s5daten)))
        {
            memcpy(&cr,&s5daten.CrParams,sizeof(CONNECT_PARAMS));
            printf("Connection parameter [%s] da\n",s5daten.Verbindungsname);
            printf("Adr len %d %X %X %X %X %X\n",
                cr.CrParams.LenDestAddr, cr.CrParams.DestAddr[0], cr.CrParams.DestAddr[1], cr.CrParams.DestAddr[2],
                cr.CrParams.DestAddr[3], cr.CrParams.DestAddr[4],cr.CrParams.DestAddr[5]);
            if (!(err = S5StartConnection(&cr)))
            {

```

```

printf("Connection started, err %d\n",cr.Fehler);
delay();
if (!cr.Fehler)
{
    s5.Verbindungsnummer = cr.Vnr;
    s5.Kennung = 1; // DB DW
    s5.DB = 10;
    s5.DW = 11;
    s5.Len = 1;
    printf("Start read ..");
    if (!(err = S5StartRead(&s5)))
    {
        printf("Read function started, err %d\n",err);
        delay();
        for (num = 0 ; num < 10 ; num++ , delay())
        {
            if (!(err = S5PollRead(100,names,&s5fehler,cr.Vnr)))
            {
                printf("PLC data available, AP-Err %d, daten %X\n",s5fehler,*names);
                break;
            }
            else
            {
                delay();
                printf("Read was polled, err %d\n",err);
            }
        }
    }
    else
    {
        printf("Start read, err %d\n",err);
    }
    printf("End, err %d\n",err);
    S5StopConnection(cr.Vnr);
}
else
{
    printf("CR error, err %d\n",err);
}
}
else
{
    printf("H1 Drv ? , err %d\n",err);
}
}
else
{
    printf("No connection parameters, err %d\n",err);
}
}
else
{
    printf("Connection not available, err %d\n",err);
}
return 0;
}

```

6.2.2 Source diagrams

Fetch-Write in general

These parameters are fixed.

[illegible]

```

1
main for Send

int main(int argn,char *argv[])
{
    int num,err;
    H1_CONNECT_PARAMS LINE cr;
    H1_SENDFPARAMS *sp;
    time_t ref,act;

    printf("H1 test program for multiple adapters. (C) W. Krings\n");

    #if !defined (NT_FUNCTIONS_INCLUDED) && !defined (OS2_FUNCTIONS_INCLUDED)

    if (argn > 1)
    {
        num = atoi(argv[1]);

        if (num) // Vector
        {
            printf("Softint %X, -> Any key continues ..",num);
            H1SetVector(num);
            printf("\n");
        }
        #endif

        CrParameterVorbesetzen(&cr);

        if (sp = (H1_SENDFPARAMS *)malloc(sizeof(H1_SENDFPARAMS) + 100))
        {
            if (!H1DriverOpen())
            {
                if (! (err = H1StartConnectCard(&cr)))
                {
                    signal(SIGINT,BreakCode);
                    printf("Connection started,   err %d\n",cr.Fehler);
                    time(&ref);
                    act = ref;
                    ref += 3; // wait a max. of 3 s

                    if (!cr.Fehler)
                    {
                        sp->Vnr = cr.Vnr;
                        sp->DataLen = 100;
                        memset(sp->Daten,0,100);
                        printf("Start send ..");

                        if (POLLMODUS) // POLL mode or wait
                        {
                            while (act < ref) // Wait no longer than the preset time
                            {
                                time(&act);

                                if (! (err = H1StartSend(sp)))
                                {
                                    if (sp->Fehler != H1_WAIT_CONNECT)
                                    {
                                        printf("Send started, err %d\n",err);
                                        while (act < ref) // Wait no longer than the preset time
                                        {
                                            time(&act);
                                            H1CheckSend(sp);

                                            if (!sp->Fehler)
                                            {
                                                printf("Data sent\n");
                                                break;
                                            }
                                            else
                                            {
                                                printf("Send p
oll
ed, err %d\n",sp->Fehler);
                                            }
                                        }
                                    }
                                    else
                                    {
                                        printf("No
conn
ecti
on,
err
%d\n",
sp->Fehler);
                                    }
                                }
                                else
                                {
                                    printf("Read
start
, err
%d\n",err);
                                }
                            }
                        }
                        else
                        {
                            err = H1SendData(sp);

                            printf("Send r
et %d,
H1err %d\n",err,sp->Fehler);
                        }
                    }
                    else
                    {
                        printf("No d
rive
r\n");
                    }
                }
            }
        }
    }
    free(sp);

    return 0;
}

```


Fetch Aktiv

```

2
main Read
int main(int argn,char *argv[])
{
    int num,err;
    H1_CONNECT_PARAMS_LINE cr;
    H1_RECPARAMS *rp;
    time_t ref,act;

    printf("H1 Test program Read for multiple adapters. (C) W. Krings\n");

    #if !defined (NT_FUNCTIONS_INCLUDED) && !defined (OS2_FUNCTIONS_INCLUDED)

    if (argn > 1)
    {
        num = atoi(argv[1]);
        if (num) // Vector
        {
            printf("Softint %X, -> Any key continues ..",num);
            H1SetVector(num);
            printf("\n");
        }
        #endif

        CrParameterVorbeseetzen(&cr); //assign Cr parameters

        if (rp = (H1_RECPARAMS *)malloc(sizeof(H1_RECPARAMS) + 512))
        {
            if (!H1DriverOpen())
            {
                if (!H1StartConnectCard(&cr))
                {
                    signal(SIGINT,BreakCode);
                    printf("Connection started err %d\n",cr.Fehler);
                    time(&ref);
                    act = ref;
                    ref += 3; // wait a max. of 3 s

                    if (!cr.Fehler)
                    {
                        rp->Vnr = cr.Vnr;
                        rp->DataLen = 512;
                        printf("Satrt read ..");

                        if (POLLMODUS) // POLL mode or wait
                        {
                            while (act < ref) // Wait no longer than the preset time
                            {
                                time(&act);
                                if (!H1StartRead(rp))
                                {
                                    if (rp->Fehler != H1_WAIT_CONNECT)
                                    {
                                        printf("Read started, err %d\n",err);
                                        while (act < ref) // Wait no longer than the preset time
                                        {
                                            time(&act);
                                            H1CheckRead(rp);
                                            if (!rp->Fehler)
                                            {
                                                printf("%d Data available (%X %X %X %X)\n",rp->RecLen,rp->Daten[0],rp->Daten[1],rp->Daten[2],rp->Daten[3]);
                                                break;
                                            }
                                            else
                                            {
                                                printf("Polled read, err %d\n",rp->Fehler);
                                            }
                                        }
                                    }
                                    else
                                    {
                                        printf("Star read\n",err);
                                    }
                                }
                                else
                                {
                                    printf("No connection\n",err);
                                }
                            }
                        }
                        else
                        {
                            err = H1ReadData(rp);
                            printf("Read r et %d, Hlerr %d\n",err,rp->Fehler);
                        }
                    }
                    else
                    {
                        printf("bad integer error vector, err %d\n",err);
                    }
                }
            }
            else
            {
                printf("No driver\n"),
            }
        }
    }

    free(rp);
    return 0;
}

```

6.2.3 The respective PLC program

The following PLC program is required when a PC-based program should be controlled from a remote PLC.

The following example is written for a PLC 135U. On other PLCs (115U, 155U) you must use the appropriate handler modules.

```

AG 115U :   FB 244  with A-NR 0  for Sendall
           FB 245  with A-NR 0  for Recall

AG 155U :   FB 120  with A-NR 0  for Sendall
           FB 121  with A-NR 0  for Recall

      BAUSTEIN#OB1
      BIB      #17025
;
;  Cycle- OB : Called from Send_All and Receive_All   (AG 135U)
;
;
0
00002      :
00004      :
00006      :SPA FB 126
           NAME #SEND-A
           SSNR =KY 0,0                SSNR (tile)  0
           A-NR =KY 0,0
           ANZW =MW 180
           PAFE =MB 198
00012      :
00014      :
00016      :SPA FB 127
           NAME #REC-A
           SSNR =KY 0,0                SSNR (tile)  0
           A-NR =KY 0,0
           ANZW =MW 184
           PAFE =MB 197
00022      :
00024      :
00026      :A   DB 10                  userdata-DB
00028      :
0002A      :L   DW 11                  read data
0002C      :L   KB 1
0002E      :+F                          increment
00030      :T   DW 11
00032      :
00034      :BE

      BAUSTEIN#OB20
      BIB      #17025
;
;  Restart-OB : interface synchronization
;
;      this synchronization (FB125-call) should also be invoked
;      for a manual restart (OB 21) and is mandatory when power returns
;      (OB 22) after a failure
;
;
00000      :
00002      :
00004      :SPA FB 125
           NAME #SYNCHRON
           SSNR =KY 0,0                SSNR (tile)  0
           BLGR =KY 0,6                block size 6  (512 Byte)
           PAFE =MB 199
0000E      :
00010      :BE

      BAUSTEIN#DB10
      BIB      #17025
;
;  userdata-DB  length 50 words
;
;
00000:      50 (
00000:      KH      = 0000
00001:      )

```

6.2.4 The respective CP143plus parameter settings

Own station parameters:

```
Own ethernet address      :      080006010000    (corresponds to EthernetStation of Net.net)
Tile address              :      0                (corresponds to the SSNR in the PLC prog)
```

Connection parameters

The following connections should be configured on the CP143(plus) to enable the exchange of data:

```
Remote ethernet address  :      080006010001
Remote TSAP              :      3131313131313131    (corresponds to the own TSAP of Net.net)
Own TSAP                 :      3131313131313131    (corresponds to the remote TSAP of Net.net)
Task type                :      FetchPass
Task number              :      82
Tile offset              :      0
Priority                  :      Prio2
Ethernet task type       :      Normal,Passiv
Multicast circuit number :      0
Automatic start          :      Yes

Remote Ethernetadresse   :      080006010001
Remote TSAP              :      3232323232323232    (corresponds to the own TSAP of Net.net)
Own TSAP                 :      3232323232323232    (corresponds to the remote TSAP of Net.net)
Task type                :      SendPass
Task number              :      83
Tile offset              :      0
Priority                  :      Prio2
Ethernet task type       :      Normal,Passiv
Multicast circuit number :      0
Automatic start          :      Yes
```

Explanation of remote ethernet address:

This comprises the ethernet address of the CP1413plus adapter or the ethernet address as defined in the file "PROTOCOL.INI".

6.2.5 The respective parameter file Net .net

;Active connection (for Fetch)

[Test Verbindung Fetch]

Leitungstyp=Ethernet

EthernetKarte=1

EthernetStation=080006010777 {please enter the ethernet address}

{of your PC. }

EthernetNSAP=

EthernetEigenerTSAP=3131313131313131

EthernetFremderTSAP=3131313131313131

EthernetStartParameter=

EthernetLeistungsart=Normal,Aktiv

EthernetPriorität=Prio2

EthernetMulticastkreis=0

;Aktive connection (for Write)

[Test Verbindung Write]

Leitungstyp=Ethernet

EthernetKarte=1

EthernetStation=080006010777 {please enter the ethernet address}

{of your PC. }

EthernetNSAP=

EthernetEigenerTSAP=3232323232323232

EthernetFremderTSAP=3232323232323232

EthernetStartParameter=

EthernetLeistungsart=Normal,Aktiv

EthernetPriorität=Prio2

EthernetMulticastkreis=0

7 H1 Driver V3.xx for Windows NT

7.1 General	7-1
7.2 Uninstalling older Windows NT drivers	7-1
7.3 Installation of the VIPA CP1413plus network adapter	7-1
7.4 Installation of the H1 driver V3.xx for Windows NT 3.51	7-2
7.4.1 Installation of SSN-BG88 (Z'nyx adapter) for PCI bus	7-2
7.4.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus	7-3
7.5 Installation of the H1 driver V3.xx for Windows NT 4.0	7-4
7.5.1 Installation of SSN-BG88 (Z'NYX adapter) for PCI-Bus	7-4
7.5.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus	7-5
7.5.3 Extensions to the configuration options	7-6
7.6 Programming	7-7
7.6.1 General notes on programming	7-7
7.6.2 H1 Layer 4 programming interface	7-21
7.6.3 PLC Layer 7 software interface	7-55
7.6.4 PLC Net file functions	7-85
7.6.5 Files, constants and structures	7-97

7 H1 Driver V3.xx for Windows NT

7.1 General

When the adapter is used in an H1 environment under Windows NT 3.51 or 4.0 it is recommended that the H1 driver V3.xx is employed.

This driver is supplied on the floppy disk SSN-SW83N which is included.

A new H1 driver for the CP1413plus-H1-PC-adapter is accompanied by a set of revised DLLs.

The libraries H1DLLNT.DLL and S5DLLNT.DLL for the new driver underwent extensive expansions and additions. Existing function names were not altered but new functions for asynchronous processing were added.

Parameters and structures have been changed in certain instances and in some cases they may differ vastly from the old ones. These changes were required to ensure a more efficient and consistent concept.



Any existing H1 driver must be uninstalled before the new H1 driver V 3.xx can be installed under Windows NT!

7.2 Uninstalling older Windows NT drivers

If an earlier version of the H1 driver is installed on your system, this must be deactivated and removed. For WinNT3.5x you must deactivate the driver by selecting "Device" and setting the H1 protocol to "Deactivated". The driver can now be uninstalled by deleting the "H1Prot" entry from HKEY_LOCAL_MACHINE-SYSTEM-CurrentControlSet-Services in the WinNT-Registry (regedt32.exe).

We recommend that you search your Windows directory for earlier versions of the driver and DLLs and that you remove these as well.

7.3 Installation of the VIPA CP1413plus network adapter

VIPA offers two different network adapters to cater for the different PC bus systems:

- for the PCI bus: Z'nyx adapter (order no.: VIPA SSN-BG88)
Description: see chapter 3.1
Hardware installation: see chapter 3.1.3
- for the ISA bus: 3COM adapter (order no.: VIPA SSN-BG85C)
Description: see chapter 3.2
Hardware installation: see chapter 3.2.3

7.4 Installation of the H1 driver V3.xx for Windows NT 3.51

7.4.1 Installation of SSN-BG88 (Z'nyx adapter) for PCI bus

For first time installations you must complete steps 1 through 3. If you are installing the H1 driver for an existing SSN-BG88 (Z'nyx adapter) you only need to complete step 3.

Step 1

Please refer to chapter 3.1.3 for instructions on installing the hardware.

Note You may have to change the configuration. We recommend that the address is set to 340h if this is still available.

Step 2

Install the driver for the network adapter as follows:

- Start Windows NT 3.51 and click on "control panel" in the main group. Select "Network" in the control panel. A dialogue box containing the current configuration is displayed.
- click on "Network adapter" and then on "Add" to install the new network adapter.
- Select the following driver from the displayed list
"DEC PCI Ethernet DECChip 21040".
- Click on "Configuration" and enter the *address* and the *interrupt*.
- Once you confirm the settings your network adapter is installed.

Step 3

Install the H1 driver as follows:

- Start Windows NT 3.51 and click on "Control panel" in the main group. Select "Network" in the control panel. A dialogue box containing the current configuration is displayed.
- Click on "Software" to install the driver. A list containing the available drivers will be displayed.
- Select "<other> Have disk" and insert the disk SSN-SW83N.
- The H1 driver is displayed. Select this driver.
- The driver is installed when you confirm your entries.
- You will be asked to reboot the computer.

7.4.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus

If this is a first time installation you must complete steps 1 through 3. If you are installing the H1 driver for an existing SSN-BG85C (3COM adapter) you only need to complete step 3.

Step 1

Please refer to chapter 3.2.3 for instructions on installing the hardware.

Note You may have to change the configuration. We recommend that the address is set to 340h if this is still available.

Step 2

Install the driver for the network adapter as follows:

- Start Windows NT 3.51 and click on "Control panel" in the main group. Select "Network" in the control panel. A dialogue box containing the current configuration is displayed.
- Click on "Network adapter" and then on "Add" to install the new network adapter.
- Select the following driver from the displayed list
"3COM Ethernetlink III ISA/PCMCIA-Adapter".
- Click on "Configuration" and enter the *address* and the *interrupt*.
- Once you confirm the settings your network adapter is installed.

Step 3

Install the H1 driver as follows:

- Start Windows NT 3.51 and click on "Control panel" in the main group. Select "Network" in the control panel. A dialogue box containing the current configuration is displayed.
- Click on "Software" to install the driver. A list containing the available drivers will be displayed.
- Select "<other> Have disk" and insert the disk SSN-SW83N.
- The H1 driver is displayed. Select this driver.
- The driver is installed when you confirm your entries.
- You will be asked to reboot the computer.

7.5 Installation of the H1 driver V3.xx for Windows NT 4.0

7.5.1 Installation of SSN-BG88 (Z'NYX adapter) for PCI-Bus

If this is a first time installation you must complete steps 1 through 3. If you are installing the H1 driver for an existing SSN-BG88 (Z'NYX adapter) you only need to complete step 3.

Step 1

Please refer to chapter 3.1.3 for instructions on installing the hardware.

Note You may have to change the configuration. We recommend that the address is set to 340h if this is still available.

Step 2

Install the driver for the network adapter as follows:

- Start Windows NT 4.0.
- Right-click on the "Network Neighbourhood" icon. Click on "Properties" in the menu. This displays a menu with tabs for the various settings.
- Click on the "Network adapter" tab and then on "Add" to install the new network adapter. A list of available drivers will be displayed.
- Select the following driver from the displayed list
"DEC PCI Ethernet DECChip 21040".
- Click on "Properties" and place a tick in "Auto detect".
- Once you confirm the settings your network adapter is installed.

Step 3

Install the H1 driver as follows:

- Right-click on the "Network Neighbourhood" icon. Click on "Properties" in the menu. This displays a menu with tabs for the various settings.
- Click on "Service" and "Add" to install the driver. A list containing the available protocols will be displayed.
- Insert the disk SSN-SW83N and click on "Disk".
- The H1 driver is displayed. Select this driver.
- The driver is installed when you confirm your entries.
- You will be asked to reboot the computer.

7.5.2 Installation of SSN-BG85C (3COM adapter) for ISA-Bus

If this is a first time installation you must complete steps 1 through 3. If you are installing the H1 driver for an existing SSN-BG85C (3COM adapter) you only need to complete step 3.

Step 1

Please refer to chapter 3.2.3 for instructions on installing the hardware.

Note You may have to change the configuration. We recommend that the address is set to 340h if this is still available.

Step 2

Install the driver for the network adapter as follows:

- Start Windows NT 4.0.
- Right-click on the "Network Neighbourhood" icon. Click on "Properties" in the menu. This displays a menu with tabs for the various settings.
- Click on the "Network adapter" tab and then on "Add" to install the new network adapter. A list of available drivers will be displayed.
- Select the following driver from the displayed list
"3COM Ethernetlink III ISA/PCMCIA-Adapter".
- Click on "Properties" and place a tick in "Auto detect".
- Once you confirm the settings your network adapter is installed.

Step 3

Install the H1 driver as follows:

- Right-click on the "Network Neighbourhood" icon. Click on "Properties" in the menu. This displays a menu with tabs for the various settings.
- Click on "Service" and "Add" to install the driver. A list containing the available drivers will be displayed.
- Insert the disk SSN-SW83N and click on "Disk".
- The H1 driver is displayed. Select this driver.
- The driver is installed when you confirm your entries.
- You will be asked to reboot the computer.

7.5.3 Extensions to the configuration options

The Windows NT 4.0 system provides additional configuration options for the H1 driver. It is possible to specify filters, limit the maximum number of connections and to define the H1 stack for use by multiple network adapters.

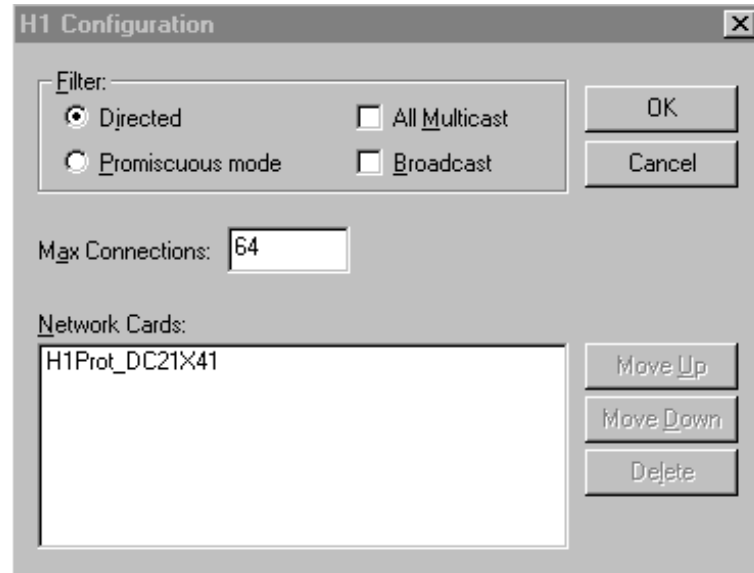


Fig. 7-1: H1 driver configuration for Windows NT 4.0

You can open the "H1-Configuration" by clicking on *Settings / Control Panel* and then on *Network*. Here you click on *Properties*. A tabbed menu will be displayed. Click on the tab for *Service* and select "H1 protocol".

Click on *Properties* to open the H1 configuration window shown above.

Parameter

Filter

Here you can specify which messages should be rejected by the filter.

The default setting for both VIPA PC network adapters is **Promiscuous Mode**.

Promiscuous Mode operation is necessary when you wish to alter the station address. Please remember, that the Promiscuous Mode places severe demands on the system, since every message must be received and decoded. If the loading of your system is excessive and you do not need to change the station address you should select **Directed**.

In **Directed** mode you can use option fields to specify that broadcast or multicast messages are rejected by the filter. If the respective field is ticked, the messages are rejected.

Max Connections

This defines the maximum number of connections.

Network Cards

When more than one network adapter is used you can specify for which specific card the H1 stack must be used.

7.6 Programming

7.6.1 General notes on programming

Layer 4

Layer

7	APPLICATION
6	PRESENTATION
5	SESSION
4	TRANSPORT
3	NETWORK
2	DATA LINK
1	PHYSICAL

H1DriverOpen
 H1DriverClose
 H1HoleVersion
 H1StarteVerbindung
 H1StarteVerbindungOverlapped
 H1StoppeVerbindung
 H1StoppeVerbindungOverlapped
 H1StoppeVerbindungen
 H1StoppeVerbindungenOverlapped
 H1HoleStationsAdresse
 H1SetzeStationsAdresse
 H1HoleStandardwerte
 H1SetzeStandardwerte
 H1SendeDaten
 H1SendeDatenEx
 H1StarteSenden
 H1StarteSendenOverlapped
 H1StarteSendenExOverlapped
 H1AbfrageSenden
 H1LeseDaten
 H1LeseDatenEx
 H1StarteLesen
 H1StarteLesenOverlapped
 H1StarteLesenEx
 H1StarteLesenExOverlapped
 H1AbfrageLesen
 H1AbfrageLesenEx
 H1GetOverlappedResult

The Layer 4 functions correspond to the "SEND DIREKT" and "RECEIVE DIREKT" functions of the VIPA CP143 plus PLC interface.

Layer 7 (Application)

Layer

7	APPLICATION
6	PRESENTATION
5	SESSION
4	TRANSPORT
3	NETWORK
2	DATA LINK
1	PHYSICAL

S5StarteVerbindung
S5StoppeVerbindung
S5StoppeVerbindungen
S5HoleRevision
S5HoleStationsAdresse
S5SetzeStationsAdresse
S5LeseAusSPS
S5LeseAusSPSOverlapped
S5StarteLesen
S5StarteLesenOverlapped
S5AbfrageLesen
S5AbfrageLesenOverlapped
S5SchreibeInSPS
S5SchreibeInSPSOverlapped
S5StartSchreiben
S5StartSchreibenOverlapped
S5AbfrageSchreiben
S5AbfrageSchreibenOverlapped
S5FetchPassiv
S5WritePassiv

The Layer 7 functions correspond to the "FETCH AKTIV" and "WRITE AKTIV" functions of the VIPA CP143 plus PLC interface.

7.6.1.1 Overlapped - functions

Windows NT provides you with asynchronous mechanisms. The H1 driver for Windows NT supports these mechanisms. You can use function calls to initiate asynchronous H1 and PLC tasks.

7.6.1.1.1 H1 overlapped functions

The required overlapped functions are provided by the DLL file `H1DLLNT.DLL`. overlapped functions return control to the system immediately after they have been initiated. This means that your application will continue operating while the driver processes the overlapped function. In principle this special Windows NT mechanism operates like the start and check functions.

Windows NT provides the API functions **WaitForSingleObject**, **WaitForMultipleObject** or the function **H1GetOverlappedResult** for checking overlapped functions.

The programming conditions shown below must be met before you can use any one of these functions: **H1StarteVerbindungOverlapped**, **H1StoppeVerbindungOverlapped**, **H1StoppeVerbindungenOverlapped**, **H1StarteSendenOverlapped**, **H1StarteSendenExOverlapped**, **H1StarteLesenOverlapped**, **H1StarteLesenExOverlapped**, **H1GetOverlappedResult**.

1. Create a structure of the type *overlapped*.
2. Create an event and pass the resulting event handle to the overlapped structure.
3. Execute the call to the H1 overlapped function.
4. You can check whether the H1 overlapped function has terminated by means of **H1GetOverlappedResult** or a **WaitForSingleObject** or **WaitForMultipleObject** function.

Overlapped functions should be used when it is necessary that other tasks are completed while an H1 overlapped function is processed asynchronously.

Note: The disk containing the drivers also contains modules with calls to overlapped functions. These you may use as a reference for your own applications.

7.6.1.1.2 PLC overlapped functions

When an PLC overlapped function is executed the parameters `UserFctCall` and `Timeout_ms` are transferred. The following function diagram is included to explain these parameters in conjunction with **S5LeseAusSPSOverlapped**.

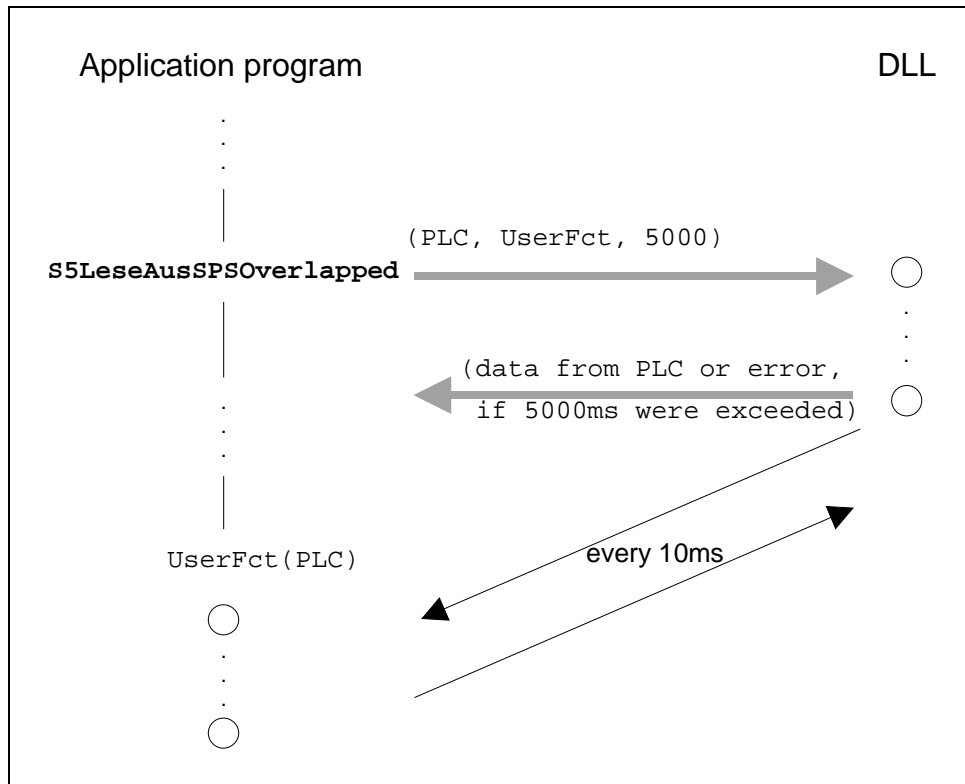


Fig. 7-2: Function diagram PLC overlapped functions

Once the function call has been initiated the application program jumps to the respective function in the DLL passing, amongst others, the parameters `UserFctCall` (`UserFct`) and `Timeout_ms` to the DLL. While the DLL processes the function a call to `UserFct` is executed at a fixed rate of 10ms. Thus, control is returned to your program in 10ms intervals while the overlapped function is active.

`Timeout_ms` is another parameter that must be passed along with the function call. Here you specify the time in which the DLL function must be completed. If this time is exceeded the connection is terminated implicitly and a return accompanied by an error message is executed.

7.6.1.2 Expedited functions

Expedited functions are those functions that process priority or urgent data. The length of data must not exceed 16 bytes.

These functions may be compared to the datagram services that operate without acknowledgement to provide fast processing of messages.

7.6.1.3 Short overview function - procedure

This overview contains a list of functions along with the steps required for their configuration.

The table contains values for **Priority**, **ConnectType**, **LenDestAddr**, etc. that are only provided as examples and should be adapted to the respective application.

Entering the connection

PC-Side	CP function	PLC side
<u>in structure</u> CONNECT_PARAMS 1. Priority = <i>PRIORITY_2</i> 2. ConnectType = <i>NORMAL_LINE</i> / <i>ACTIVE_LINE</i> (depending on the type of connection, may also be <i>NORMAL_LINE</i> / <i>PASSIVE_LINE</i>) 3. LenDestAddr = 6 (bytes) 4. DestAddr = <i>0020d582000f</i> (ethernet address external station) 5. LenDestTSAP = 8 6. DestTSAP (<i>externalTsap</i>) = <i>22334455</i> (stations are crossed over) 7. LenOwnTsap = 8 8. OwnTsap (own Tsap) = <i>11223344</i> (stations are crossed over) 9. Vnr (connection number) contains the number of the connection and the number of the adapter (0-3)	none	none

Tab. 7-1: Procedure to enter the connection

Layer 4: PLC transmits data to the PC

PC side	CP function	PLC side
<u>RECEIVE</u> H1LeseDaten() H1StarteLesen() H1AbfrageLesen() <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE</i> / <i>PASSIVE_LINE</i>	H1 connection Send	FB Send FB Control (alternative) FB SendAll (cyclic)

Tab. 7-2: Procedure PLC transmits data to the PC

Layer 4: PC transfers data to the PLC

PC side	CP function	PLC side
<u>SEND</u> H1SendeDaten() H1StarteSenden() H1AbfrageSenden() <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE</i> / <i>ACTIVE_LINE</i>	H1-Verbindung Receive	FB Receive FB Control (alternative) FB ReceiveAll (cyclic)

Tab. 7-3: Procedure PC transfers data to PLC

Layer 4 Dual connection: use a common TSAP to write data into or to read data from the PLC

PC side	CP function	PLC side
2 tasks must be configured:		
<u>SEND</u> H1SendeDaten() H1StarteSenden() H1AbfrageSenden() <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE / ACTIVE_LINE</i>	<u>H1 connection</u> Main connection: Receive Multi-connection: Send	FB Receive FB Control (alternative) FB ReceiveAll (cyclic)
<u>RECEIVE</u> H1LeseDaten() H1StarteLesen() H1AbfrageLesen() <u>in structure</u> CONNECT_PARAMS ConnectTyp = <i>NORMAL_LINE / ACTIVE_LINE</i>	<u>H1-Verbindung</u> Main-connection Send Multi-connection Receive	FB Send FB Control (alternative) FB SendAll (cyclic)

Tab. 7-4: Procedure for a dual connection

Layer 7: PC reads data from the PLC (Fetch)

PC side	CP function	PLC side
<u>FETCH</u> S5LeseAusSPS () S5StarteLesen () S5AbfrageLesen () <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE / AKTIVE_LINE</i>	Fetch Passive	FB SendAll (cyclic) FB ReceiveAll (cyclic)

Tab. 7-5: Procedure read data from PLC

Layer 7: PC transfers data into the PLC (Write)

PC side	CP function	PLC side
<u>WRITE</u> S5SchreibeInSPS () S5StartSchreiben () S5AbfrageSchreiben () <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE / AKTIVE_LINE</i>	Write Passive	FB SendAll (cyclic) FB ReceiveAll (cyclic)

Tab. 7-6: Procedure transfer data into PLC

Layer 7: PLC transfers data to PC

PC side	CP function	PLC side
<u>WRITE</u> S5WritePassiv() <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE / PASSIV_LINE</i>	Write active	FB Send FB SendAll (cyclic) FB ReceiveAll (cyclic)

Tab. 7-7: Procedure PLC transfers data to PC

Layer 7: PLC reads data from PC

PC side	CP function	PLC side
<u>FETCH</u> S5FetchPassiv() <u>in structure</u> CONNECT_PARAMS ConnectType = <i>NORMAL_LINE / PASSIV_LINE</i>	Fetch active	FB Fetch FB SendAll (cyclic) FB ReceiveAll (cyclic)

Tab. 7-8: Procedure PLC reads data from PC

7.6.1.4 Explanation of H1 parameters

Extract from H1Def.h

```
typedef struct _CONNECT_PARAMS
{
    UCHAR Priority;           // Priority
    UCHAR ConnectType;       // Connection type
    USHORT Lsap;             // LinkServiceAccessPoint
    UCHAR LenDestAddr;       // Length of destination address
    UCHAR DestAddr[6];       // Destination address
    UCHAR Multicast;         // Multicast circuit number
    UCHAR LenNSAP;           // Length destination NSAP_ID
    UCHAR NSAP[12];          // Destination NSAP-ID
    UCHAR LenDestTSAP;       // Length destination TSAP_ID
    UCHAR DestTSAP[16];      // Destination TSAP-ID
    UCHAR LenOwnTSAP;        // Length own TSAP_ID
    UCHAR OwnTSAP[16];       // own TSAP-ID
    UCHAR LenConnParams;     // Length of additional connect parameters
    UCHAR ConnParams[16];    // Additional connect parameters
}CONNECT_PARAMS;
```

H1-Parameter

Priority

This parameter is included to provide compatibility with the NET.NET file functions. Any values in this parameter are ignored.

ConnectType

Here you must specify the type of connection.

The following definitions have been provided for the connection types:

NORMAL_LINE, DATAGRAMM_LINE, MULTICAST_LINE, BROADCAST_LINE,
PASSIVE_LINE, ACTIVE_LINE;

The following definitions may be combined by means of an OR function:

NORMAL_LINE | ACTIVE_LINE,
NORMAL_LINE | PASSIVE_LINE;

Active / Passive determines whether your own station establishes a connection **actively** or whether it **passively** waits for a connection. Please take care that you do not specify the same value for both sides of a connection.

Lsap

Lsap is used for datagram services and for time functions.

LenDestAddr

.This parameter is included to provide compatibility with the NET.NET file functions. Any values in this parameter are ignored..

DestAddr[6]

Parameter specifying the ethernet address of the destination station. Ethernet addressing parameters consist of binary data. You may specify any binary number from 0h to FFh. You define destination stations like PLC's by means of the ethernet address. Ethernet addresses have a length of 6 bytes. The first three characters define the manufacturer of the destination system. These bytes are allocated by the central IEEE committee. Where the manufacturers code has not been allocated it must be ensured that the first byte is even, i.e. it must not return a remainder if it is divided by two. The last three bytes may contain any arbitrary value. Ethernet addresses must be unique within a network.

Multicast

.This parameter is included to provide compatibility with the NET.NET file functions. Any values in this parameter are ignored.

LenNSAP

Length of the NSAP (Network Service Access Point)

NSAP

The NSAP (Network Service Access Point) determines the network number for linked networks. This parameter is rarely used as very few networks have the facilities to support it.

LenDestTSAP

Here you can define the length of the TSAP of the destination system. The length of a TSAP can be from 0 to 16 bytes. A length of 0 specifies that no TSAP is to be used.

DestTSAP

TSAP (Transport Service Access Point) of the destination system. The TSAP determines the address of the connection to the destination system. H1 TSAP's are equivalent to the TCP/IP ports. To establish a connection, the TSAP's of the local station and the destination station must be identical. This condition is easily met if you specify the same value for the local and the external TSAP for both directions of the connection.

LenOwnTSAP

This determines the length of the TSAP of the local station. The length of the TSAP may be from 0 to 16 bytes. A length of 0 specifies that no TSAP is to be used.

OwnTSAP

TSAP (Transport Service Access Point) of the local station. The TSAP determines the address of the connection in the local (own) system which should be used for the transfer of data.

LenConnParams

Here you can specify the length of additional connection data that should be transferred along with the respective messages.

ConParams

It is possible to transfer additional connection data when the connection is established. These bytes must be entered into **ConParams**. This parameter may be used to assist with system tasks, e.g. PG functionality or the protection of the connection by means of a password.

Connection parameter, error

Extract from H1Def.h

```
typedef struct
{
    CONNECT_PARAMS CrParams;           // Connection parameters
    CONNREF Vnr;                       // Adapter number/connection number
    unsigned short Fehler;              // Error code as connection is established
}H1_CONNECT_PARAMS;
```

CrParams

CrParams contains the structure and data shown above.

Vnr

A connection is identified by an adapter number and a connection number. The type of the **Vnr** parameter is CONNREF (defined in h1def.h) and it represents a 32-bit ULONG. The most significant word should contain the adapter number. The least significant word is used by the function **H1StarteVerbindung** to return the connection number. This connection number must be used for any future function calls that relate to this connection. The connection number may be released by the function **H1StoppeVerbindung**.

Fehler

"Fehler" (error) may contain an error message as described in H1 error messages if an attempt to establish a connection was unsuccessful.

No.	Message	Description
0	OK	no error occurred, command was completed
1	H1_BAD_CR_PARAMS	invalid CRParameter
2	H1_NO_SLOT	maximum number of connections established
3	H1_WAIT_CONNECT	connection was not established or interrupted
4	H1_NOT_IMPLEMENTED	function not implemented
5	H1_BAD_LINE	invalid connection number
6	H1_WAIT_DATA	no data available
7	H1_WAIT_SEND	data has not yet been sent
8	H1_INTERNAL_ERROR	internal error
10	H1_NO_DRIVER	no driver
12	H1_BLOCKED_DATA	blocked data was received
13	H1_NO_ADAPTER	specified adapter does not exist
15	H1_NOT_SUPPORTED	function not supported

Tab. 7-9: H1 error messages

7.6.1.5 H1 error messages

Any errors are returned via the respective structure (of the type H1_CONNECT_PARAMS, H1_SENDDPARAMS, H1_RECPARAMS) in the variable **Fehler** (error).

H1_BAD_CR_PARAMS	The connection parameters are invalid or bad.
H1_NO_SLOT	The maximum number of connections has already been started. You must terminate one of the existing connections before you start another connection.
H1_WAIT_CONNECT	This is not a true error message. It only indicates that a connection has been registered that has not yet been established.
H1_NOT_IMPLEMENTED	This message is returned if you attempt to access a function that has not been implemented on your system.
H1_BAD_LINE	This message may have more than one reason: <ul style="list-style-type: none">- the specified connection number is incorrect.- driver was never opened.- the respective connection no longer exists.
H1_WAIT_DATA	No data was received as yet. You can determine the status of a connection by means of H1AbfrageLesen .
H1_WAIT_SEND	Data has not been sent as yet. You can determine the status of a connection by means of H1AbfrageSenden .
H1_INTERNAL_ERROR	An error has occurred in the driver. Please contact the VIPA-Hotline.
H1_NO_DRIVER	The driver does not exist or it has not been entered correctly.
H1_BLOCKED_DATA	Blocked data was received. When the blocking is terminated a 0 (OK) is returned. You must set bit 15 in the connection number when you wish to transfer data from multiple non-contiguous memory locations.
H1_NO_ADAPTER	The following circumstances may have occurred: <ul style="list-style-type: none">- adapter does not exist or is damaged- driver does not exist or its settings are incorrect Check that the adapter number is correct (adapter 1 =0) Did the system recognise the adapter?
H1_NOT_SUPPORTED	The "Event Viewer" of Windows NT may not contain any entries. The specified function is not supported.

7.6.1.6 Defining the ethernet address

The ethernet address (6 bytes) consists of the following:

- bytes 1, 2, 3: 0x00 0x20 0xD5 (VIPA network identifier)
- bytes 4, 5, 6: these 3 bytes are located on the CP1413plus.

Z'NYX adapter

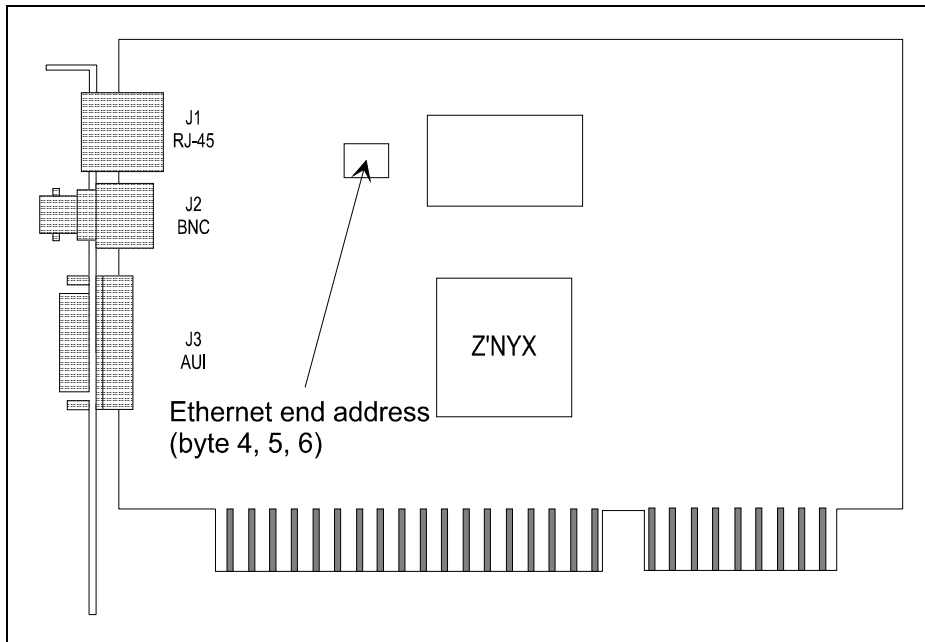


Fig. 7-3: The ethernet end address of the Z'NYX adapter (bytes 4, 5, 6)

3COM adapter

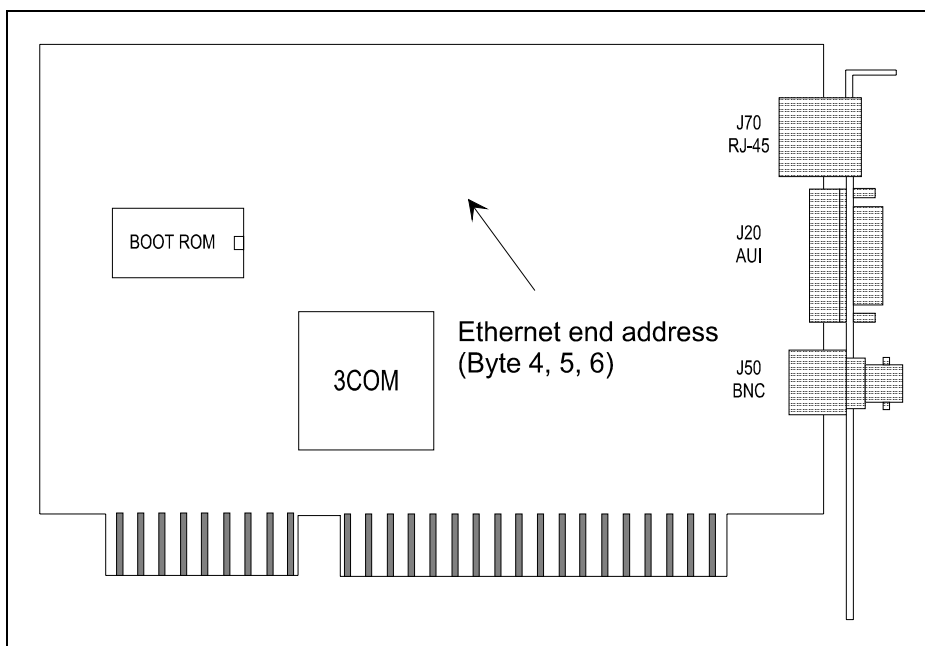


Fig. 7-4: The ethernet end address of the 3COM adapter (bytes 4, 5, 6)

7.6.2 H1 Layer 4 programming interface

7.6.2.1 General notes on the H1 programming interface

The H1 programming interface provides direct access to the functions of level 4 A of the ISO level model.

All the specified data is transferred "as is", i.e. no additional conversion occurs.

Before any station can be accessed a connection to the destination system must have been established. This connection can then be used to exchange data with the destination system.

It is possible to establish multiple connections to different or even to the same system.

When one of these connections is no longer required it should be terminated.

All the required functions are located in dynamic link libraries (.dll).

- H1DLLNT.DLL, S5DLLNT.DLL

The following export libraries are supplied to provide the required interfaces for linking with custom programs:

- H1DLLNT.LIB, S5DLLNT.LIB

These libraries were created and tested with VisualC++ 4.0.

The type of connection is determined by the connection parameters. For the CP143plus these parameters are defined on the PLC side by means of the parameter configuration program NCS.

7.6.2.2 General H1 layer 4 functions

The following applies to all functions of the layer 4 interface:

- Layer 4 interface functions have the prefix **H1**.
- The value returned by these functions is the value returned by the operating system for the function **DeviceIOCtrl**. Please refer to the file **H1DEF.H** for possible return values.
- It is possible to access layer 4 tasks directly from an application program running on the PC. All required structures are specified in the file **H1DEF.H**. Do not use any other structures. VIPA reserves the right to change this file if system expansions or error corrections are become necessary.

7.6.2.2.1 Driver open

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1DriverOpen(void)
```

The **H1DriverOpen** function initialises the VIPA H1 driver for access. Do not execute this function more than once.

Parameter s

none.

Returns

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This function must be the first one called. The driver may be closed by executing the function **H1DriverClose**.

See also

H1DriverClose

7.6.2.2.2 Driver close

```
#include <H1Def.h>
unsigned short WENTRY_C H1DriverClose(void)
```

The **H1DriverClose** function releases any allocated resources.
All the connections established for the current task are terminated.

Parameters

none

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This must be the last function executed. The driver becomes accessible only after **H1DriverOpen**. All established connections of the current task are terminated.

See also

H1DriverOpen

7.6.2.2.3 Determine the version of the driver

```
#include <H1Def.h>
unsigned short WENTRY_C H1HoleVersion(unsigned short W_POINTER
version)
```

This function determines the version of the H1 software.

Parameters

version a pointer to an unsigned short variable that contains the version number of the H1 driver in BCD code.

Example: a return of 304 indicates version 3.04

Return

0
H1_No_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

Note

The version number is defined as follows: The most significant digit specifies the main version number and the two least significant digits specify the sub-version.

The driver must have been opened by means of **H1DriverOpen**.

7.6.2.2.4 Establish a connection

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteVerbindung(H1_CONNECT_PARAMS
W_POINTER cr)
```

The function **H1StarteVerbindung** establishes connections.

Parameters

cr a pointer to a structure of the type **H1_CONNECT_PARAMS**. The connection parameters must have been defined.

Possible values returned by **cr.Fehler**:

```
0
H1_BAD_CR_PARAMS
H1_WAIT_CONNECT
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER
```

Returns

<> 0 an operating system error has occurred.
0 the **cr.Fehler** element contains one of the above values.

Note

If this function was completed successfully and without an error a connection number is returned via the structure. This number is valid until the connection is terminated by means of **H1StoppeVerbindung**. It is not possible to terminate the attempt to establish a connection by means of **H1StoppeVerbindung**.

The driver must have been opened by means of **H1DriverOpen**.

See also

```
H1StarteVerbindungOverlapped
H1StoppeVerbindung
H1StoppeVerbindungOverlapped
H1StoppeVerbindungen
H1StoppeVerbindungenOverlapped
H1GetOverlappedResult
```

7.6.2.2.5 Establish a connection and continue (asynchronous)

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteVerbindungOverlapped(
H1_CONNECT_PARAMS W_POINTER cr, LPOVERLAPPED ov)
```

The function **H1StarteVerbindungOverlapped** is used to establish a connection. This function must be used if you do not want to wait for the completion of the function establishing the connection. The completion of the establishment of a connection is signalled by an event in the **ov** structure. You may check whether the process has been completed by means of the operating system function **H1GetOverlappedResult**.

Parameters

cr a pointer to a structure of the type **H1_CONNECT_PARAMS**. The connection parameters and the adapter number must have been specified.

Possible values returned by **cr.Fehler**:

```
0
H1_BAD_CR_PARAMS
H1_NO_SLOT
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

ov a pointer to a structure of the type **OVERLAPPED**. You must first create an event. The event is enabled when the connection has been established.

The function **GetOverlappedResult** will now return a value of **TRUE** to indicate that the connection has been established successfully.

Returns

<> 0 an operating system error has occurred.

0 the **cr.Fehler** element contains one of the above values.

Note

If this function was completed successfully and without an error a connection number is returned via the structure. This number is valid until the connection is terminated by means of **H1StoppeVerbindung**. You may define a time limit for the attempt to establish a connection by means of the function **H1SetzeStandardwerte**. The driver must have been opened by means of **H1DriverOpen**.

See also

H1StarteVerbindung	H1StoppeVerbindung
H1StoppeVerbindungen	H1StoppeVerbindungOverlapped
H1GetOverlappedResult	H1StoppeVerbindungenOverlapped

7.6.2.2.6 Close a connection

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StoppeVerbindung(CONNREF vnr)
```

The function **H1StoppeVerbindung** is used to close the connection for a specific adapter. The adapter number is contained in the connection number.

Parameters

vnr	the connection number consists of the adapter number in the most significant word and the connection reference in the least significant word.
-----	---

Returns

```
0  
H1_NO_SLOT  
H1_BAD_LINE  
H1_NO_DRIVER  
H1_NO_ADAPTER
```

Note

The connection number is not available any more once the function terminates.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung** or **H1StarteVerbindungOverlapped**.

See also

```
H1StarteVerbindung  
H1StarteVerbindungOverlapped  
H1StoppeVerbindungen  
H1StoppeVerbindungOverlapped  
H1StoppeVerbindungenOverlapped  
H1GetOverlappedResult
```

7.6.2.2.7 Close a connection and continue (asynchronously)

```
#include <H1Def.h>
unsigned short WENTRY_C H1StoppeVerbindungOverlapped(CONNREF
verbindungsnummer, LPOVERLAPPED ov)
```

The **H1StoppeVerbindungOverlapped** function initiates the closing of an existing connection and returns to the originator immediately. The completion is signalled by an event in the **ov** structure. You may check whether the process has been completed by means of the operating system function **H1GetOverlappedResult**.

Parameters

verbindungsnummer	the connection number contains of the adapter number in the most significant word and the connection number in the least significant word.
ov	pointer to a structure of the type OVERLAPPED . You must have created an event previously. The event will be issued when the connection has been closed successfully. The function H1GetOverlappedResult will then return TRUE to indicate that the connection has been terminated.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

Note

This connection number can no longer be used once this function completes.
The driver must have been opened by means of **H1DriverOpen** and a connection established and by means of **H1StarteVerbindung** or **H1StarteVerbindungOverlapped**.

See also

H1StarteVerbindung
H1StarteVerbindungOverlapped
H1StoppeVerbindung
H1StoppeVerbindungen
H1StoppeVerbindungenOverlapped
H1GetOverlappedResult

7.6.2.2.8 Close all connections

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StoppeVerbindungen(void)
```

The function **H1StoppeVerbindungen** closes all established connections.

Parameters

none

Returns

0

H1_BAD_LINE

H1_NO_DRIVER

H1_NO_ADAPTER

Note

All connection numbers that have been established will become invalid when this function has completed.

The driver must have been opened by means of **H1DriverOpen** and a connection established by means of **H1StarteVerbindung** or **H1StarteVerbindungOverlapped**.

See also

H1StarteVerbindung

H1StarteVerbindungOverlapped

H1StoppeVerbindung

H1StoppeVerbindungOverlapped

H1StoppeVerbindungenOverlapped

H1GetOverlappedResult

7.6.2.2.9 Close all connections and continue (asynchronously)

```
#include <H1Def.h>
unsigned short WENTRY_C H1StoppeVerbindungenOverlapped (LPOVERLAPPED
ov)
```

The **H1StoppeVerbindungenOverlapped** function initiates the closing of all existing connections and returns to the originator immediately. The completion is signalled by an event in the **ov** structure. You may check whether the process has been completed by means of the operating system function **H1GetOverlappedResult**.

Parameters

ov pointer to a structure of the type **OVERLAPPED**. You must have created an event previously. The event will be issued when the connection has been closed successfully.

The function **H1GetOverlappedResult** will then return TRUE to indicate that the connection has been terminated.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

Note

All connection numbers become unavailable once this function completes.

The driver must have been opened by means of **H1DriverOpen** and a connection established by means of **H1StarteVerbindung** or **H1StarteVerbindungOverlapped**.

See also

H1StarteVerbindung
H1StarteVerbindungOverlapped
H1StoppeVerbindung
H1StoppeVerbindungOverlapped
H1StoppeVerbindungen
H1GetOverlappedResult

7.6.2.2.10 Determine station address

```
#include <H1Def.h>
unsigned short WENTRY_C H1HoleStationsAdresse(unsigned char W_POINTER
adresse, int nKarte=0)
```

This function determines the station address of the local (own) station.

Parameters

adresse a pointer to a memory area that contains the station parameters.

 Byte 0,1 = length of the station address

 Byte 2..7 = current station address

 Byte 8..13 = ROM station address

nKarte adapter number (0 - 3), default=0. This is a default parameter that may not be required.

Returns

0

H1_NO_DRIVER

H1_NO_ADAPTER

Note

Station parameters consist of two sets of ethernet addresses. These are the permanent address located in the ROM of the adapter, and the current address. The latter address is currently used on the ethernet LAN.

See also

H1SetzeStationsAdresse

7.6.2.2.11 Set the station address

```
#include <H1Def.h>
unsigned short WENTRY_C H1SetzeStationsAdresse(unsigned char
W_POINTER adresse, int nKarte=0)
```

You can change the local (own) station address by means of this function.

Parameters

adresse	pointer to a memory area containing the new station address. These 6 bytes contain the station address. For H1 the most significant byte must not be a broadcast- or a multicast address.
nKarte	adapter number (0 - 3), default = 0 . This is a default parameter that may not be required.

Returns

0
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

Note

Station addresses must never be duplicated! This would cause collisions on the network. The station address can only be altered if the MAC driver (H1 driver) is set to operate in promiscuous mode. For **WindowsNT** you may select this mode in **Network services**. The default value for this setting is the promiscuous mode.

To guarantee proper operation the station address must not be changed on a running system. Promiscuous mode places heavy demands on the system, as every message must be received and decoded. Please refer to the **Technical Manual** for your network if you require more detailed information.

Example

```
unsigned char adresse [6];
    adresse[0] = 0x00;
    adresse[1] = 0x20;
    adresse[2] = 0xd5;
    adresse[3] = 0x80;
    adresse[4] = 0x02;
    adresse[5] = 0x01;
    H1SetzeStationsAdresse(adresse,2); // Address for adapter 2
```

See also

H1HoleStationsAdresse

7.6.2.2.12 Read H1 system values

```
#include <H1Def.h>
unsigned short WENTRY_C H1HoleStandardwerte(H1_INITVALUES
W_Pointer init, int nKarte=0)
```

This function displays the H1 system values.

Parameters

init	pointer to a memory area containing the system values.
nKarte	adapter number (0 - 3), default=0. This is a default parameter that may not be required.

Returns

0
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

Note

These settings affect all connections. Should the H1 system values be altered by means of **H1SetzeStandardwerte** on a running system, then the new parameters will be used when a new connection is established.

See also

H1SetzeStandardwerte

7.6.2.2.13 Write H1 system values

```
#include <H1Def.h>
unsigned short WENTRY_C H1SetzeStandardwerte(H1_INITVALUES
W_Pointer init, int nKarte=0)
```

You can change the H1 standard values by means of this function.

Parameters

<code>init</code>	pointer to a memory area containing the new system values.
<code>nKarte</code>	adapter number (0 - 3), default=0. This is a default parameter that may not be required.

Returns

0
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

Note

These settings affect all connections. Should the H1 standard values be altered buy means of **H1SetzeStandardwerte** on a running system, then the revised parameters will be used immediately. The result may be that one connection operates with both, old and new parameters.

H1 system values

Window sizes

<code>MaxCredit</code>	Maximum window size. Permanently 0.
<code>MinCredit</code>	Minimum window size. Permanently 0.

Default settings for requests

<code>PersistenceCount</code>	specifies a time limit for an attempt to establish a connection (multiple of the <code>AbortTimeout</code>). Default: 1.
<code>AbortTimeout</code>	monitoring time for request (multiples of 10 milliseconds). Default: 6000 = 1 minute.

Settings used when negotiating the transport connection

<code>ProtOption</code>	code defining the transport protocol option (fixed at 2).
<code>ProtClass</code>	code defining the transport protocol class (fixed at 4).
<code>TPDUSize</code>	maximum size of a transport protocol data unit (TPDU) as a power of two. Default: $10 = 2^{10} = 1024$ bytes.
<code>TPDUAddOpt</code>	parameter for the transmission of retry messages (fixed at 3).
<code>RetransmissionTimeout</code>	time delay before a TP-ACK PDU is repeated (multiple of 10 milliseconds). Default: 100 = 1 second

MinRetransmissionTime	multiples of 10 milliseconds. Default: 10 = 0,1 second
ClosingAbortTimeout	multiples of 10 milliseconds. Default: 600 = 6 seconds.

Parameter for transport acknowledgements (ACK's)

FlowControlWindowTimeout	multiples of 10 milliseconds. Default: 1000
InactivityMaxCount	maximum value: default: 3

Repetition of acknowledgements

OpenWindowTimeout	multiples of 10 milliseconds. Default: 100 = 0,1 second.
MaxOpenWindowCount	maximum value: default: 8

See also

H1HoleStandardwerte

7.6.2.3 H1 layer 4 specific functions

7.6.2.3.1 Transmitting data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SendeDaten(H1_SENDFPARAMS W_POINTER send)
```

This function transmits data over an established connection. Once the function is initiated it will only return when the transmission was completed successfully or when the connection was terminated.

Parameters

send pointer to a structure containing transmission parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_SENDFPARAMS) + **DataLen**. Bit 15 of element **Vnr** must be set if the message must form a part of a blocked transmission. On the receiving side this bit sets **H1_BLOCKED_DATA** in the element **Fehler**. Bit 15 must never be set in the last message of a blocked transmission.

Possible values returned by **send.Fehler** are:

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

< > 0 An error has occurred in the operating system.
0 The **send.Fehler** element contains one of the above values

Note

This function will only return once the transmission has been completed successfully or when a timeout occurs. The function may be started and polled on a cyclic basis by means of the functions **H1StarteSenden**/**H1AbfrageSenden**. The function **H1SendeDatenOverlapped** continues and does not wait for completion.

The driver must have been opened by means of **H1DriverOpen** and a connection established by means of **H1StarteVerbindung** or **H1StarteVerbindungOverlapped**.

See also

```
H1SendeDatenEx
H1SendeDatenExOverlapped
H1StarteSenden
H1StarteSendenOverlapped
H1StarteSendenExOverlapped
H1AbfrageSenden
H1GetOverlappedResult
```

7.6.2.3.2 Transmit expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1SendeDatenEx(H1_SENDFPARAMS W_POINTER  
send)
```

This function transmits expedited data on an existing connection if the connection permits transmission of such data. Expedited data is priority or urgent data.

Parameters

send pointer to a structure containing transmit parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_SENDFPARAMS) + **DataLen**.
Blocked data messages are not permitted.

Possible values returned by **send.Fehler** are:

```
0  
H1_BAD_LINE  
H1_NO_DRIVER  
H1_NO_ADAPTER  
H1_NOT_SUPPORTED
```

Returns

<> 0 An error has occurred in the operating system.
0 The **send.Fehler** element contains one of the above values.

See also

```
H1SendeDaten  
H1starteSenden  
H1starteSendenOverlapped  
H1starteSendenExOverlapped  
H1AbfrageSenden  
H1GetOverlappedResult
```

7.6.2.3.3 Start send

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteSenden(H1_SENDFPARAMS W_POINTER
send)
```

This function starts a send task. The function **H1AbfrageSenden** must be called repeatedly until the send task completes successfully.

Parameters

send pointer to a structure containing transmit parameters. The parameters **Vnr**, **DataLen** and **SendLen** (=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_SENDFPARAMS) + **DataLen**. Bit 15 of element **Vnr** must be set if the message must form a part of a blocked transmission. On the receiving side this bit sets **H1_BLOCKED_DATA** in the element **Fehler**. Bit 15 must never be set in the last message of a blocked transmission.

Possible values returned by **send.Fehler** are:

```
0
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

< > 0 An error has occurred in the operating system.
0 The **send.Fehler** element contains one of the above values.

Note

This function starts a send task. The result of the function can be determined by means of **H1AbfrageSenden**. Instead of the **H1StarteSenden/H1AbfrageSenden** pair you may also use the function **H1SendeDatenOverlapped** which indicates completion of the transmission by means of an event. The driver must have been opened by means of **H1DriverOpen** and a connection established by means of **H1StarteVerbindung**.

See also

```
H1SendeDaten
H1SendeDatenEx
H1StarteSendenOverlapped
H1StarteSendenExOverlapped
H1AbfrageSenden
H1GetOverlappedResult
```

7.6.2.3.4 Start sending data and continue (asynchronous)

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteSendenOverlapped(H1_SENDFPARAMS
W_POINTER send, LPOVERLAPPED ov)
```

This function initiates a send task for data on an established connection and immediately returns to the caller. The result of the function is returned via the `send` structure. The completion of the send procedure (successful, without error) is indicated by the event in the `ov` structure. You may also check for the termination of the send task by means of the operating system function **H1GetOverlappedResult**.

Parameters

`send` pointer to a structure containing transmit parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_SENDFPARAMS) + **DataLen**.
Bit 15 of element **Vnr** must be set if the message must form a part of a blocked transmission. On the receiving side this bit sets **H1_BLOCKED_DATA** in the element **Fehler**. Bit 15 must never be set in the last message of a blocked transmission.

Possible values returned by `send.Fehler` are:

```
0
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

`ov` pointer to a structure of the type **OVERLAPPED**. You must have created an event previously. The event will be issued when the connection has been closed successfully.

The function **H1GetOverlappedResult** will then return TRUE.

Returns

< > 0 An error has occurred in the operating system.
0 The `send.Fehler` element contains one of the above values

Note

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1SendeDaten
H1SendeDatenEx
H1StarteSenden
H1StarteSendenExOverlapped
H1AbfrageSenden
H1GetOverlappedResult
```

7.6.2.3.5 Start sending expedited data and continue (asynchronous)

```
#include <HlDef.h>
unsigned short WENTRY_C HlStarteSendenExOverlapped (H1_SENDFPARAMS
W_POINTER send, LPOVERLAPPED ov)
```

This function initiates a send task for expedited data (urgent data) on an established connection and immediately returns to the caller. The result of the function is returned via the `send` structure. The completion of the send procedure (successful, without error) is indicated by the event in the `ov` structure. You may also check for the termination of the send task by means of the operating system function **HlGetOverlappedResult**.

Parameters

`send` pointer to a structure containing transmit parameters. The parameters **Vnr**, **DataLen** and **SendLen** (=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_SENDFPARAMS) + DataLen. Blocked data messages are not permitted.

Possible values returned by `send.Fehler` are:

```
0
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

`ov` pointer to a structure of the type **OVERLAPPED**. You must have created an event previously. The event will be issued when the connection has been closed successfully.

The function **HlGetOverlappedResult** will then return TRUE.

Return

< > 0 An error has occurred in the operating system.
0 The `send.Fehler` element contains one of the above values

Note

The driver must have been opened by means of **HlDriverOpen** and the respective connection must have been established by means of **HlStarteVerbindung**.

See also

```
HlSendeDaten
HlSendeDatenEx
HlStarteSenden
HlStarteSendenOverlapped
HlAbfrageSenden
HlGetOverlappedResult
```

7.6.2.3.6 Check send status

```
#include <H1Def.h>
unsigned short WENTRY_C H1AbfrageSenden(H1_SENDFPARAMS W_POINTER
send)
```

This function checks whether a send task that was initiated by means of **H1StarteSenden** has been completed.

Parameters

send a structure containing the send parameters.

Returns

< > 0 an error has occurred in the operating system.

0 the send.Fehler element contains one of the values below.

```
0
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Note

You may use the **H1StarteSendenOverlapped** function instead of the pair of functions **H1StarteSenden**/**H1AbfrageSenden**. In this case completion of the transmission is signalled by an event. The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1SendeDaten
H1SendeDatenEx
H1StarteSenden
H1StarteSendenOverlapped
H1StarteSendenExOverlapped
H1GetOverlappedResult
```

The following page contains a flow chart depicting how these functions should be implemented in your program.

7.6.2.3.7 Flowchart "Transmitting a message"

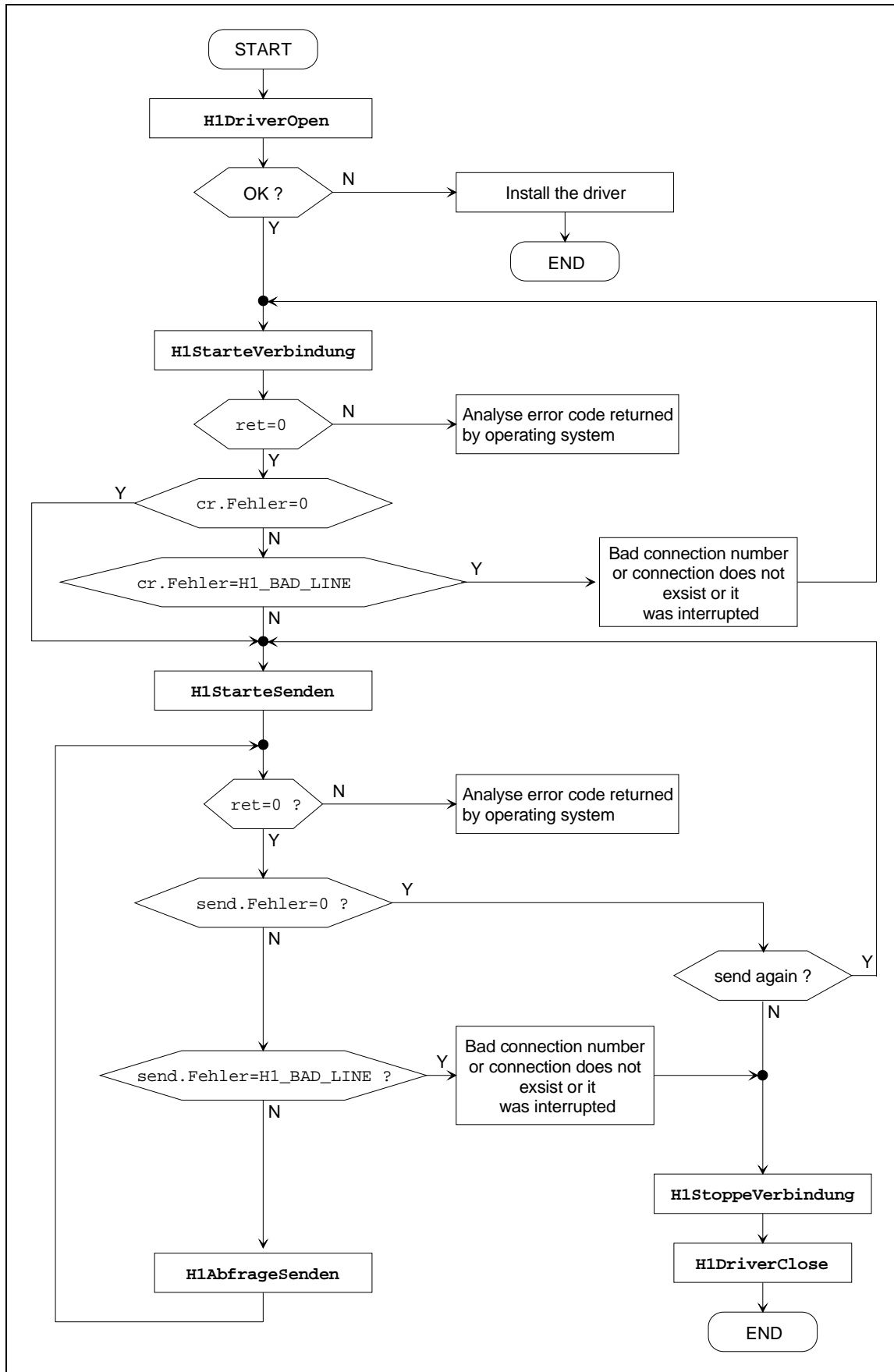


Fig. 7-5: Flowchart for "Transmitting a message"

7.6.2.3.8 Read data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1LeseDaten(H1_REC_PARAMS W_POINTER rec)
```

This function is used to read data. It will only return to the caller when data has been received or when the connection has been terminated.

Parameters

rec pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_REC_PARAMS) + **DataLen**. Possible values returned by **rec.Fehler** are:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

<> 0 An error has occurred in the operating system.
0 The **rec.Fehler** element contains one of the above values.

Note

This function only returns to the caller when data was received successfully or when a timeout has occurred. Where large quantities of data are received resulting in the **H1_BLOCKED_DATA** message, you must ensure that **H1LeseDaten** is called repeatedly until all data has been transferred. In this case the **RecLen** parameter is used as input and as return parameter. **RecLen** must be set to the correct value before the function is called and it represents the offset of the data. When the function completes, it returns the length of the data block read in **RecLen**. This length must be deducted from **DataLen** every time another read function is initiated (see example).

The functions **H1StarteLesen**/**H1AbfrageLesen** are available for starting and checking the read function repeatedly (polling). You may not use both **H1LeseDaten** and **H1StarteLesen**/**H1AbfrageLesen** tasks for a single connection.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

H1LeseDatenEx
H1StarteLesen
H1StarteLesenEx
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesen
H1AbfrageLesenEx
H1GetOverlappedResult

Example

```
H1_RECPARAMS *H1Rec;
int RecLen;
int ret =0;
.
.
H1Rec = (H1_RECPARAMS*)malloc(sizeof(H1_RECPARAMS)+RECVLEN);

H1Rec->Vnr = H1ConnectParams.Vnr; /* Connection number */

H1Rec->DataLen=RECVLEN; /* Size of the data packet */
RecLen = H1Rec->RecLen =0; /* reset */

H1Rec->Fehler = H1_WAIT_DATA;

while( !ret && H1Rec->DataLen && H1Rec->Fehler)
/* Condition : DataLen >= the actual data that was sent
*/
{
    RecLen= H1Rec->RecLen;
    ret    = H1LeseDaten( H1Rec);

    if(!ret && H1Rec->Fehler == H1_BLOCKED_DATA ) {
        H1Rec->DataLen -= H1Rec->RecLen; /* deduct the last
                                         block received */
        H1Rec->RecLen  += RecLen;        /* Offset */
    }
}
```

7.6.2.3.9 Read expedited data

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1LeseDatenEx(H1_REC_PARAMS W_POINTER rec)
```

This function initiates a read task for expedited data. Expedited data is priority or urgent data.

Parameters

rec pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_REC_PARAMS) + **DataLen**.

Possible values returned by **rec.Fehler** are:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

< > 0 An error has occurred in the operating system.
0 The **rec.Fehler** element contains one of the above values.

Note

This function only returns to the caller when data was received successfully or when a timeout has occurred. The functions **H1StarteLesen**/**H1AbfrageLesen** are available for starting and checking the read function repeatedly (polling). You may not use both **H1LeseDaten** and **H1StarteLesen**/**H1AbfrageLesen** tasks for a single connection.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1LeseDaten
H1StarteLesen
H1StarteLesenEx
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesen
H1AbfrageLesenEx
H1GetOverlappedResult
```

7.6.2.3.10 Start read

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteLesen(H1_REC_PARAMS W_POINTER rec)
```

This function initiates a read task.

Parameter s

rec pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_REC_PARAMS) + **DataLen**.

Possible values returned by **rec.Fehler** are:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Return

< > 0 an error has occurred in the operating system.
0 the **rec.Fehler** element contains one of the above values.

Note

This function initiates a read task. The result of the function can be checked by means of **H1AbfrageLesen**. When **H1_WAIT_DATA** is returned the function **H1AbfrageLesen** must be called again.

Where large quantities of data are received resulting in the **H1_BLOCKED_DATA** message, you must ensure that **H1LeseDaten** is called repeatedly until all data has been transferred. In this case the **RecLen** parameter is used as input and as return parameter. **RecLen** must be set to the correct value before the function is called and it represents the offset to the data. When the function completes it returns the length of the data block read in **RecLen**. This length must be deducted from **DataLen** every time another read function is initiated (see example).

You may not use both **H1LeseDaten** and **H1StarteLesen**/**H1AbfrageLesen** tasks for a single connection.

You may use the **H1StarteLesenOverlapped** function instead of the pair of functions **H1StarteLesen**/**H1AbfrageLesen**. In this case completion of the transmission is signalled by an event.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

H1LeseDaten
H1LeseDatenEx
H1StarteLesenEx
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesen
H1AbfrageLesenEx
H1GetOverlappedResult

Example

```
H1_RECPARAMS *H1Rec
int ret=0;
int RecLen;
.
.
H1Rec = (H1_RECPARAMS*)malloc(sizeof(H1_RECPARAMS)+RECLLEN);

H1Rec->Vnr = H1ConnectParams.Vnr; /* Connection number */

H1Rec->DataLen = RECLLEN;          /* Size of data packet */
RecLen = H1Rec->RecLen= 0;        /* reset */

H1Rec->Fehler = H1_WAIT_DATA;

while(!ret && H1Rec->DataLen && H1Rec->Fehler)
/* Condition : DataLen >= the data actually transmitted */
{
    RecLen = H1Rec->RecLen; /* remember last offset */
    ret = H1StarteLesen(H1Rec);

    while (!ret && H1Rec->Fehler == H1_WAIT_DATA)
        ret = H1AbfrageLesen(H1Rec);

    if (H1Rec->Fehler == H1_BLOCKED_DATA) {
        H1Rec->DataLen -= H1Rec->RecLen; /* deduct last
                                           received block */
        H1Rec->RecLen += RecLen;        /* Offset */
    }
    else
    {
        /* Error=H1_BAD_LINE, H1_NO_DRIVER or H1_NO_ADAPTER
           or H1_NOT_SUPPORTED */
    }
}
```

7.6.2.3.11 Start read data and continue (asynchronous)

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1StarteLesenOverlapped(H1_REC_PARAMS  
W_POINTER rec, LPOVERLAPPED ov)
```

This function initiates a read operation on an established connection and immediately returns to the caller. The result of the function is returned in the structure `rec`. The completion of the read operation (successful or error) is indicated by setting the event in the `ov` structure. You may also check for completion of the read operation by means of the operating system function **H1GetOverlappedResult**.

Parameters

`rec` pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_REC_PARAMS) + **DataLen**.

Possible values returned by `rec.Fehler` are:

```
0  
H1_BLOCKED_DATA  
H1_BAD_LINE  
H1_WAIT_DATA  
H1_NO_DRIVER  
H1_NO_ADAPTER  
H1_NOT_SUPPORTED
```

Returns

<> 0 an error has occurred in the operating system.
0 the `rec.Fehler` element contains one of the above values.

Note

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1LeseDaten  
H1LeseDatenEx  
H1StarteLesen  
H1StarteLesenEx  
H1StarteLesenExOverlapped  
H1AbfrageLesen  
H1AbfrageLesenEx  
H1GetOverlappedResult
```

7.6.2.3.12 Start reading expedited data

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteLesenEx(H1_REC_PARAMS W_POINTER
rec)
```

This function initiates a read task for expedited data. Expedited data is priority or urgent data.

Parameters

rec pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_REC_PARAMS) + DataLen.

Possible values returned by **rec.Fehler** are:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

< > 0 an error has occurred in the operating system.
0 The **rec.Fehler** element contains one of the above values.

Note

This function initiates a read task. The result of the function can be checked by means of **H1AbfrageLesenEx**. You may use the **H1StarteLesenExOverlapped** function instead of the pair of functions **H1StarteLesenEx**/**H1AbfrageLesenEx**. In this case completion of the transmission is signalled by an event.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**. You may not use both **H1LeseDatenEx** and **H1StarteLesenEx**/**H1AbfrageLesenEx** tasks for a single connection.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1LeseDaten
H1LeseDatenEx
H1StarteLesen
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesen
H1AbfrageLesenEx
H1GetOverlappedResult
```

7.6.2.3.13 Start reading expedited data and continue (asynchronous)

```
#include <H1Def.h>
unsigned short WENTRY_C H1StarteLesenExOverlapped(H1_RECPARAMS
W_POINTER rec, LPOVERLAPPED ov)
```

This function initiates a read task for expedited data. Expedited data is priority or urgent data.

This function initiates a read operation on an established connection and immediately returns to the caller. The result of the function is returned in the `rec` structure. The completion of the read operation (successful or error) is indicated by setting the event in the `ov` structure. You may also check for completion of the read operation by means of the operating system function **H1GetOverlappedResult**.

Parameters

`rec` pointer to a structure containing receive parameters. The parameters **Vnr**, **DataLen** and **SendLen**(=0) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_RECPARAMS) + **DataLen**.

Possible values returned by `rec.Fehler` are:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

`ov` pointer to a structure of the type **OVERLAPPED**. The **hEvent** must be set to a handle for a valid event. The event must be created by the user.

Returns

< > 0 An error has occurred in the operating system.
0 The `rec.Fehler` element contains one of the above values.

Note

This function initiates a read task. The result of the function can be checked by means of **H1AbfrageLesenEx**. You may not use both **H1LeseDaten** and **H1StarteLesen**/**H1AbfrageLesen** tasks for a single connection.

The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

H1LeseDaten	H1LeseDatenEx	H1StarteLesenEx
H1StarteLesen	H1AbfrageLesenEx	H1AbfrageLesen
H1StarteLesenOverlapped		H1GetOverlappedResult

7.6.2.3.14 Check read

```
#include <H1Def.h>
```

```
unsigned short WENTRY_C H1AbfrageLesen(H1_REC_PARAMS W_POINTER rec)
```

This function determines whether a read task that was initiated by means of **H1StarteLesen** is still active.

Parameter s

<code>rec</code>	pointer to a structure containing receive parameters. Parameters Vnr and DataLen must be specified. The buffer located at the end of the structure must have a length of <code>sizeof(H1_REC_PARAMS) + DataLen</code> .
------------------	---

Possible values returned by `rec.Fehler`:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

<code><> 0</code>	An error has occurred in the operating system.
<code>0</code>	The <code>rec.Fehler</code> element contains one of the above values.

Note

When a return value of `H1_BLOCKED_DATA` occurs the function **H1StarteLesen** must be called again. You may not use both **H1LeseDaten** and **H1StarteLesen/H1AbfrageLesen** tasks for a single connection. The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**.

See also

```
H1LeseDaten
H1LeseDatenEx
H1StarteLesen
H1StarteLesenEx
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesenEx
H1GetOverlappedResult
```

7.6.2.3.15 Check read expedited data

```
#include <H1Def.h>
unsigned short WENTRY_C H1AbfrageLesenEx(H1_RECPARAMS W_POINTER rec)
```

This function determines whether a read task for expedited data that was initiated by **H1StarteLesenEx** is still active. Expedited data consists of priority data or urgent data.

Parameters

rec pointer to a structure containing receive parameters. Parameters **Vnr** , **DataLen** and **RecLen** (=offset) must be specified. The buffer located at the end of the structure must have a length of **sizeof**(H1_RECPARAMS) + DataLen.

Possible values returned by **rec.Fehler**:

```
0
H1_BLOCKED_DATA
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NOT_SUPPORTED
```

Returns

<> 0 An error has occurred in the operating system.
0 The **rec.Fehler** element contains one of the above values.

Note

You may not use both **H1LeseDaten** and **H1StarteLesen/H1AbfrageLesen** tasks for a single connection. The driver must have been opened by means of **H1DriverOpen** and the respective connection must have been established by means of **H1StarteVerbindung**. When **H1_WAIT_DATA** is returned the function **H1AbfrageLesenEx** must be called again (Polling).

See also

```
H1LeseDaten
H1LeseDatenEx
H1StarteLesen
H1StarteLesenEx
H1StarteLesenOverlapped
H1StarteLesenExOverlapped
H1AbfrageLesen
H1GetOverlappedResult
```

7.6.2.3.16 Check read data and continue (asynchronous)

```
#include <H1Def.h>
```

```
BOOL WENTRY_C H1GetOverlappedResult(CONNREF Vnr, LPOVERLAPPED ov,  
LPWORD lpNumberOfBytesTransferred, BOOL bWait)
```

This function may be used to determine whether an **OVERLAPPED** task has been completed.

Parameters

vnr	the connection number containing of the card number in the most significant word and the connection reference in the least significant word.
ov	pointer to a structure of the type OVERLAPPED . This is a structure that is used for Overlapped functions. The completion of the above functions can be checked by means of a call to the function H1GetOverlappedResult .
lpNumberOfBytesTransferred	the number of bytes that where transferred
bWait	determines whether the function returns when a "Pending" condition occurs. If this is TRUE the function will wait for completion of the OVERLAPPED operation.

Returns

TRUE	function completed.
FALSE	function not yet completed.

See also

```
H1StarteVerbindungOverlapped  
H1StoppeVerbindungOverlapped  
H1StoppeVerbindungenOverlapped  
H1SendeDatenExOverlapped  
H1StarteSendenOverlapped  
H1StarteSendenExOverlapped  
H1StarteLesenOverlapped  
H1StarteLesenExOverlapped
```

7.6.2.3.17 Flowchart "Read a message"

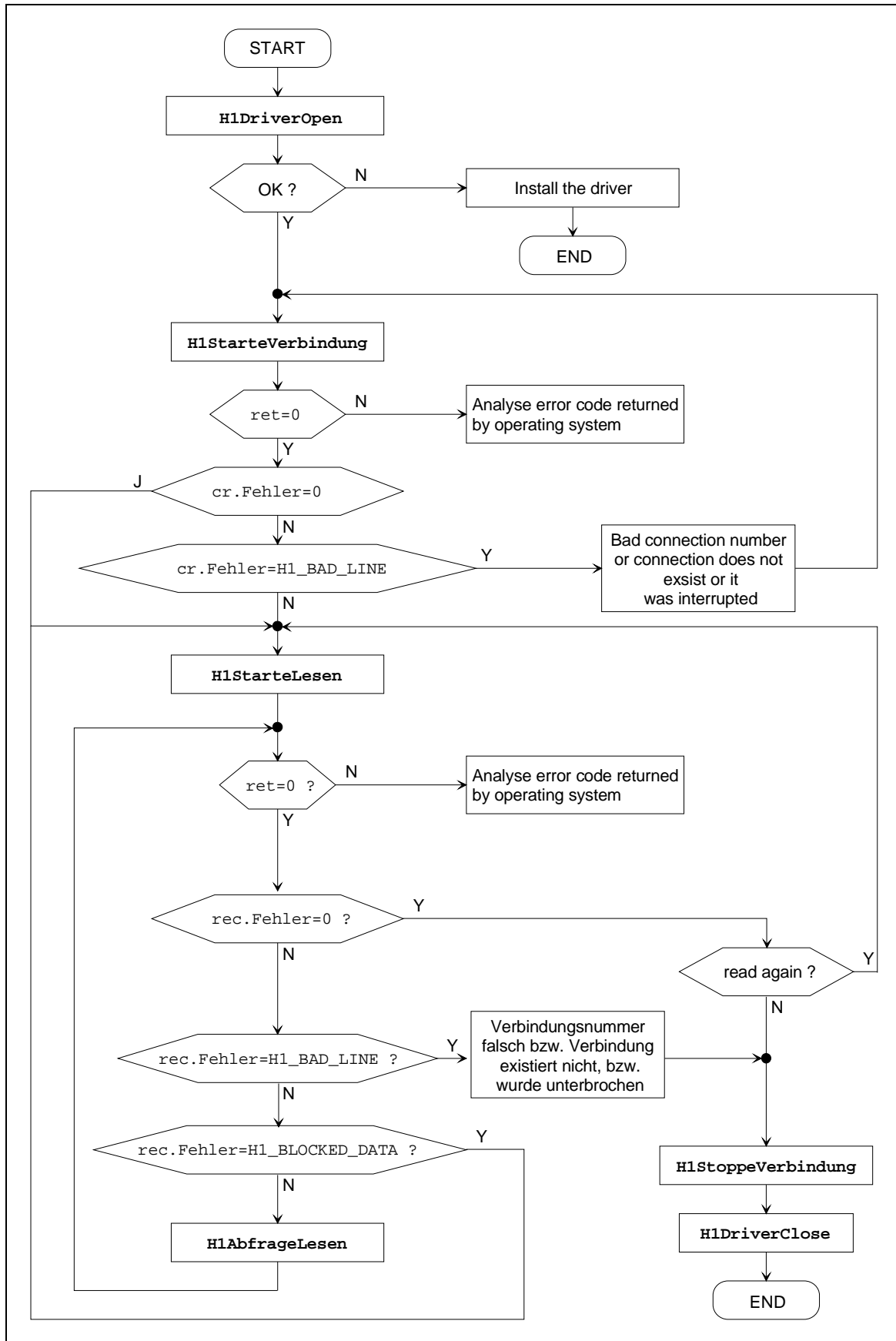


Fig. 7-6: Flowchart for "Read a message"

7.6.3 PLC Layer 7 software interface

7.6.3.1 General information on the PLC software interface

The PLC programming functions provide direct access to PLC's via your Industrial Ethernet (H1) network. You can access to access all areas e.g. data modules, function modules, system modules etc. You may, for instance, alter individual data items or load and overwrite complete modules. The different PLC's on a network are addressed by means of connection parameters. Before you access any PLC you must have established a connection to this controller. You may then use this connection to communicate with the PLC.

It is possible to establish multiple connections to a single or to many destination systems. The type of connection is determined by means of connection parameters. For the CP143plus on the PLC side these parameters are generated by means of the NCS configuration program.

Connections that are no longer required should be closed.

You must configure the passive tasks **SendAll** and **RecAll** before communications between PLC and PC can be established.

The PLC functions **S5StarteLesen(Overlapped)**, **S5AbfrageLesen(Overlapped)**, **S5StartSchreiben(Overlapped)**, **S5AbfrageSchreiben(Overlapped)**, **S5LeseAusSPS(Overlapped)**, **S5SchreibeInSPS(Overlapped)** represent the active part of a connection.

When a request for data is issued from your PC, the request to the PLC also contains the source of the data. The PLC responds with **SendAll** to your request. It retrieves the data from the specified source and transfers these to the PC.

When data must be transferred from the PC to the PLC the destination is included in the data sent to the PLC. The **RecAll** task receives the data and forwards it to specified destination.

All the functions defined on the H1 level are supported.

You may also configure your PC as passive and your PLC as active. The functions **S5FetchPassiv** and **S5WritePassiv** provide the basis for a simple PLC-simulation, i.e. the PC behaves as a passively configured PLC. The PC waits for tasks that instruct it to read or write data.

7.6.3.2 General layer 7 functions

7.6.3.2.1 Start connection

```
#include <S5ACCESS.H>
```

```
USHORT WENTRY_C s5StarteVerbindung(H1_CONNECT_PARAMS W_POINTER cr)
```

The **s5StarteVerbindung** function is used to define a connection to another system on the network.

You must establish a connection by means of **s5StarteVerbindung** before you can use it.

Parameters

cr pointer to a structure containing connection parameters.

The value **Vnr** is set if the function completes without error. This represents the connection number. However, this number will only be returned if:

- all drivers have been installed.
- the connection parameters are correct.
- the remote station acknowledges the connection if required by the connection parameters.

A valid connection number does not imply that the connection to the partner has been established. This also depends on the connection parameters.

Returns

0
H1_WAIT_CONNECT
H1_BAD_CR_PARAMS
H1_NO_SLOT
H1_NO_DRIVER
H1_NO_ADAPTER

See also

s5StoppeVerbindung

7.6.3.2.2 Terminate a connection

```
#include <S5ACCESS.H>
```

```
USHORT WENTRY_C S5StoppeVerbindung(CONNREF vnr)
```

The **S5StoppeVerbindung** function is used to terminate a defined connection. It must be executed even if the connection has never truly existed. All internal memory locations in the library and in the drivers are released when this function completes.

Parameters

vnr Here you must supply a valid connection number of a connection that was started by means of **S5StarteVerbindung** and that has not yet been terminated by **S5StoppeVerbindung**.

The connection number contains the adapter number in the most significant word and the connection number in the least significant word.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung

7.6.3.2.3 Terminate all connections

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5StoppeVerbindungen (void)
```

The **S5StoppeVerbindungen** function is used to terminate all established connections.

Parameters

none

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung

7.6.3.2.4 Get revision levels

```
#include <S5ACCESS.H>
```

```
USHORT WENTRY_C S5HoleRevision(CP_REVISION W_POINTER cprev)
```

This function returns the revision levels of all installed drivers and libraries in a structure called **cprev**.

In your program you should always make sure that the minimum revision level of the installed drivers is matched or exceeded. You can retrieve the revision level of the H1 driver by means of `cprev.H1Rev`. It is possible that some functions contained in this description are not implemented fully if they have an earlier revision level than required.

Parameters

<code>cprev</code>	pointer to the structure <code>CP_REVISION</code> . The element cb must contain the length of the structure. The easiest manner to achieve this is by means of sizeof(CP_REVISION) .
--------------------	--

Returns

0	OK
1	incorrect length of structure
H1_NO_DRIVER	
H1_NO_ADAPTER	

7.6.3.2.5 Determine station address

```
#include <S5Access.h>
USHORT WENTRY_C S5HoleStationsAdresse(unsigned char W_POINTER
adresse, int nKarte=0)
```

This function determines the station address of the local station (PC).

Parameters

adresse Pointer to a memory location containing the current station parameters.

- Byte 0,1 = length of station address
- Byte 2..7 = current station address
- Byte 8..13 = ROM station address (fixed)

nKarte adapter number (0 - 3) Default = 0. This parameter has a default value and may be ignored when the function is called.

Returns

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

The station parameter consists of two sets of ethernet addresses. These are the permanent address that is located in the ROM of the adapter, and the current address which may be modified. As long as the current station address is not modified, it is identical to the station address in ROM.

See also

H1DriverOpen
H1DriverClose
S5SetzeStationsAdresse

7.6.3.2.6 Set station address

```
#include <S5Access.h>
USHORT WENTRY_C S5SetzeStationsAdresse(unsigned char W_POINTER
adresse)
```

This function modifies the local (own) station address.

Parameters

adresse Pointer to a memory location containing the new station address. The remaining 6 bytes contain the station address.

For an H1 environment the highest byte may not contain a broadcast- or multicast-address.

Returns

0
H1_NO_DRIVER
H1_NO_ADAPTER

Note

The station address must be unique, otherwise it is possible that collisions occur on the network.

It is only possible to modify the station address when the MAC driver (H1 driver) is set to operate in promiscuous mode. In windows NT this mode can be selected from the properties of the network services (default=promiscuous).

To ensure proper operation you must not change the station address at run time. Promiscuous mode places severe demands on the system since every message must be received and decoded. It is most likely that messages will be lost. Please refer to the technical reference manual for your network.

Example

```
unsigned char adresse [6];
    adresse[0] = 0x00;
    adresse[1] = 0x20;
    adresse[2] = 0xd5;
    adresse[3] = 0x80;
    adresse[4] = 0x02;
    adresse[5] = 0x01;
    S5SetzeStationsAdresse(adresse);
```

See also

S5HoleStationsAdresse

7.6.3.3 Specific layer 7 functions

7.6.3.3.1 Fetch active

Read from PLC

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5LeseAusSPS(S5_PARAMS W_POINTER s5)
```

The **S5LeseAusSPS** function is used to read data from a controller. Your system will be unavailable until the function returns. You must only use this function if it will not impact upon other processes.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values: **Kennung**, **DB**, **DW** and **Len** in the structure **s5** must contain valid entries. The structure **s5** must contain a valid connection number that was started by means of the function **S5StarteVerbindung**. You must allocate the required amount of memory after the structure **s5**:

```
s5=malloc(sizeof(S5_PARAMS)+RECVLEN);
```

The function was completed properly if it returns 0, otherwise the return value contains the H1 error.

If the returned value is 0 you can access the data as follows:

```
PLCData[0]=(UCHAR*)s5+sizeof(S5_PARAMS)+0;
```

```
PLCData[1]=(UCHAR*)s5+sizeof(S5_PARAMS)+1;
```

Tab. 7-9 contains an overview of the H1 errors and their description. A 0 signifies that the data was transferred properly and without errors.

Fehler contains PLC errors returned by the other station. Chapter 7.6.3.4 contains an overview of PLC error codes. A 0 in **Fehler** indicates that the data was transferred without errors.

Returns

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

Note

The function returns control only when the data is complete or when an error is detected. This may require some time depending on the connection parameters as well as the responses from the other station. This function can not be interrupted.

See also

```
S5StarteLesen
S5AbfrageLesen
```

Read from PLC SPS and continue (asynchronous)

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5LeseAusSPSOverlapped(
    S5_PARAMS W_POINTER    s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5LeseAusSPSOverlapped** is used to read data from a PLC. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

Parameters

- | | |
|-------------|---|
| s5 | <p>pointer to a structure of the type <code>S5_PARAMS</code>. The values: Kennung, DB, DW and Len must contain valid entries. The structure s5 must contain a valid connection number that was returned the function S5StarteVerbindung. You must reserve a sufficiently large memory area at the end of the s5 structure to accommodate the data returned from the controller:</p> <pre>s5=malloc(sizeof(S5_PARAMS)+RECVLEN);</pre> <p>The function was completed properly if it returns 0, otherwise the return value contains the H1 error.</p> <p>If the returned value is 0 you can access the data as follows:</p> <pre>PLCData[0]=(UCHAR*)s5+sizeof(S5_PARAMS)+0; PLCData[1]=(UCHAR*)s5+sizeof(S5_PARAMS)+1;</pre> <p>Tab. 7-9 contains an overview of the H1 errors and their description. A 0 signifies that the data was transferred properly and without errors.</p> <p>Fehler contains PLC errors returned by the other station. Chapter 7.6.3.4 contains an overview of S5 error codes. A 0 in Fehler indicates that the data was transferred without errors.</p> |
| UserFctCall | <p>pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).</p> <p>The User-Function must be defined as follows:</p> <pre>void WENTRY_C UserCallback(S5PARAMS W_POINTER s5);</pre> |
| Timeout_ms | <p>the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.</p> |

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteLesenOverlapped
S5AbfrageLesenOverlapped

Start read operation

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5StarteLesen(S5_PARAMS W_POINTER s5)
```

The **S5StarteLesen** function initialises a read operation. The requested data is transferred by means of the **S5AbfrageLesen** function.

You must not issue multiple start read operations for a connection without retrieving returned data by means of **S5AbfrageLesen**.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**. You must allocate the required amount of memory after the **s5** structure:

```
s5=malloc(sizeof(S5_PARAMS)+RECVLEN);
```

If the returned value is 0 you can poll for data by means of **S5AbfrageLesen**, otherwise the returned value contains an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5LeseAusSPS
S5AbfrageLesen

Start read operation and continue (asynchronous)

```
#include <S5ACCESS.H>
USHORT WENTRY_C
S5StarteLesenOverlapped(
    S5_PARAMS W_POINTER s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5StarteLesenOverlapped** function initialises a read operation. The requested data is transferred by means of the **S5AbfrageLesenOverlapped** function. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

Parameter s

s5 pointer to a structure of the type `S5_PARAMS`. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**. You must allocate the required amount of memory after the **s5** structure:
`s5=malloc(sizeof(S5_PARAMS)+RECVLEN);`
 If the returned value is 0 you can poll for data by means of **S5AbfrageLesen**, otherwise the returned value contains an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

UserFctCall pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).

The User-Function must be defined as follows:

```
void WENTRY_C UserCallback(S5PARAMS W_POINTER
s5);
```

Timeout_ms the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.

Returns

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

See also

```
S5LeseAusSPS
S5AbfrageLesen
```


Check read operation

```
#include <S5ACCESS.H>
```

```
USHORT WENTRY_C S5AbfrageLesen(S5_PARAMS W_POINTER s5)
```

The **S5AbfrageLesen** function determines whether the data that was previously requested by means of a **S5StarteLesen** function is available. This function must be repeated until the data becomes available or until an error message indicating a faulty connection is received.

Parameter s

s5 pointer to a structure of the type S5_PARAMS. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

The function will either return a 0 or an H1 error.

If the returned value is 0 you can access the received data as follows:

```
PLCData[0]=(UCHAR*)s5+sizeof(S5_PARAMS)+0;
```

```
PLCData[1]=(UCHAR*)s5+sizeof(S5_PARAMS)+1;
```

Tab. 7-9 contains an overview of the H1 errors and their description. Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the S5 error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

Returns

0
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER

See also

S5LeseAusSPS
S5StarteLesen

Check read and continue (asynchronously)

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5AbfrageLesenOverlapped(
    S5_PARAMS W_POINTER s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5AbfrageLesenOverlapped** function determines whether the data that was previously requested by means of a **S5StarteLesenOverlapped** function is available. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

You must continue calling this function until you receive the expected data or an error message that indicates a connection error.

Parameters

s5 pointer to a structure of the type `S5_PARAMS`. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

The function will either return a 0 or an H1 error.

If the returned value is 0 you can access the received data as follows:

```
PLCData[0]=(UCHAR*)s5+sizeof(S5_PARAMS)+0;
PLCData[1]=(UCHAR*)s5+sizeof(S5_PARAMS)+1;
```

Tab. 7-9 contains an overview of the H1 errors and their description.

Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the S5 error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

UserFctCall pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).

The User-Function must be defined as follows:

```
void WENTRY_C UserCallback(S5PARAMS W_POINTER
s5);
```

Timeout_ms the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.

Returns

0
H1_BAD_LINE
H1_WAIT_DATA
H1_NO_DRIVER

See also

S5LeseAusSPS
S5StarteLesen

Flowchart "Read from PLC"

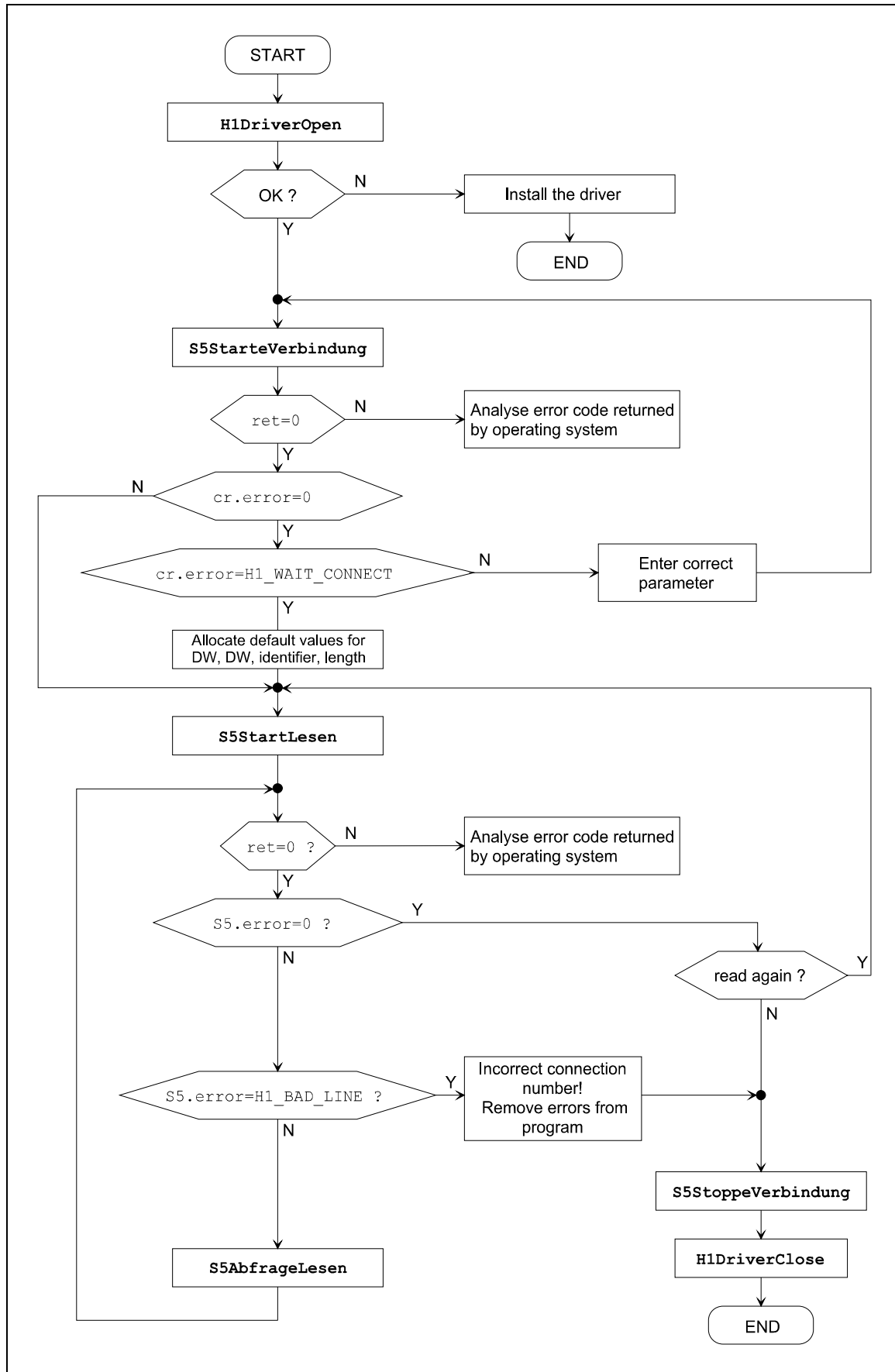


Fig. 7-7: Flowchart "Read from PLC"

7.6.3.3.2 Write Active

Write to PLC

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5SchreibeInSPS(S5_PARAMS W_POINTER s5)
```

The function **S5SchreibeInSPS** is used when data must be written into the PLC and when the function should not be interrupted.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**. You must allocate the required amount of memory after the **s5** structure:
s5=malloc(sizeof(S5_PARAMS)+SENDLEN);

Write data can be supplied as follows:

```
memcpy((UCHAR*)s5+sizeof(S5_PARAMS),
        &PCData,SENDLEN);
```

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

Returns

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

Note

The function returns control only when the data is complete or when an error is detected. This may require some time depending on the connection parameters as well as the responses from the other station. This function can not be interrupted.

See also

```
S5StarteVerbindung      S5StarteVerbindungAdapter
S5StartSchreiben       S5AbfrageSchreiben
S5StoppeVerbindung
```

Write to PLC and continue (asynchronously)

```
#include <S5ACCESS.H>
USHORT WENTRY_C  S5SchreibeInSPSOverlapped(
    S5_PARAMS W_POINTER s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5SchreibeInSPSOverlapped** is used to write data into the PLC. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

Parameters

- | | |
|-------------|--|
| s5 | <p>pointer to a structure of the type <code>S5_PARAMS</code>. The values Kennung, DB, DW and Len of the s5 structure must contain valid entries. s5 must contain a valid connection number which was returned by the function S5StarteVerbindung. You must allocate the required amount of memory for received data after the s5 structure:</p> <pre>s5=malloc(sizeof(S5_PARAMS)+SENDLEN);</pre> <p>Write data can be supplied as follows:</p> <pre>memcpy((UCHAR*)s5+sizeof(S5_PARAMS), &PCData,SENDLEN);</pre> <p>The function will either return a 0 or an H1 error.</p> <p>Tab. 7-9 contains an overview of the H1 errors and their description.</p> <p>Any S5 errors returned by the other station are located in Fehler. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in Fehler indicates that the data transfer was completed properly.</p> |
| UserFctCall | <p>pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).</p> <p>The User-Function is defined as follows:</p> <pre>void WENTRY_C UserCallback(S5PARAMS W_POINTER s5);</pre> |
| Timeout_ms | <p>the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.</p> |

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung
S5StartSchreiben
S5StoppeVerbindung
S5AbfrageSchreiben

Start write operation

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5StartSchreiben(S5_PARAMS W_POINTER s5)
```

The **S5StartSchreiben** function initialises a write operation.

You may not start more than one start write operations for a single connection.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**. You must allocate the required amount of memory for received data after the **s5** structure:

```
s5=malloc(sizeof(S5_PARAMS)+SENDLEN);
```

Write data can be supplied as follows:

```
memcpy((UCHAR*)s5+sizeof(S5_PARAMS),
        &PCData,SENDLEN);
```

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

If the function returns a 0 you can check for completion of the operation by polling with **S5AbfrageSchreiben**.

Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung
S5SchreibeInSPS
S5AbfrageSchreiben
S5StoppeVerbindung

Start write operation and continue (asynchronously)

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5StartSchreibenOverlapped(
    S5_PARAMS W_POINTER s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5StartSchreibenOverlapped** function initialises a write operation. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

You must not issue multiple start write operations for a connection.

Parameters

<code>s5</code>	<p>pointer to a structure of the type <code>S5_PARAMS</code>. The values Kennung, DB, DW and Len of the s5 structure must contain valid entries. s5 must contain a valid connection number which was returned by the function S5StarteVerbindung. You must allocate the required amount of memory for received data after the s5 structure:</p> <pre>s5=malloc(sizeof(S5_PARAMS)+SENDLEN);</pre> <p>Write data can be supplied as follows:</p> <pre>memcpy((UCHAR*)s5+sizeof(S5_PARAMS), &PCData,SENDLEN);</pre> <p>The function will either return a 0 or an H1 error.</p> <p>Tab. 7-9 contains an overview of the H1 errors and their description.</p> <p>If the function returns a 0 you can check for completion of the operation by polling with <code>S5AbfrageSchreiben</code>.</p> <p>Any S5 errors returned by the other station are located in Fehler. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in Fehler indicates that the data transfer was completed properly.</p>
<code>UserFctCall</code>	<p>pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).</p> <p>The User-Function is defined as follows:</p> <pre>void WENTRY_C UserCallback(S5PARAMS W_POINTER s5);</pre>
<code>Timeout_ms</code>	<p>the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.</p>

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung
S5SchreibeInSPS
S5AbfrageSchreiben
S5StoppeVerbindung

Poll a write operation

```
#include <S5ACCESS.H>
```

```
USHORT WENTRY_C S5AbfrageSchreiben(S5_PARAMS W_POINTER s5)
```

The **S5AbfrageSchreiben** function is used to check whether data that was previously transferred by means of an **S5StartSchreiben** function has been sent or not.

This function must be repeated until the data has been sent successfully or until an error message indicating a faulty connection is received.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

Returns

0
H1_BAD_LINE
H1_WAIT_SEND
H1_NO_DRIVER
H1_NO_ADAPTER

See also

S5StarteVerbindung
S5StarteVerbindungAdapter
S5SchreibeInSPS
S5StartSchreiben

Poll a write operation and continue (asynchronously)

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5AbfrageSchreibenOverlapped(
    S5_PARAMS W_POINTER s5,
    void (WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5),
    long Timeout_ms)
```

The **S5AbfrageSchreibenOverlapped** function is used to check whether data that was previously transferred by means of an **S5StartSchreibenOverlapped** function has been sent or not. Internally, this function employs the overlapped mechanism to access the function `UserFctCall` cyclically while you are waiting for an event.

Parameters

s5 pointer to a structure of the type `S5_PARAMS`. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

Any S5 errors returned by the other station are located in **Fehler**. Chapter 7.6.3.4 contains an overview of the PLC error codes. A 0 in **Fehler** indicates that the data transfer was completed properly.

UserFctCall pointer to a function located in the application program. This is accessed at a rate of 10ms by the DLL while the function is being processed. This mechanism returns control to your program while the overlapped function is being executed (see chapter 7.6.1.1.2).

The User-Function is defined as follows:

```
void WENTRY_C UserCallback(S5PARAMS W_POINTER
s5);
```

Timeout_ms the time limit for completion of the function by the DLL (see chapter 7.6.1.1.2). The connection will be terminated if a timeout occurs.

Returns

```
0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
```

See also

```
S5StarteVerbindung
S5SchreibeInSPS
S5StartSchreiben
```

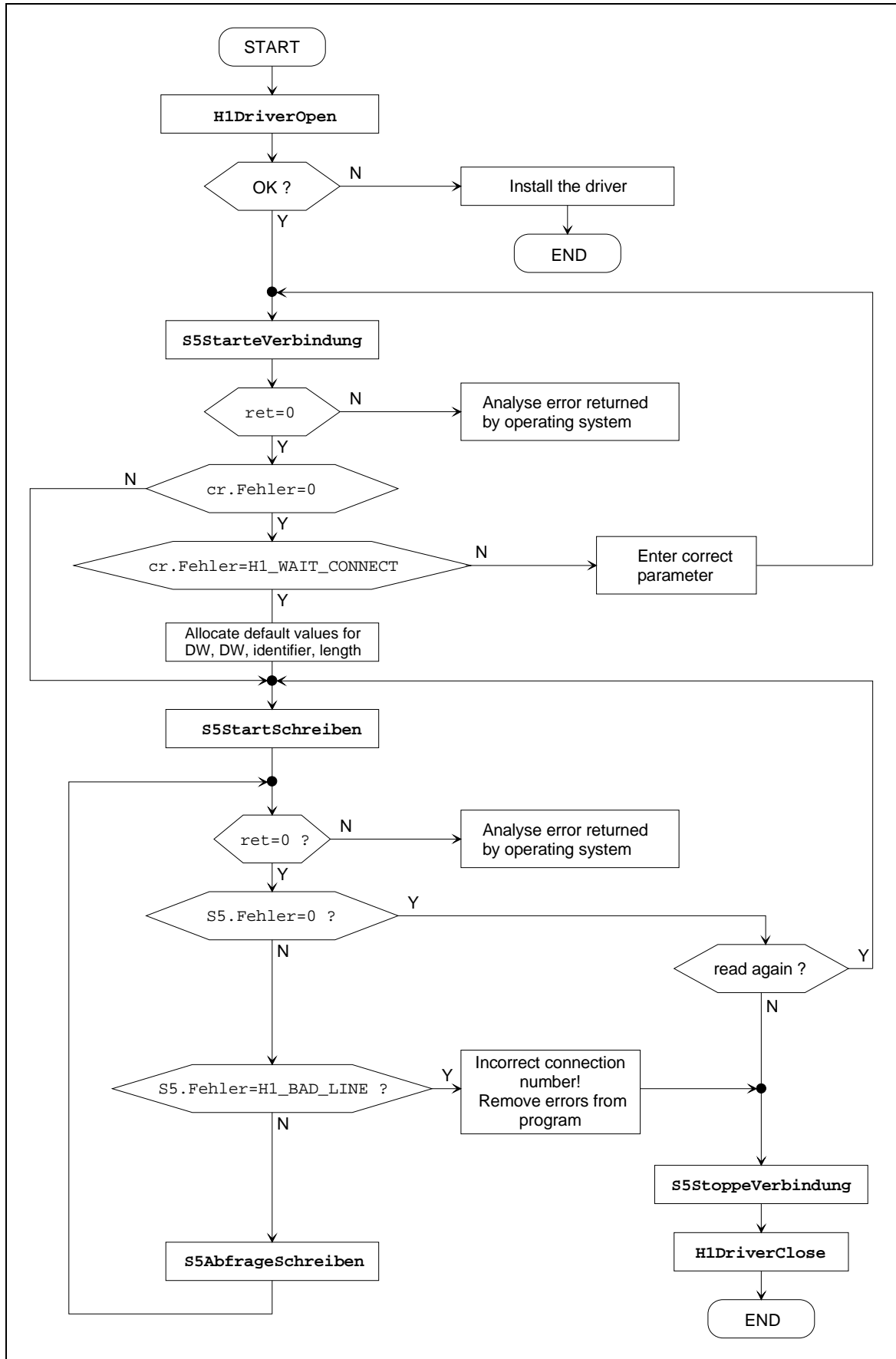
Flowchart for "Write to PLC"

Fig. 7-8: Flowchart "Write to PLC"

7.6.3.3.3 Fetch passive

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5FetchPassiv(S5_PARAMS W_POINTER s5, S5_PARAMS
W_POINTER(WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5))
```

The **S5FetchPassiv** function sends the data that a remote station requested by means of an active fetch task.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure must contain valid entries. **s5** must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

Any data must be entered into a memory allocation located after the **s5** structure. You should allocate the maximum amount of memory required for the requested data before calling the function. If the amount of data referenced by **s5** is not available before the function is initiated you may reallocate memory once the amount has become available.

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

UserFctCall

pointer to the call-back function. Once the request has been received from the remote station the requested data is supplied to this function. The requested data is located in **Kennung**, **DB**, **DW**, and **Len** of the **s5** structure that the initial function call referred to. If the request from the remote station is not valid the function will not be initiated and **S5FetchPassiv** is terminated by an error. The remote station can be informed of errors (e.g. invalid data requested) by means of **Fehler** in the call-back function. Any errors are returned to the requesting remote station as S5 error code, otherwise the data bytes are reversed and transferred to the remote station. The data must be located in memory after the **s5** structure. The caller must supply the data by means of references contained in **DB**, **DW** and **Kennung**. The length of data is calculated from **Kennung** and **Len**.

The User-Function is defined as follows:

```
S5PARAMS W_POINTER WENTRY_C UserCallback(
S5PARAMS W_POINTER s5);
```

Note

The call-back function must return a value consisting of a valid pointer to an **s5** structure. This structure may contain the same address as the received structure or another address that you have reallocated. Reallocation is one way of dynamically managing the memory space required for data.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NO_MEMORY

7.6.3.3.4 Write Passive

```
#include <S5ACCESS.H>
USHORT WENTRY_C S5WritePassiv(S5_PARAMS W_POINTER s5,
void(WENTRY_C *UserFctCall)(S5_PARAMS W_POINTER s5))
```

The **S5WritePassiv** function accepts the data that was sent by an active write from the remote station.

Parameters

s5 pointer to a structure of the type **S5_PARAMS**. The remote station has supplied the values **Kennung**, **DB**, **DW** and **Len** of the **s5** structure.

s5 must contain a valid connection number which was returned by the function **S5StarteVerbindung**.

In this case the type of connection in **H1_CONNECT_PARAMS** must be set to **NORMAL_LINE** | **PASSIV_LINE**.

Any data must be entered into a memory allocation located after the **s5** structure. You should allocate the maximum amount of memory required for the requested data before calling the function. If the amount of data referenced by **s5** is not available before the function is initiated you may reallocate memory once the amount has become available.

The function will either return a 0 or an H1 error.

Tab. 7-9 contains an overview of the H1 errors and their description.

UserFctCall

pointer to the call-back function. Once the request has been received from the remote station the requested data is supplied to this function. The requested data is located in **Kennung**, **DB**, **DW**, and **Len** of the **s5** structure that the initial function call referred to. If the request from the remote station is not valid the function will not be initiated and **S5WritePassiv** is terminated by an error. The remote station can be informed of errors (e.g. invalid data requested) by means of **Fehler** in the call-back function. Any errors are returned to the requesting remote station as S5 error code, otherwise data will be returned. The data must be located in memory after the **s5** structure. During the transfer the data bytes will be reversed. The caller must supply the data by means of references contained in **DB**, **DW** and **Kennung**. The length of data is calculated from **Kennung** and **Len**.

The User-Function is defined as follows:

```
S5PARAMS W_POINTER WENTRY_C UserCallback(
S5PARAMS W_POINTER s5);
```


Note

The call-back function must return a value consisting of a valid pointer to an **s5** structure. This structure may contain the same address as the received structure or another address that you have reallocated. Reallocation is one way of dynamically managing the memory space required for data.

Returns

0
H1_BAD_LINE
H1_NO_DRIVER
H1_NO_ADAPTER
H1_NO_MEMORY

7.6.3.4 PLC error codes

A function returning a value of 0 indicates that `s5Fehler` contains the error code received from the remote station. The following table contains a listing of these error codes.

Code	Description
0	no error
1	bad Q/ZTYP at the handler module
2	PLC-area not available (DB not present)
3	PLC-area too small
4	QVZ-error occurred in PLC
5	error in display word (ANZW)
6	invalid ORG format
7	no data buffers available
8	no unused transport connections
9	error at the remote station
A	connection error (connection was terminated or could not be established)
B	message error (firmware error)
C	initialisation error (e.g. RECEIVE to SEND)
D	termination after RESET
E	task with READ/WRITE (no initialisation from PLC possible)
F	task does not exist
FF	system error

Tab. 7-10: Transport error codes

7.6.4 PLC Net file functions

7.6.4.1 General functions

7.6.4.1.1 Set NET.NET filename

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5SetzeNetDateiname(char W_POINTER
Dateiname);
```

This function specifies a new value for the name and path of the NET.NET file.

Parameter s

Dateiname	pointer to a memory area containing the new filename and path. All Net-file functions that do not refer to a specific filename will use this new filename.
-----------	--

Returns

0	OK, name was accepted
1	filename too long

Note

The specified path must exist but the file is not required. A new file will be created if the file does not exist. Any existing file with the same name will be overwritten.

7.6.4.1.2 Read NET.NET filename

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
void WENTRY_C S5GetNetDateiname(char W_POINTER Dateiname);
```

This function returns the filename of the Net.Net file.

Parameters

Dateiname	pointer to a memory area containing the current filename and path.
-----------	--

7.6.4.2 Layer 7 Net file functions

The following functions refer to the parameter file described in the appendix. These functions are divided into two groups:

- File functions referring to the standard parameter file. This is the Net .net file.
- File functions referring to a file whose name was specified.

File functions for the standard parameter file

Net .net is the standard parameter file and it is located in the root directory of drive C:. The name and path of this file can be changed at any time. To do this, proceed as follows:

1. Set the variable CONNETFILE in your system. This variable modifies the name and path of the file.

Example: The file name is H1 .net and it is located in the directory C:\H1

```
SET CONNETFILE = C:\H1\H1.net
```

2. Call the following function:

```
S5SetzeNetDateiname(char W_POINTER Dateiname)
```

File function specifying the filename

The filename of the parameter file accompanies the function call.

7.6.4.2.1 Get connection parameters from standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5HoleVerbindungsparameter(
    S5_CONNECTIONDATA W_POINTER s5daten)
```

A number of parameters are required when establishing a connection by means of the function **S5StarteVerbindung**. To allow for the modification of parameters in an application program these values may be retrieved from a file by means of the function **S5HoleVerbindungsparameter**.

You may use an ASCII editor to create and to modify this file. Chapter 7.6.5.1 describes the format of this file.

Parameters

s5daten	a pointer to the structure S5_CONNECTIONDATA. The element Connectionname must contain a valid name. The connection name is the section name in the connection file.
---------	---

Returns

0	OK, data is located in the structure.
TEXT_NO_NET_FILE	Connection file does not exist.
TEXT_NO_SECTION	Connection name could not be found.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5StarteVerbindung
S5StoppeVerbindung
S5SchreibeVerbindungsparameter
S5HoleVerbindungsparamsKarte
S5SchreibeVerbindungsparamsKarte
S5ListeVerbindungen

7.6.4.2.2 Get connection parameters from standard file for multiple adapters

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5HoleVerbindungsparamsKarte(
    S5_CONNECTIONDATA_ADAPTER W_POINTER s5daten)
```

A number of parameters are required if, for instance, a connection should be established by means of the function **S5StarteVerbindungAdapter**. To allow for the modification of parameters in an application program these values may be retrieved from a file by means of the function **S5HoleVerbindungsparameters**.

You may use an ASCII editor to create and to modify this file. Chapter 7.6.5.1 describes the format of this file.

Parameter

s5daten	a pointer to the structure S5_CONNECTIONDATA. The element Connectionname must contain a valid name. The connection name is the section name in the connection file.
---------	---

Returns

0	OK, data is located in the structure.
TEXT_NO_NET_FILE	Connection file does not exist.
TEXT_NO_SECTION	Connection name could not be found.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5StarteVerbindung
S5StoppeVerbindung
S5HoleVerbindungsparameter
S5SchreibeVerbindungsparameter
S5SchreibeVerbindungsparamsKarte
S5ListeVerbindungen

7.6.4.2.3 Write connection parameters into standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5SchreibeVerbindungsparameter(
    S5_CONNECTIONDATA W_POINTER s5daten)
```

In a program providing a user interface for modification of the configuration of the H1 connection any data entered can be saved by means of **S5SchreibeVerbindungsparameter**. This function uses the same file as described for **S5HoleVerbindungsparameter**.

Parameters

s5daten	is a pointer to the structure S5_CONNECTIONDATA. All elements must contain valid settings.
---------	---

Returns

0	OK, the data is located in the file.
TEXT_NO_MEM	Not enough memory available for processing the function.
TEXT_DISK_FULL	General disk write error.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5StarteVerbindung
S5StoppeVerbindung
S5HoleVerbindungsparameter
S5HoleVerbindungsparamsKarte
S5SchreibeVerbindungsparamsKarte
S5ListeVerbindungen

7.6.4.2.4 Write connection parameters into standard file for multiple adapters

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5SchreibeVerbindungsparamsKarte(
    S5_CONNECTIONDATA_KARTE W_POINTER s5daten)
```

In a program providing a user interface for modification of the configuration of the H1 connection any data entered can be saved by means of **S5SchreibeVerbindungsparamsKarte**. This function uses the same file as described for **S5HoleVerbindungsparameter**.

Parameters

s5daten	is a pointer to the structure S5_CONNECTIONDATA_ADAPTER. All elements must contain valid settings.
---------	--

Returns

0	OK, the data is located in the file.
TEXT_NO_MEM	Not enough memory available for processing the function.
TEXT_DISK_FULL	General disk write error.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5StarteVerbindung
S5StoppeVerbindung
S5HoleVerbindungsparameter
S5HoleVerbindungsparamsKarte
S5SchreibeVerbindungsparameter
S5ListeVerbindungen

7.6.4.2.5 Get list of connections from standard file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5ListeVerbindungen(unsigned short len,
      char W_POINTER mem)
```

The **S5ListeVerbindungen** function returns a list containing all the names of the connections that are currently located in the file. This may, for instance, be used to check whether a specific connection exists or not.

Parameters

len	specifies the length of the memory area used for the transfer.
mem	is the pointer to the memory area. The allocated length must equal or exceed the setting in len.

Returns

0	OK, memory now contains the connection names separated by null terminators and a byte. Bit 0: the connection is in use if this is set to 1. Bit 1: always 0 Bit 2: a 1 indicates an H1 connection Bit 3: a 1 indicates a TCP/IP connection Bit 4: a 1 indicates a send connection (SEND DIREKT) Bit 5: a 1 indicates a receive connection (RECEIVE DIRECT) The list ends with two null terminators.
TEXT_NO_NET_FILE	the connection file does not exist.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5StarteVerbindung
S5StoppeVerbindung
S5HoleVerbindungsparameter
S5HoleVerbindungsparamsKarte
S5SchreibeVerbindungsparameter
S5SchreibeVerbindungsparamsKarte

7.6.4.2.6 Get a list of connections from the specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5ListeNetVerbindungen(char W_POINTER
        Dateiname, short len, char mem)
```

The function **S5ListeNetVerbindungen** returns a list with all the names of the connections that have been entered into the file. You may use this function to check whether a specific connection exists or not..

Parameters

Dateiname	the name of the Net file from which the list of connections must be retrieved.
len	specifies the length of the memory area used for the transfer.
mem	is the pointer to the memory area. The allocated length must equal or exceed the setting in len.

Returns

0	OK, memory now contains the connection names separated by null terminators and a byte. Bit 0: the connection is in use if this is set to 1. Bit 1: always 0 Bit 2: a 1 indicates an H1 connection Bit 3: a 1 indicates a TCP/IP connection Bit 4: a 1 indicates a send connection (SEND DIRECT) Bit 5: a 1 indicates a receive connection (RECEIVE DIRECT) The list ends with two null terminators.
TEXT_NO_NET_FILE	the connection file does not exist.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

H1LeseParameter
H1SchreibeParameter
S5LeseParameter
S5SchreibeParameter

7.6.4.2.7 Read H1 parameter record from specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C H1LeseParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    H1_CONNECT_PARAMS W_POINTER cr)
```

Reads a record of H1 parameters from the specified .Net file.

Parameters

netfile	the name of the .Net file.
vname	the name of the respective connection.
cr	the structure where the parameters should be saved.

Returns

0	OK, data is located in the structure.
TEXT_NO_NET_FILE	Connection file does not exist.
TEXT_NO_SECTION	Connection name could not be found.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5ListeNetVerbindungen
H1SchreibeParameter
S5LeseParameter
S5SchreibeParameter

7.6.4.2.8 Write H1 parameter record to the specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C H1SchreibeParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    H1_CONNECT_PARAMS W_POINTER cr)
```

Writes one record of H1 parameters into the specified .Net file.

Parameters

netfile	the name of the .Net file.
vname	the name of the respective connection.
cr	the structure where the parameters should be saved.
All parameters must contain valid entries.	

Returns

0	OK, data is located in the structure.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5ListeNetVerbindungen
H1LeseParameter
S5LeseParameter
S5SchreibeParameter

7.6.4.2.9 Read PLC parameter record from the specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5LeseParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    S5_ANSCHALTUNG W_POINTER s5)
```

Reads a record of H1 parameters from the specified .Net file.

Parameters

netfile	the name of the .Net file.
vname	the name of the respective connection.
s5	the structure where the parameters should be saved.

Returns

0	OK, data is located in the structure.
TEXT_NO_NET_FILE	Connection file does not exist.
TEXT_NO_SECTION	Connection name could not be found.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5ListeNetVerbindungen
H1LeseParameter
H1SchreibeParameter
S5SchreibeParameter

7.6.4.2.10 Write PLC parameter record to the specified file

```
#include <MSGINDEX.H>
#include <S5ACCESS.H>
unsigned short WENTRY_C S5SchreibeParameter(
    char W_POINTER netfile,
    char W_POINTER vname,
    S5_ANSCHALTUNG W_POINTER s5)
```

Writes one record of H1 parameters into the specified .Net file.

Parameter

netfile	the name of the .Net file.
vname	the name of the respective connection
s5	the structure where the parameters should be saved.
All parameters must contain valid entries	

Returns

0	OK, data is located in the structure.
TEXT_NO_MEM	Not enough memory available for processing the function.

Note

This function makes use of files. It is therefore not possible to call the function from a driver.

See also

S5ListeNetVerbindungen
H1LeseParameter
H1SchreibeParameter
S5LeseParameter

7.6.5 Files, constants and structures

7.6.5.1 Description of the parameter file

The standard name for the parameter file is `Net.net`. It contains a number of sections, each consisting of a group of related parameters. The sections and settings in the `Net.net` file have the following format:

[Sectionname]
Keyname=value

Here the **[Sectionname]** is the name of a section.

The square brackets ([]) enclosing the name are required and the left-hand bracket must be located on the left-hand border.

The **Keyname=value** entry defines the value of each parameter. The key name is the name for the respective parameter. These names may consist of an arbitrary combination of letters and numbers followed immediately by the equals character (=). The value depends on the respective parameter. It may contain a real number, a string of characters or a string of characters enclosed in quotation marks. Most sections have a number of different parameters.

You may also include comments in the initialisation file. Every comment line must begin with a semicolon (;).

[EigeneAdresse]	(own address) section name containing station parameters.
KachelBasis=	(page frame base) page frame base address of the local (own) station
Stationsname=	(Station name) name of the local station
Adresse=	(address) the address of the local station

The following sections of `Net.net` are connection names.

[Name der Verbindung]	(connection name) section name of a connection, where the connection parameters are located.
------------------------------	--

The following variables are acceptable for a connection section:

Leitungstyp= Ethernet

(line type = ethernet) this connection is based on an ethernet H1 connection.

EthernetKarte=

(ethernet adapter) the number of the adapter that must be used. This number starts at 1 and ends with the number of the last adapter that has been installed.

EthernetStation= aabbccddeeff

Ethernet address of the **addressed** station. Station addresses are always 6 Bytes in length. You must specify 12 numbers. This entry is a HEX number

Example: EthernetStation=080006010001h

EthernetNSAP=

NSAP of the addressed station. NSAP is not implemented on Siemens H1 systems.

EthernetEigenerTSAP=

(EthernetOwnTSAP) Own TSAP. A TSAP can be from 2 to 16 characters in length. This entry is a HEX number.

Example: EthernetEigenerTSAP=2020202030202031h

EthernetFremderTSAP=

(EthernetRemoteTSAP) Remote TSAP. A TSAP can be from 2 to 16 characters in length. This entry is a HEX number.

Example: EthernetFremderTSAP=2020202030202031h

EthernetStartParameter=

Additional connection parameters. These parameters are used internally by the destination system. The effect of the parameters depends on the destination system. This entry is a HEX number. Example: EthernetStartParameter=20303436h

EthernetLeistungsart=

(Ethernet line type) Defines the type of operation. Accepted values are "Normal", "Normal, Aktiv", "Normal, Passiv", "Broadcast", "Multicast" and "Datagramm". "Normal" on its own means "Normal, Aktiv". Example: EthernetLeistungsart=Normal,Aktiv

EthernetPriority=

The priority of the connection. Industrial Ethernet (H1) accepts the following:

Prio0, Prio1 "Express" priority (up to 16 bytes of data)

Prio2, Prio3 normal priorities

Prio4 lowest priority.

Example: EthernetPriority=Prio2

EthernetMulticastkreis=

(EthernetMulticastCircuit) This contains the multicast circuit number for multicast connections. For any other type of connection this parameter is ignored.

Example: EthernetMulticastkreis=2

The following parameters are provided for information purposes only, i.e. they have no influence on the operation:

Auftragsnummer= (task number)

AuftragsKachelOffset= (task page frame offset)

Auftragsart= (task type)

VerbindungBenutzt= (connection in use)

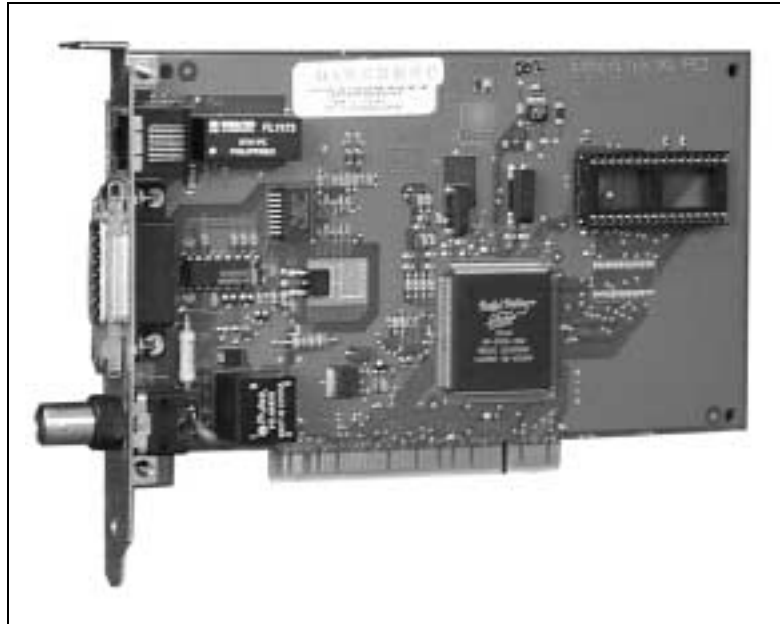
8 Supplement

Please find here the current product description which was not ready for print earlier.

PCI-Bus Network Card from 3COM

Hint

The PCI-Bus Network card from Z'nyx has been replaced by the PCI-Bus-Card from 3COM . The following is a discription of the PCI-Bus-3COM-Card from VIPA.



PCI-Bus-Card 3COM : (Best.-Nr.:VIPA SSN-BG88C)

The card supports the Plug- and Play-technology this means that it is fully integrated in the system without the user having to mesh with it.

Features

- Supports the Plug- and Play-technology this means that it is fully integrated in the system without the user having to mesh with it.
- "Auto Select Media Type" , means the automatic recognition of the jack that is connected with the network.
- Ethernet-connection via BNC, AUI and RJ-45 max. 10Mbit/s
- 32 Bit PCI-Bus

Scope of delivery

Hardware

- 32-Bit-PCI- Network Card
- BNC T-Stück

Software

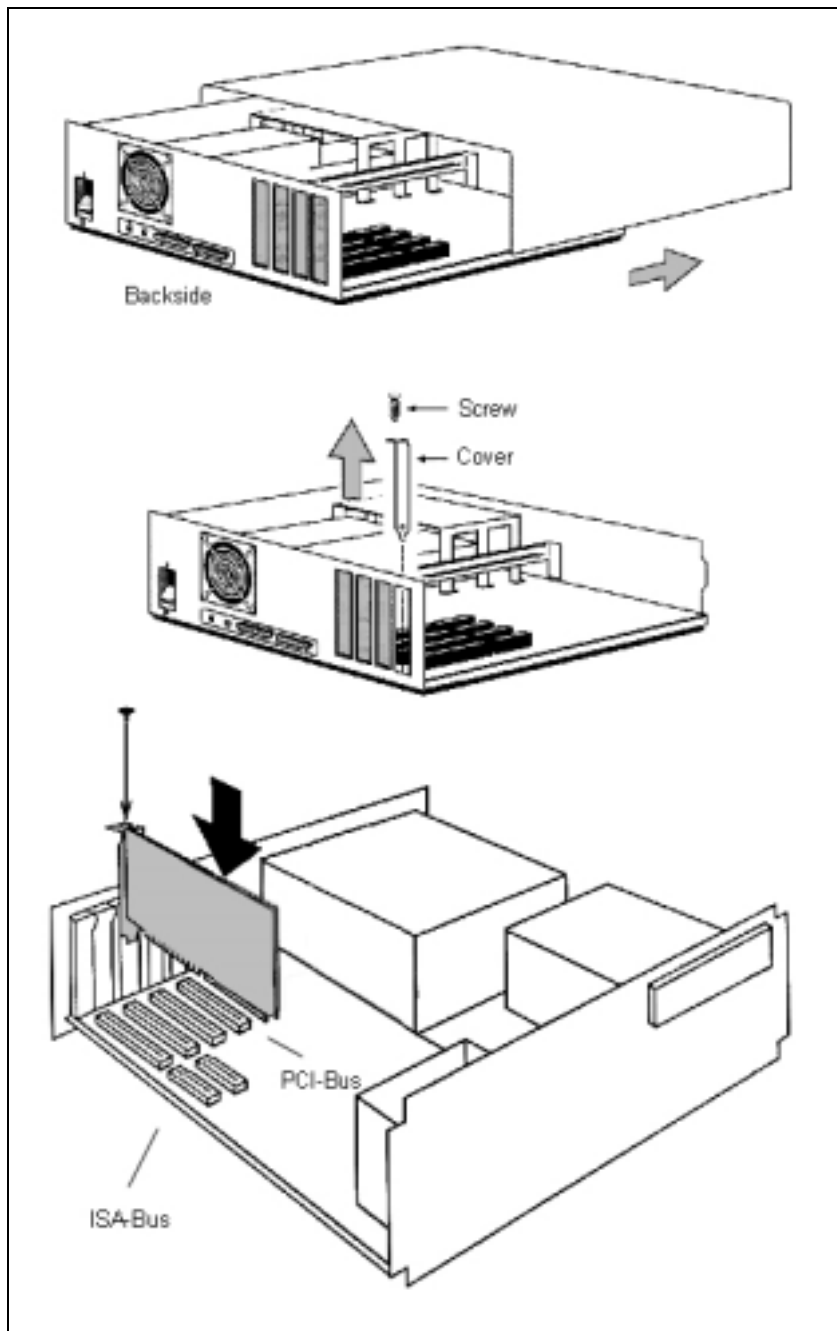
- **Disk 1: SW88C Disk 1/3** - Ether-driver for WIN98/95/NT
- **Disk 2: SW88C Disk 2/3** - NDIS 3/4/5, WfW 3.11
- **Disk 3: SW88C Disk 3/3** - DOS, Utils, Netware
- **Disk 4: SW83N Disk 1/1** - CP1413plus H1-driver for NT 3.5x and 4.0

Mounting of the card.

Before the software is installed the network card must be mounted in the computer.

Please pay attention to the following Hints:

1. Turn off the computer and remove the power cable.
2. Remove the cover of the computer according to the instructions of the manufacturer.
3. Remove the metal Slot-covers on the backside of the PC.
4. Place the card as shown in the diagram. Only one PCI-Slot may be used (connection rail).
5. Close the case and turn on the PC .



Mounting of the 3COM-Card

Software installation

Installation under Windows 98

The 3COM Network Card Driver is a fixed component of the Windows 98 driver library and therefore no installation is necessary after the mounting of the card and the new start.

The card is recognized after a new start and is fully automatically integrated in the system.

In the event of problems proceed according to the description under Windows 95.

Installation under Windows 95

This section describes how a card driver is installed under Windows 95. The driver supports 16-Bit-Applications.

Before the Installation

The following data is necessary for the installation. The data can be obtained from your system administrator:

- Network access (user name, password)
- Computer name
- Workgroup name
- Main server
- First local network (The letters are ordered in accordance with the mounted drives).

Windows 95 must be installed, the PCI-Network-Card mounted and connected to the network.

Driver-Installation

- Switch on the PC and start Windows 95.

Switch on the PC and start Windows 95. As soon as Windows 95 starts, the dialogue box states "Neue Hardware gefunden" ("New hardware found...") and follows with "Diskette einlegen" ("Insert disk").

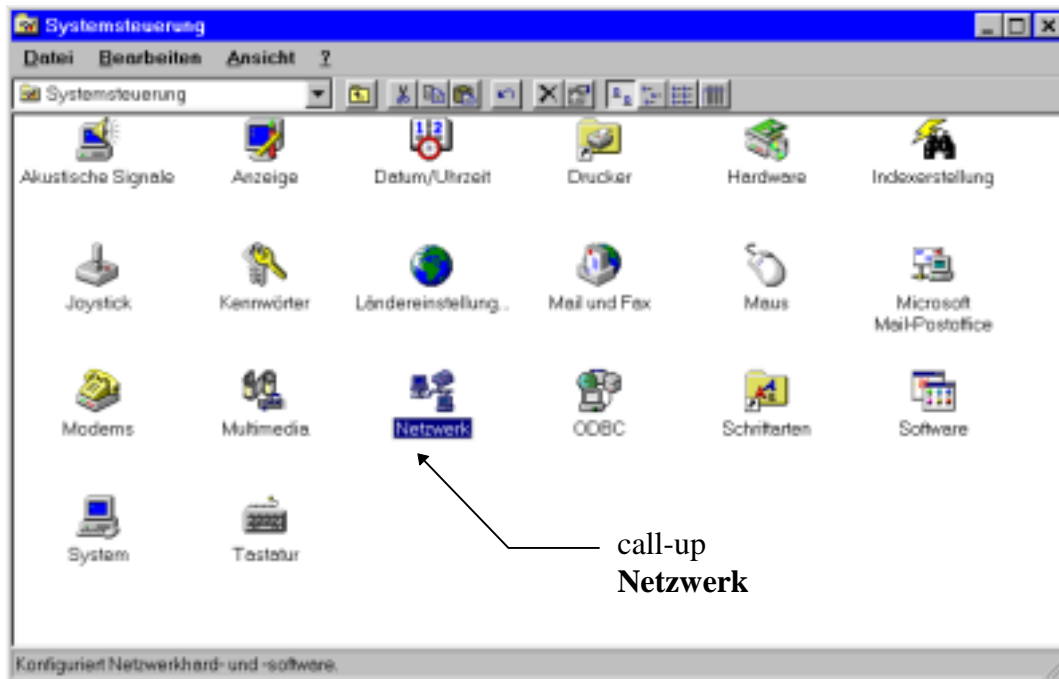
- Insert the disk "SW88C" (Disk 1/3) and start the installation.

Now Windows 95 transfers all necessary information to the hard-disk. During this installation it is requested that the above mentioned data be inputted.

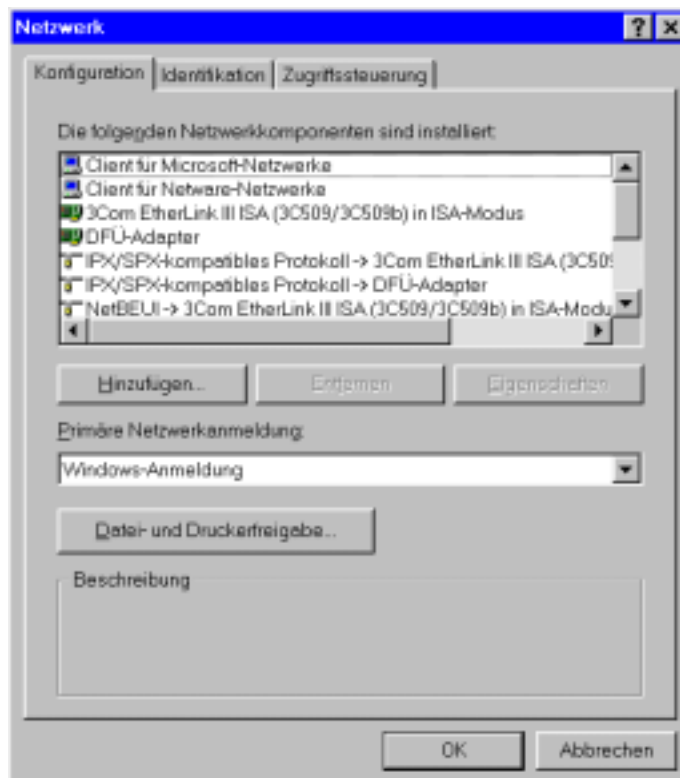
The network card should be registered in the network environment before the PC is booted anew.

Registration in the network

Go to the Windows 95 Start-Menü to the Menü-point "Einstellungen". Select here the "Systemsteuerung". The window *Systemsteuerung* will open.

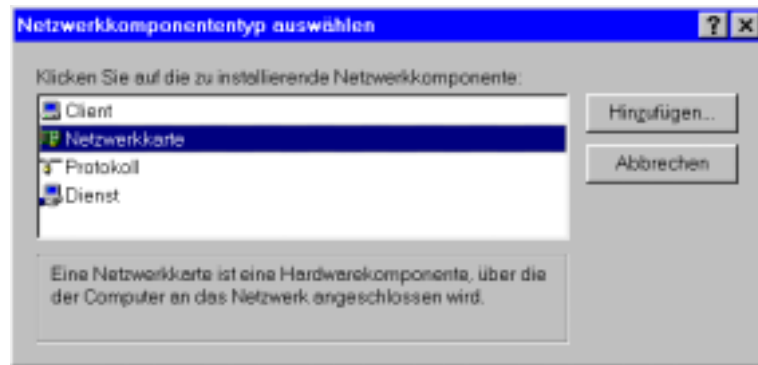


In this window call-up the programm "Netzwerk". The dialogue window *Netzwerk* will open.



Click here on [**Hinzufügen**].

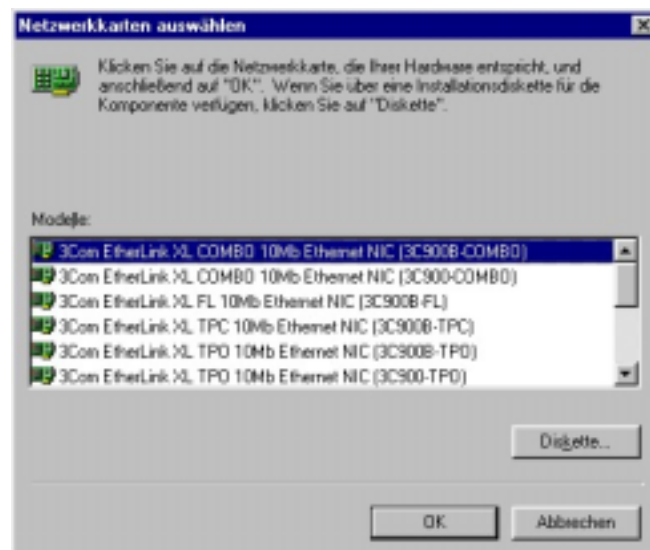
The window select "Netzwerkkomponententyp" (network component typ) will open. The VIPA-network-card can be entered here.



To do this select "Netzwerkkarte" and click **[Hinzufügen]**.

Insert the disk "SW88C" (Disk 1/3) und click **[Diskette]**. Select the disk-drive and click **[OK]**.

The driver list from 3COM will be displayed:



In the manufacturer list select the following card and click **[OK]**:

"3COM EtherLink XL COMBO (3C900-COMBO)"

or the newer version

"3COM EtherLink XL COMBO (3C900B-COMBO)"

All necessary data will now be transfered to the PC.

Start the PC anew. Do not forget to remove the disk from the drive.

The Network Card is now installed under Windows 95.

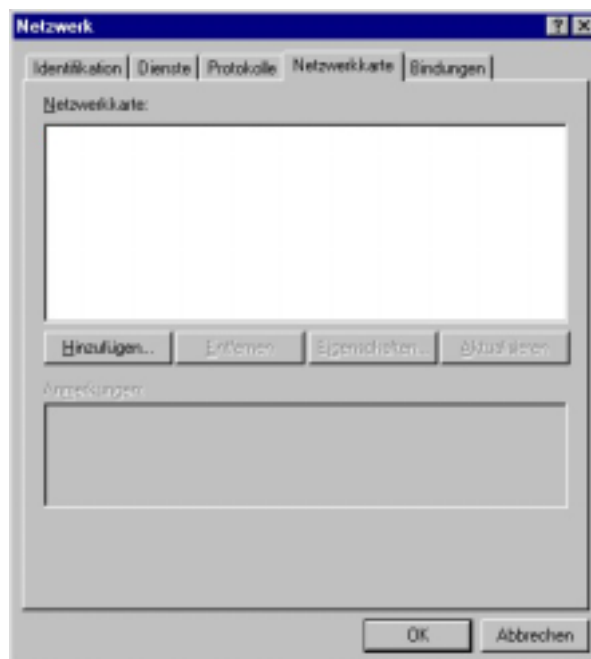
Installation under Windows NT

Start Windows NT and select under Start > Einstellung > Systemsteuerung "Netzwerk".

The following window will open.

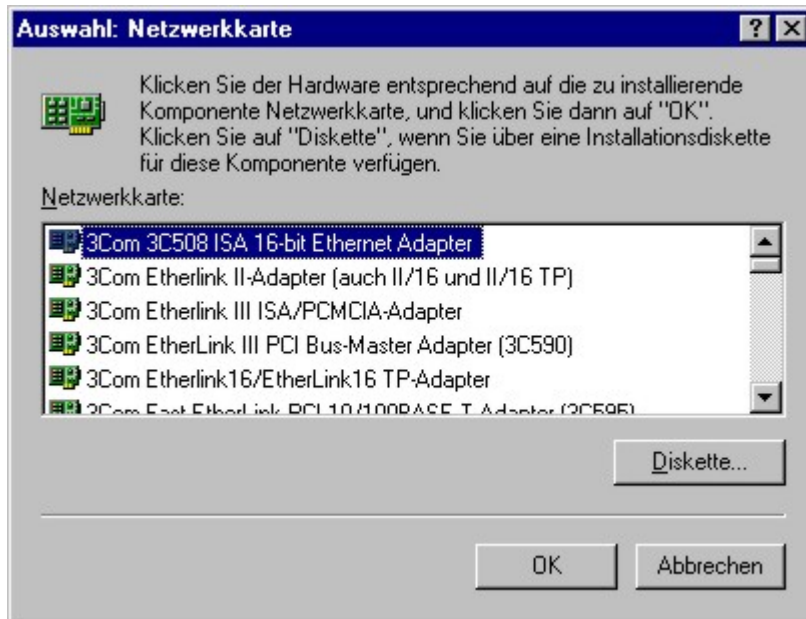


Insert the disk "SW88C" (Disk 1/3) and select the register "Netzwerkkarte".

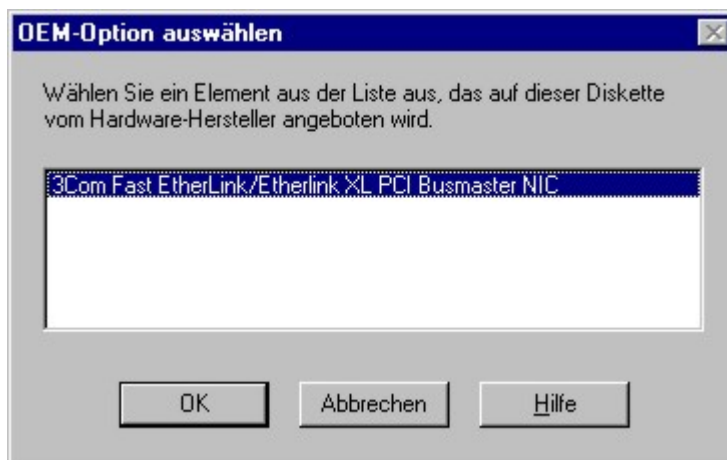


Click here **[Hinzufügen]**.

A selection of the network cards will be opened.



Click [**Diskette**].The following dialogue window will open with the spezified driver.



Select the driver and confirm with [**OK**].

All necessary data will now be transfered to the PC.

Start the PC anew. Do not forget to remove the disk from the drive.

The network card is now installed under Windows NT.

H1- drive under Windows NT

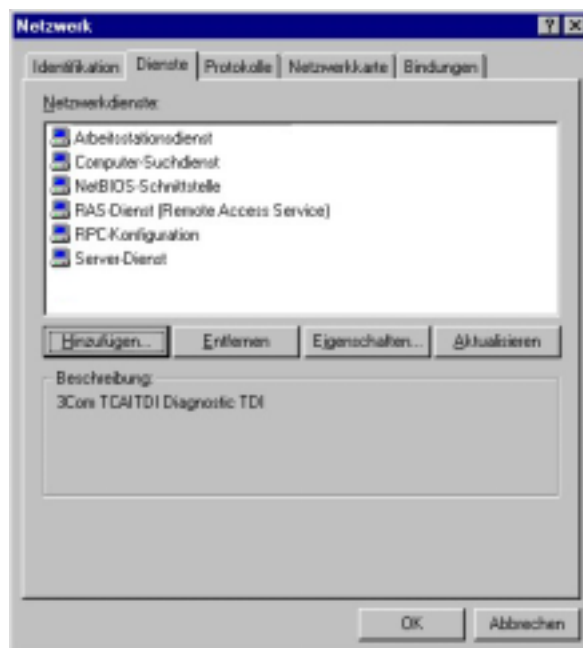
For the installation of the H1-driver under Windows NT the delivered disk"SSN-SW83N" (Disk 1/1) is necessary.

Select under Start > Einstellung > Systemsteuerung "Netzwerk".

The following window will open:

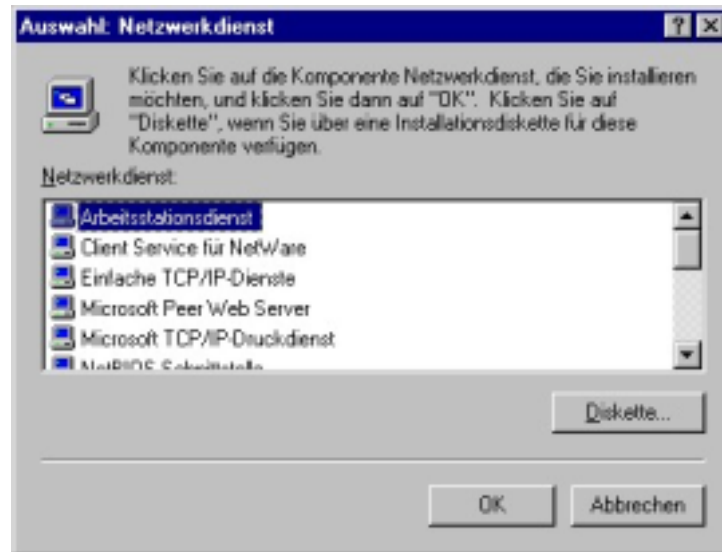


Insert the disk "SSN-SW83N" (Disk 1/1) and select the register"Dienste". The following dialogue window will open:

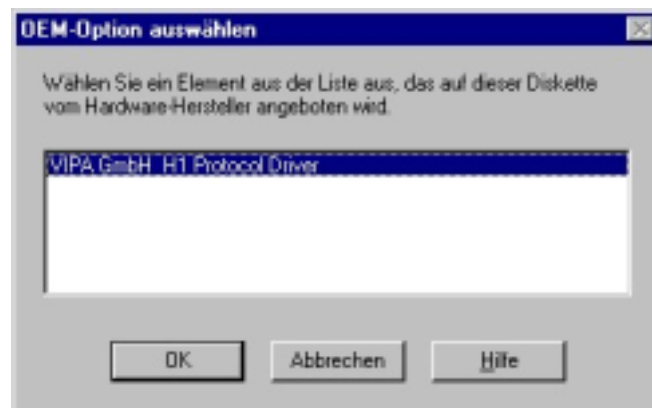


Click here [**Hinzufügen**].

A selection list will be opened.



Click on disk.



All necessary data will now be transferred to the PC.

Start the PC anew. Do not forget to remove the disk from the drive.

The H1-drive is now installed under Windows NT.

Technical Data

CP1413plus PCI-Bus (3COM - Card)

Supply voltage	DC 5 V \pm 5 % DC \pm 12 V
Current consumption	with 5 V max. 250 mA with 12 V max. 400 mA
Ethernet-Interface	AUI (IEEE 802.3i) BNC RJ-45
PC-Interface	32-Bit PCI-Bus Rev 2.0
Protection type	IP00 according to DIN 40050
Ambient Conditions	
- Operating temperature	0 °C ... 70 °C
- Storage temperature	-40 °C ... +70 °C
- Relative humidity	max. 90% no condensation
Constructive	
- Format	printed-circuit board AT-Format
L x H	175 mm x 100 mm
- Mounting width	1 SEP (ca. 15 mm)
- Space requirement in PC	1 mounting place
- Weight	ca. 0,3 kg

Appendix

A Technical data	A-1
B Abbreviations	B-1
C List of figures	C-1
D Index	D-1

Appendix

A Technical data

CP1413plus PCI-Bus (Z'nyx adapter)

Supply voltage	5 V DC \pm 5 %
Current consumption	1,2 A max. @ 5 V
Ethernet-Interface	AUI (IEEE 802.3) BNC, RJ-45
PC interface	32-Bit PCI-Bus Rev 2.0
Protection class	IP00 as per DIN 40050
Permissible environmental conditions	
- Operating temperature	0 °C ... 50 °C
- Storage temperature	-40 °C ... +70 °C
- Humidity	90% max. non-condensing
Construction	
- format	short AT-type board
L x H	136 mm x 100 mm
- Width	1 SEP (app. 15 mm)
- Space required in the PC	1 short slot
- Weight	app. 0,3 kg

CP1413plus ISA-Bus (3Com adapter)

Supply voltage	5 VDC \pm 5 % 12 VDC \pm 5 %
Current consumption max.	0,2 A max. @ 5 V 0,5 A @ 12 V
Ethernet-interface	AUI (IEEE 802.3) BNC, RJ-45
PC-Interface	16-Bit ISA-Bus
I/O-Base	200h - 3E0h in 10h-Schritten, EISA
Interrupt Request	3, 4, 5, 7, 9, 10, 11, 12, 15
Plug and Play	Yes
Protection class	IP00 as per DIN 40050
Permissible environmental conditions	
- Operating temperature	0 °C ... 70 °C
- Storage temperature	-40 °C ... +70 °C
- Humidity	90% max. non condensing
Construction	
- format	short AT-type board
L x H	160 mm x 100 mm
- Width	1 SEP (app. 15 mm)
- Space required in the PC	1 short slot
- Weight	app. 0,2 kg

B Abbreviations**A/B**

ABM	Asynchronous Balanced Mode
AFI	Authority and format identifier (specified by network administrator)
AK	Data Acknowledgement
AP	Application Protocol

C

CC	Connection Confirm
CCITT	Consultative Committee for International Telegraphy and Telephony
CR	Connection Request
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detect

D

DA	Destination Address
DC	Disconnect Confirm
DISC	Disconnect
DLC	Data Link Control
DM	Disconnect Mode
DR	Disconnect Request
DSP	Domain Specific Part (which station has been addressed)
DT	Data

E

EA	Expedited Data Acknowledgement
ED	Expedited Data
ER	Error (Report)

F

FCS	Frame Check Sequence
FRMR	Frame Reject Response

G/H

HDLC	High-Level Data Link Control
------	------------------------------

I/J/K

ID	Identifier
IDI	Initial Domain Identifier (which network has been addressed)
IEEE	Institute of Electrical and Electronic Engineers
IP	Internet Protocol
ISO	International Organisation for Standardisation

L

LAN	Local Area Network
LI	Length Indicator
LLC	Logical Link
LSB	Least Significant Bit

M

MAC	Media Access
MS	More Segments
MSB	Most Significant Bit

N/O

NRZ	Non Return to Zero
NSAP	Network Service Access Point

P/Q

PDU	Protocol Data Unit
-----	--------------------

R

R(E)J	Reject
RNR	Receiver Not Ready
RR	Receive Ready

S

SA	Source Address
SABME	Set Asynchronous Balanced Mode Extended
SAP	Service Access Points
SFD	Start Frame Delimiter
SP	Segmentation Permitted

T

TCP	Transmission Control Protocol
TOP	Technical and Office Protocols
TPDU	Transport Data Control Units
TSAP	Transport Service Access Point

U/V

UA	Unnumbered Acknowledge
UDP	User Datagram Protocol
UI	Unnumbered Information

W

WAN	Wide Area Network
-----	-------------------

X/Y/Z

XID	Exchange Identification
XNS	Xerox Network System

C List of figures

Fig. 1-1: CP1413plus for the PCI-Bus	1-3
Fig. 1-2: CP1413plus for the ISA-Bus	1-3
Fig. 2-1: Parts of an ethernet network	2-1
Fig. 2-2: Thin ethernet network hardware	2-4
Fig. 2-3: Example for a thin-ethernet cable network	2-5
Fig. 2-4: thick-ethernet network hardware	2-9
Fig. 2-5: example for a thick-ethernet network	2-10
Fig. 2-6: Adapters for linking unlike segments	2-14
Fig. 2-7: Example of a network combining thin and thick ethernet cable	2-15
Fig. 2-8: Star topology of a twisted pair network	2-16
Fig. 2-9: Twisted pair network hardware	2-17
Fig. 3-1: CP1413plus for PCI-bus	3-1
Fig. 3-2: CP1413plus for ISA-bus	3-1
Fig. 3-1: Installation of the Z'nyx adapter	3-4
Fig. 3-2: PCI adapter connection to a thin-ethernet-network	3-5
Fig. 3-3: PCI adapter connection to a thick ethernet-network	3-5
Fig. 3-4: PCI adapter connection to a twisted pair-network	3-6
Fig. 3-5: Installation of the 3Com adapter	3-32
Fig. 3-6: Selecting "On-board Coax (BNC)"	3-37
Fig. 3-7: Selecting "External (AUI/DIX)"	3-38
Fig. 3-8: Selecting "On-board TP (RJ-45) "	3-38
Fig. 4-1: The ethernet end-address of the Z'nyx-adapter (byte 4, 5, 6)	4-10
Fig. 4-1: The ethernet end-address of the 3Com-adapter (byte 4, 5, 6)	4-10
Fig. 4-2: Flowchart for "Send a telegram"	4-33
Fig. 4-3: Flowchart for "Read a telegram"	4-40
Fig. 4-4: Flowchart "Read from the PLC"	4-64
Fig. 4-5: Flowchart "Write to the PLC"	4-68
Fig. 7-1: H1 driver configuration for Windows NT 4.0	7-6
Fig. 7-2: Function diagram PLC overlapped functions	7-10
Fig. 7-3: The ethernet end address of the Z'NYX adapter (bytes 4, 5, 6)	7-20
Fig. 7-4: The ethernet end address of the 3COM adapter (bytes 4, 5, 6)	7-20
Fig. 7-5: Flowchart for "Transmitting a message"	7-42
Fig. 7-6: Flowchart for "Read a message"	7-54
Fig. 7-7: Flowchart "Read from PLC"	7-70
Fig. 7-8: Flowchart "Write to PLC"	7-79

D Index**3**

3COM adapter	3-30
Boot Prom	3-35
default settings	3-33
driver optimization	3-36
Guide	7-1
hardware installation	3-32
Interrupt Request Level	3-34
Maximum Modem	3-36
network cabling	3-37
Plug and Play	3-37
software installation	3-39
MS-DOS	3-40
OS/2	3-53
Windows 95	3-27; 3-59
Windows NT	3-57; 7-2; 7-3; 7-5
transceiver type	3-35

3COM-adapter	
guide	3-1

3COM-board	
locator	1-3

A

AbfrageSchreibenOverlapped	7-78
Access functions	1-5
Active	4-6; 7-16
Address	4-6; 7-17
AUI-connector	2-8

B

BNC-connectors	2-3
BNC-terminator	2-3
BNC-T-pieces	2-3

C

Check read	4-38; 7-51
Check read expedited data	4-39; 7-52
Check read operation	4-63; 7-67
Check Send	4-32
Check send status	7-41
Check write operation	7-77
Close a connection	7-27
Close all connections	7-29
Combination thin/thick ethernet cable	2-14
Combining thin/thick ethernet cable in networks	2-14
Communication functions	1-5
Connection	4-6; 7-16
Connection type	4-6; 7-16
Construction	1-4

D

Defining the ethernet address	7-20
Definition of call types	5-12
Definition of calling codes for drivers	5-10
Definition of the layer 7 interface	5-7
DestAddr	7-17
Destination address	4-6; 7-17
DestTSAP	
external	7-17

Determine station address	7-31; 7-60
Determine the version of the driver	7-24
Determining the driver version	4-15
Determining the ethernet address	4-10
Directed	7-6
Driver close	4-14; 7-23
Driver open	4-13; 7-22
DrvFCall.h	5-10

E

Error	7-18
Error codes of S5Fehler	4-71
Error messages	
.Fehler	4-9; 7-19
Establish a connection	7-25
Ethernet address	4-6; 7-17
Ethernet network-terminology	2-1
Example	
parameter file Net.net	6-10
PLC program	6-8
Example CP143plus parameter settings	6-9
Example H1 send	6-1
Example PLC reception	6-3

F

Fetch active	4-61; 7-62
Fetch Passive	4-69; 7-80
Filter	7-6
Flowchart	
Write to PLC	7-79
Flowchart	
Read	4-40; 7-54
Read from PLC	7-70
Read from the PLC	4-64
send	4-33
Transmit	7-42
Write to the PLC	4-68
Function - procedure	4-3; 7-11

G

Get a list of connections from the standard file	4-52
Get connection parameter	4-48
Get connection parameters	7-87
for multiple adapters	4-49; 7-88
Get H1 system values	4-26
Get line parameter	4-20
Get list of connections from file	4-53; 7-92
Get list of connections from standard file	7-91
Get revision levels	4-45; 7-59
Get station address	4-22
Get station address for multiple adapters	4-23
Guidelines	2-7

H

H1 Configuration	7-6
H1 error messages	7-19
H1 protocol driver	5-4
H1AbfrageLesen	7-51
H1AbfrageLesenEx	7-52
H1AbfrageSenden	7-41

H1CheckRead 4-38
 H1CheckReadEx 4-39
 H1CheckSend 4-32
 H1Def.h 5-4
 H1DriverClose 4-14; 7-23
 H1DriverOpen 4-13; 7-22
 H1GetLineparameter 4-20
 H1GetOverlappedResult 7-53
 H1GetStandardvalues 4-26
 H1GetStationAddress 4-22
 H1GetStationAddressCard 4-23
 H1GetVersion 4-15
 H1HoleStandardwerte 7-33
 H1HoleStationsAdresse 7-31; 7-60
 H1HoleVersion 7-24
 H1LeseDaten 7-43
 H1LeseDatenEx 7-45
 H1LeseParameter 7-93
 H1-parameter 4-6
 H1ReadData 4-34
 H1ReadDataEx 4-35
 H1ReadParameter 4-57
 H1SchreibeParameter 7-94
 H1SendData 4-29
 H1SendDataEx 4-30
 H1SendeDaten 7-36
 H1SendeDatenEx 7-37
 H1SetStandardvalues 4-27
 H1SetStationAddress 4-24; 4-55
 H1SetStationAddressCard 4-25; 4-56
 H1SetVector 4-28; 4-54
 H1SetzeStandardwerte 7-34
 H1SetzeStationsAdresse 7-32
 H1StartConnect 4-16
 H1StartConnectCard 4-17
 H1StarteLesen 7-46
 H1StarteLesenEx 7-49
 H1StarteLesenExOverlapped 7-50
 H1StarteLesenOverlapped 7-48
 H1StarteSenden 7-38
 H1StarteSendenExOverlapped 7-40
 H1StarteSendeOverlapped 7-39
 H1StarteVerbindung 7-25
 H1StarteVerbindungOverlapped 7-26
 H1StartRead 4-36
 H1StartReadEx 4-37
 H1StartSend 4-31
 H1StopConnect 4-18
 H1StoppeVerbindung 7-27; 7-30
 H1StoppeVerbindungen 7-29
 H1StoppeVerbindungOverlapped 7-28
 H1TestConnection 4-19; 4-21
 H1WriteParameter 4-58
 Hub 2-16

I

Initiate connection 4-16
 Initiate connections for multiple adapters 4-17

L

Layer 4 functions 4-1
 Layer 4 functions NT 7-7
 Layer 4 general functions 4-13; 7-22
 Layer 4 program interface 4-11
 Layer 4 programming interface 7-21
 Layer 4 specific functions 7-36
 Layer 7 file functions 4-47; 7-86
 Layer 7 functions 4-2
 Layer 7 functions NT 7-8
 Layer 7 general functions 4-42; 7-56
 Layer 7 program interface 4-41
 Layer 7 software interface 7-55
 Layer 7 specific functions 4-61; 7-62
 LenConnParams 7-17
 LenDestAddr 7-16
 LenDestTSAP 7-17
 LenNSAP 7-17
 LenOwnTSAP 7-17
 List Connections 4-21
 LSAP 7-16

M

MS-DOS vector 4-54
 Multicast 7-17
 Multicast circuit 4-7
 Multicast circuit-No. 4-7
 Multicast connections 4-7

N

Network planning 2-1
 NSAP 4-7; 7-17
 N-type connector 2-8
 N-type plug 2-8
 N-type-terminator 2-8

O

Operation 1-4
 Overlapped functions 7-9
 OwnTSAP
 own 7-17

P

Parameter file 5-1; 7-97
 Parameters of [H1PROT_NIF] 3-62
 Passive 4-6; 7-16
 PCI-bus 3-2
 Planning a network-layout 2-18
 Poll a write operation 4-67
 priority 4-6
 Programming
 general 4-1; 7-7
 Promiscuous Mode 7-6

R

Read data 4-34; 7-43
 Read expedited data 4-35; 7-45
 Read from PLC 4-61; 7-62
 Read H1 parameter record from file 4-57; 7-93
 Read Net.Net file name 4-46

- Read Net.Net filename..... 7-85
 Read PLC parameter record from file..... 4-59; 7-95
 Regulations 2-6; 2-11
S
 PLC error codes..... 7-84
 S5AbfrageLesen 7-67
 S5AbfrageLesenOverlapped 7-68
 S5AbfrageSchreiben 7-77
 S5Access.h..... 5-7
 S5CheckRead..... 4-63
 S5FetchPassiv 4-69; 7-80
 S5GetConnectionCard 4-49
 S5GetConnectionParameter..... 4-48
 S5GetNetDateiname 7-85
 S5GetNetfileName 4-46
 S5GetRevision 4-45
 S5HoleRevision 7-59
 S5HoleVerbindungsparameters 7-87
 S5HoleVerbindungsparamsKarte 7-88
 S5LeseAusSPS 7-62
 S5LeseParameter 7-95
 S5ListConnections 4-52
 S5ListeNetVerbindungen..... 7-92
 S5ListeVerbindungen 7-91
 S5ListNetConnections 4-53
 S5PollWrite 4-67
 S5PutConnectionParameter 4-50
 S5ReadFromPLC..... 4-61
 S5ReadParameter..... 4-59
 S5SchreibeInSPS 7-71
 S5SchreibeParameter..... 7-96
 S5SchreibeVerbindungsparameter..... 7-89
 S5SchreibeVerbindungsparamsKarte 7-90
 S5SetzeNetDateiname 7-85; 7-86
 S5SetzeNetfileName 4-46
 S5SetzeStationsAdresse 7-61
 S5StartConnection 4-42
 S5StartConnectionCard 4-43
 S5StarteLesen 7-65
 S5StarteLesenOverlapped..... 7-66
 S5StarteVerbindung..... 7-56
 S5StartRead 4-62
 S5StartSchreiben 7-74
 S5StartWrite 4-66
 S5StopConnection 4-44
 S5StoppeVerbindung 7-57
 S5StoppeVerbindungen 7-58
 S5WriteConnectionCard..... 4-51
 S5WriteParameter..... 4-60
 S5WritePassiv..... 7-82
 S5WritePassive..... 4-70
 S5WriteToPLC..... 4-65
 Safety and handling precautions 1-1
 Send data 4-29
 Send expedited data..... 4-30
 Set MS-DOS entry Vector 4-28
 Set NET.NET filename..... 7-85
 Set NET.NET-file name 4-46
 Set station address..... 4-24; 4-55; 7-32; 7-61
 Set station address for multiple adapters..... 4-25; 4-56
 Source diagram Fetch Aktiv..... 6-7
 Source diagram Write Aktiv 6-6
 Source diagrams Fetch-Write..... 6-5
 Special features 1-5
 Specific layer 4 functions..... 4-29
 Standards and specifications 2-19
 Start a connection for multiple adapters..... 4-43
 Start connection 4-42; 7-56
 Start read..... 4-36; 4-62; 7-46
 Start read expedited data..... 4-37
 Start read operation..... 7-65
 Start reading expedited data..... 7-49
 Start Send..... 4-31; 7-38
 Start write operation..... 4-66; 7-74
 StartSchreibenOverlapped 7-75
T
 Technical data A-1
 Technical data thick-ethernet 2-13
 Technical data thin-ethernet..... 2-7
 Terminate a connection..... 4-18
 Terminate a connection..... 4-44; 7-57
 Terminate all connections 7-58
 Test connection 4-19
 Thick-ethernet-cable 2-8
 Thick-ethernet-cable networks..... 2-8
 Thin-ethernet-cable 2-3
 Thin-ethernet-cable network 2-5
 thin-ethernet-network 2-3
 Transceiver 2-8
 Transceiver-cable..... 2-8
 Transmit expedited data..... 7-37
 Transmitting data 7-36
 Transport error codes..... 4-71; 7-84
TSAP
 own..... 4-7
 remote 4-7
 Twisted Pair 2-16
V
 Vnr 7-18
W
 WMKTypes.h 5-12
 Write Active..... 4-65; 7-71
 Write connection parameter 7-89
 Write connection parameters 4-50
 for multiple adapters 4-51; 7-90
 Write H1 parameter record to file 4-58; 7-94
 Write H1 system values 4-27; 7-34
 Write Passive 4-70; 7-82
 Write PLC parameter record to file..... 4-60; 7-96
 Write to PLC..... 7-71
 Write to the PLC 4-65
Z
 Z'nux adapter..... 3-2
 Guide..... 7-1

hardware installation	3-4	Windows NT	3-25; 7-4
network cabling	3-5	Z'nyx-adapter	
software installation	3-7	guide	3-1
MS-DOS	3-8	Z'nyx-board	
OS/2	3-21	locator	1-3