



Driver Manual

Hilscher TCP/UDP IP Driver
Access to Hilscher Devices via TCP/IP and UDP/IP

Language: English

Index	Date	Version	Chapter	Revision
1	19.03.02	1.000	all	Drawn up
2	20.07.04	1.010	all 2.5.1 3.5.1, 3.6 3.8 4	Windows 9x, NT, 2000, XP Examples: Hints added Introduced Acknowledge configuration Added System Messages Error Numbers, chapter extended
3	20.06.08	1.010	3.8.8	Section 3.8.8 removed from document

Although this program has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this program for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this program or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifications at any time in as far as this contributes to technical progress. The version of the manual supplied with the program applies.

1	INTRODUCTION	5
1.1	Terms for this Manual	5
1.2	Overview	6
1.3	Message Structure	7
2	USING SYSTEM SPECIFIC TCP/UDP IP API	9
2.1	Parameters	9
2.2	Open Connection Endpoints	10
2.3	Establishing of a Connection (Device = Server)	10
2.4	Waiting for Incoming Connection (Device = Client)	10
2.5	Sending and Receiving Data	10
2.5.1	Examples	11
2.6	Avoiding TCP Send Delay	15
2.6.1	Acknowledge Message Format	15
3	USING TCP/UDP IP DRIVER	16
3.1	General	16
3.2	Operating systems	16
3.3	Function Overview	16
3.4	Contents for Windows 9x, Windows NT/2000/XP	17
3.5	Installation of the Hilscher IP Driver	18
3.5.1	Standard Registry Entries Windows 9x, Windows NT/2000/XP	19
3.5.2	Driver File Installation	21
3.5.3	Driver Utilities	21
3.6	Configure the Windows 9x/2000/NT/XP Driver	22
3.7	Programming Instructions	23
3.7.1	Include the Interface API in your Application	23
3.7.2	The Application Programming Interface	23
3.7.3	Important Hints	24
3.8	System Messages	25
3.8.1	Message DevReset	25
3.8.2	Message DevSetHostState	27
3.8.3	Message DevPutParameter	29
3.8.4	Message DevGetParameter	31
3.8.5	Message DevGetTaskState	33
3.8.6	Message DevTriggerWatchDog	35
3.8.7	Message DevGetInfo	37
4	ERROR NUMBERS	44

4.1	Error Numbers of Driver Functions (Return Values)	44
4.1.1	Error Numbers -1 .. -70.....	44
4.1.2	Error Numbers -1000 .. -1200.....	44
4.1.3	Error Numbers 10000 .. 10092	45
4.2	Error Numbers of System Messages	46
5	CONTACTS	48

1 Introduction

This manual describes the way of accessing Hilscher devices with IP interfaces via TCP/IP or UDP/IP and the application programming interface (API) to Hilscher devices. In the following DEVICE stands for communication interface, the communication module, NetNode, NetLink or any other device from Hilscher with IP interface.

The general mechanism of data transfer is protocol independent and for each hardware the same procedure and is described therefore in the Toolkit Manual 'General Definitions'

All parameter and data have basically the description LSB/MSB. This corresponds to the convention of the Microsoft C compiler. The storage format of word oriented send and receive process data of the handled I/O DEVICES is configurable.

Values with a following 'h' are in hexadecimal notation such as 1Eh = 30. Values without any following letter are in decimal notation.

Supplementary information is contained in the following Manuals:

- Toolkit Manual 'General Definitions' (Tk:TKIT),
- Device Driver Manual 'Device Driver' (Dd:DevDrv),
- Protocol Interface Manuals of used protocols.

1.1 Terms for this Manual

DPM	Dual-Port Memory this is the physical interface to all communication board (DPM is also used for PROFIBUS- DP Master).
CIF	Communication InterFace
COM	Communication Module
HOST	Application on the PC or a similar device
DEVICE	Synonym for communication interfaces or communication modules
RCS	Realtime Communicating System , this is the name of the operating system that runs on the communication boards
DLL	Dynamic Link Library

1.2 Overview

There are two ways of accessing Hilscher devices via IP protocol.

- Using the IP Driver
- Using the system specific IP stack Application Programming Interface (like Windows or Berkley sockets) on the system directly

In both cases communication is made by sending and receiving Hilscher messages over the TCP/IP or UDP/IP protocol. The format and the meaning of these messages are described in the Toolkit Manual 'General Definitions' (Tk:TKIT) and in Protocol Interface manuals of the given protocol on the device. For Example: Using a NetLink with MPI, the description is made in Protocol Interface manual of NetLink-MPI, MPI-Interface.

1.3 Message Structure

A message consists of an 8 byte message header, and optional 8 byte telegram header and up to 247 bytes of user data.

- **Message Header** Used by the DEVICE operating system for transporting and address the message. This structure is fixed and constant.
- **Telegram Header** Defines the action for the protocol task.
- **User data** Send/received data.

Parameter	Type	Meaning	
Msg.Rx	byte	Receiving Task	Message Header
Msg.Tx	byte	Sending Task	
Msg.Ln	byte	Message length	
Msg.Nr	byte	Identification Code	
Msg.A	byte	Response Code	
Msg.F	byte	Error Code	
Msg.B	byte	Command Code	
Msg.E	byte	Extension Code	
Msg.DeviceAdr	byte	Device Address	Telegram Header
Msg.DataArea	byte	Data Area	
Msg.DataAdr	word	Data Address	
Msg.DataIdx	byte	Data Index	
Msg.DataCnt	byte	Data Quantity	
Msg.DataType	byte	Data Type	
Msg.Fnc	byte	Function / Service	
Msg.D[0-246]	byte ... byte	User Data	Telegram User Data

General structure of a message

This is an example for a PROFIBUS-FMS command message. For other protocols the structure is the same, but the containing parameters must be changed when Modbus Plus is used for example, from communication reference to slave address, object index to register address, or service to function code.

Parameter	Type	Meaning	
Msg.Rx	byte	Receiving Task	Message Header
Msg.Tx	byte	Sending Task	
Msg.Ln	byte	Message length	
Msg.Nr	byte	Identification Code	
Msg.A	byte	Response Code	
Msg.F	byte	Error Code	
Msg.B	byte	Command Code	
Msg.E	byte	Extension Code	
Msg.DeviceAdr	byte	Communication Reference	Telegram Header
Msg.DataArea	byte	Data Block	
Msg.DataAdr	word	Object Index	
Msg.DataIdx	byte	Object Subindex	
Msg.DataCnt	byte	Data Quantity	
Msg.DataType	byte	Data Type	
Msg.Fnc	byte	Service	
Msg.D[0-246]	byte ... byte	User Data	Telegram User Data

General structure of a message

2 Using System specific TCP/UDP IP API

It is very simple to access a device from own applications via TCP/IP or UDP/IP. The application just has to establish an IP connection and to send specified Hilscher messages.

If the device works as TCP/IP or UDP/IP server, an application has to work as TCP/IP or UDP/IP client and has to open the connection. If the device works as TCP/IP or UDP/IP client, the device will establish the connection and the application has to wait for incoming connections.

The following section will describe the way of communication with the well known BSD socket or Winsock model. Most IP Application Programming Interfaces (API) are based on these model.

2.1 Parameters

Standard IP communication with Hilscher devices is handled over **TCP/IP Port 1099**, device is server. Other communication modes have to be configured on the devices, if possible.

Standard transmitter number (msg.tx) in messages is 255 (0xff).

2.2 Open Connection Endpoints

Open a socket with the function call:

```
SOCKET socket( int af, int type, int protocol);
```

On Windows systems, call WSStartup() before opening a socket.

Bind the successfully opened socket to local IP address with function:

```
int bind( SOCKET s,  
         const struct sockaddr FAR *name, int namelen);
```

2.3 Establishing of a Connection (Device = Server)

Connect the bound socket to the server with function call:

```
int connect( SOCKET s,  
            const struct sockaddr FAR *name, int namelen);
```

2.4 Waiting for Incoming Connection (Device = Client)

Set socket in listen state

```
int listen( SOCKET s, int backlog);
```

Wait for incoming connection on socket in listen state:

```
SOCKET accept( SOCKET s,  
              struct sockaddr FAR *addr, int FAR *addrlen);
```

2.5 Sending and Receiving Data

After a successful connect you can send and receive messages with the function calls:

```
int send( SOCKET s, const char FAR *buf, int len, int flags);
```

```
int recv( SOCKET s, char FAR *buf, int len, int flags);
```

Please see your system specific development documentation for further details.

2.5.1 Examples

2.5.1.1 Application in client mode

The following shows a simple example to establish a TCP/IP connection (device = server) and sending and receiving a message with Windows sockets.

```
.....
#include "winsock.h"
#define FIXED_PORT      1099
typedef struct MESSAGETELEGRAMtag {
unsigned char rx;          /* receiver          */
unsigned char tx;          /* transmitter      */
unsigned char ln;          /* length           */
unsigned char nr;          /* number           */
unsigned char a;          /* answer           */
unsigned char f;          /* fault            */
unsigned char b;          /* command          */
unsigned char e;          /* extension        */
unsigned char device_adr; /* device address   */
unsigned char data_area; /* data area        */
unsigned short data_adr; /* data address     */
unsigned char data_idx;  /* data index       */
unsigned char data_cnt;  /* data count       */
unsigned char data_type; /* data type        */
unsigned char function;  /* function         */
unsigned char d[ 247 ];
} MESSAGETELEGRAM;

int      err;
WORD     wVersionRequired;
WSADATA  wsaData;
SOCKET   soc;
wVersionRequired = MAKEWORD(1,1);
// initialize WinSock library
err = WSASStartup(wVersionRequired, &wsaData);
if (err != 0)
    exit(1);
```

```

// create a TCP/IP socket
soc = socket ( AF_INET, SOCK_STREAM, 0);
if (soc != INVALID_SOCKET)
{
    struct sockaddr_in LocalAddr;
    LocalAddr.sin_family = AF_INET;
    LocalAddr.sin_addr.s_addr = htonl( INADDR_ANY );
    LocalAddr.sin_port = 0;
    // bind the socket to local address
    if( bind( soc, (struct sockaddr *)&LocalAddr, sizeof(LocalAddr)) != SOCKET_ERROR )
    {
        struct sockaddr_in RemoteAddr;
        char szAddress[] = {"192.168.10.161"};
        unsigned long IpAddress = inet_addr( szAddress );
        RemoteAddr.sin_family = AF_INET;
        RemoteAddr.sin_addr.s_addr = IpAddress;
        RemoteAddr.sin_port = htons(FIXED_PORT);

        //connect to server with IP address 192.168.10.161 on port 1099
        if( connect( soc, (struct sockaddr *)&RemoteAddr, sizeof(RemoteAddr)) == 0 )
        {
            MESSAGETELEGRAM SendMessage;
            MESSAGETELEGRAM ReceiveMessage;
            int SendLen, ReceiveLen, MsgLenTotal;
            // build message to read data block with MPI protocol, see Protocol Interface
            // manual of NetLink-MPI, MPI-Interface
            SendMessage.rx = 3;
            SendMessage.tx = 255;
            SendMessage.ln = 8;
            SendMessage.nr = 0;
            SendMessage.a = 0;
            SendMessage.f = 0;
            SendMessage.b = 0x31;
            SendMessage.e = 0;
            SendMessage.device_adr = 2;
            SendMessage.data_area = 0;
            SendMessage.data_adr = 10;           // DB = e.g. 10
            SendMessage.data_idx = 0;
            SendMessage.data_cnt = 1;
            SendMessage.data_type = 5;
            SendMessage.function = 1;           // Read = 1
            MsgLenTotal = SendMessage.ln + 8;
            // Send data over TCP/IP connection to device
            SendLen = send(soc, (char*)&SendMessage, MsgLenTotal, 0);
            if( SendLen == MsgLenTotal)
            {
                // receive answer message
                ReceiveLen = recv( soc, (char*)&ReceiveMessage, sizeof(ReceiveMessage), 0);
                // .. do something with answer
            }
        }
    }
}
}

```

```
// close socket
closesocket ( soc );
// Cleanup and return
WSACleanup();
.....
```

Note:

The typical communication for read respectively write messages is this:

- To read data (SendMessage.function = 1) the send message has a total length of 16 = 8 + SendMessage.In. Then in ReceiveMessage the ReceiveMessage.In has
 - in case of no error (ReceiveMessage.f == 0) a length of greater than 8 (where $n = \text{ReceiveMessage.In} - 8$ is the number of user data in bytes)
 - in case of error (ReceiveMessage.f != 0) a length of 8.
- To write data (SendMessage.function = 2) the send message has a total length of 8 + SendMessage.In + N, where N is the number of user data in bytes that should be written. Then in ReceiveMessage the ReceiveMessage.In has
 - in case of no error (ReceiveMessage.f == 0) a length of 8
 - in case of error (ReceiveMessage.f != 0) a length of 8.

2.5.1.2 Application in server mode

```

.....
#include "winsock.h"
.....
int      err;
WORD     wVersionRequired;
WSADATA  wsaData;
SOCKET   soc, AcceptedSocket;
wVersionRequired = MAKEWORD(1,1);
// initialize WinSock library
err = WSASStartup(wVersionRequired, &wsaData);
if (err != 0)
    exit(1);

// create a TCP/IP socket
soc = socket ( AF_INET, SOCK_STREAM, 0);
if (soc != INVALID_SOCKET)
{
    struct sockaddr_in LocalAddr;
    LocalAddr.sin_family = AF_INET;
    LocalAddr.sin_addr.s_addr = htonl( INADDR_ANY );
    LocalAddr.sin_port = htons(FIXED_PORT);
    // bind the socket to local address
    if( bind( soc, (struct sockaddr *)&LocalAddr, sizeof(LocalAddr)) != SOCKET_ERROR )
    {
        char szAddress[MAX_PATH];
        int AddressLen = sizeof( szAddress );
        // set socket in listen state
        if( listen( soc, SOMAXCONN ) == NO_ERROR )
        {
            //accept incoming connection
            if( (AcceptedSocket = accept( soc, (struct sockaddr FAR*)&szAddress,
                                         &AddressLen)) != INVALID_SOCKET )
            {
                MESSAGETELEGRAM SendMessage;
                MESSAGETELEGRAM ReceiveMessage;
                int SendLen, ReceiveLen;

                ReceiveLen = recv(AcceptedSocket, (char*)&ReceiveMessage,
                                 sizeof(ReceiveMessage), 0);
                .....
                // see example below
            }
        }
    }
}

```

Note:

The example shows the use of sockets in blocking mode. Every function call on a socket is blocking until the command was successfully done or if any error occurred. You can use the `ioctlsocket()` or the `select()` function to handle sockets in non-blocking mode. Another way to handle blocking sockets in a Windows environment is to use threads for sending and receiving data.

2.6 Avoiding TCP Send Delay

On some circumstances it is possible that TCP/IP messages send from Hilscher devices will block on the device until an answer from the TCP/IP stack of the host application is available, also if no answer from application is required. This is, because by default Windows TCP/IP stack is waiting about 200ms until a TCP/IP acknowledge is sent to the communication partner. The device is not able to send a new TCP/IP message until this acknowledge, so new messages will block on the device. To avoid this, the application can send an own acknowledge. This is done by sending an defined message (see below) over TCP/IP to the device. This message causes automatically an TCP/IP acknowledge and the device is able to send new messages to the application.

Example: A Hilscher device, like a NetNode or NetLink, has to send TCP/IP messages with user data to the remote application in high speed. If no answer from the application is required, the device can send max. 5 messages per second, because Windows TCP/IP stack is waiting about 200ms until an acknowledge is sent to the device. The device is not able to send new data until this acknowledge is received. If the application sends the defined acknowledge message right after receiving the user data, the device is able to send new user data right after this.

The defined acknowledge message has no further effects on the device.

2.6.1 Acknowledge Message Format

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> Operating System RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	0	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	255	<u>Reply Identification</u> Acknowledge Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard

Message format of acknowledge message

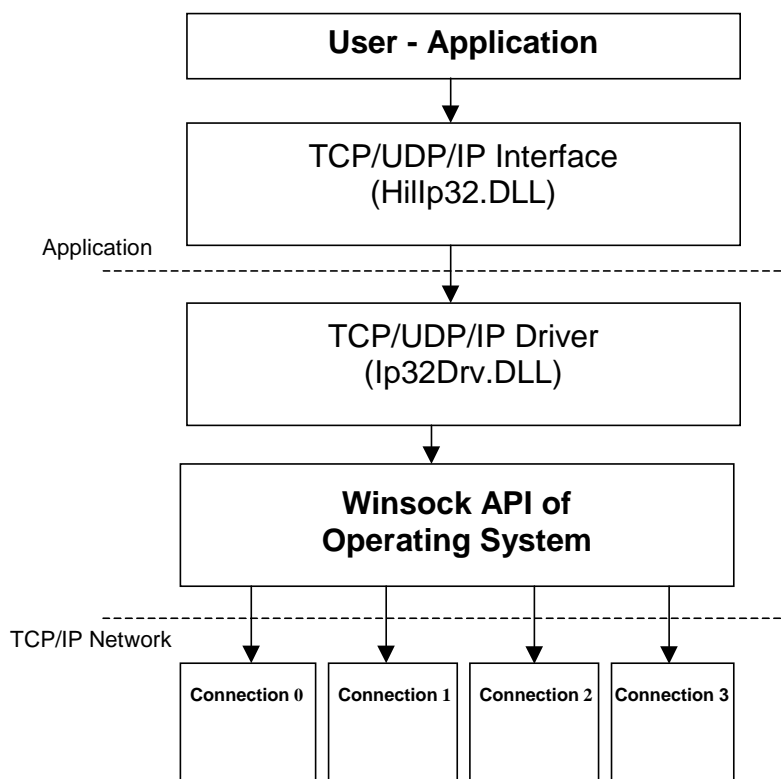
3 Using TCP/UDP IP Driver

3.1 General

The API of the TCP/UDP IP Driver is the same like the API of CIF Device Driver for our CIF cards and COM modules. The driver provides the same functionality like the CIF Device Driver, please see Device Driver manual (Dd:DevDrv) for further information. If someone has already an application running with the CIF Device Driver it is very easy to use TCP/UDP/IP Driver instead.

3.2 Operating systems

For **Windows 9x**, **Windows NT**, **Windows 2000** and **Windows XP** Hilscher offers the TCP/UDP IP driver. The communication between the application and the driver is done by a DLL. This DLL can be statically or dynamically linked to the application.



TCP/UDP IP Driver components

3.3 Function Overview

The IP drivers for Windows 9x, Windows NT, Windows 2000 and Windows XP can handle up to four connections.

On each connection only one command can be active at the same time, because there is no command queuing in the driver implemented.

3.4 Contents for Windows 9x, Windows NT/2000/XP

Directory	Subdirectory	Description
<INSTALL>	API	Application Programming Interface, libraries and header files to access the 32 Bit driver DLL (the DLL is installed by the driver installation)
	Demo	Simple Message and IO data transfer source code example (IODemo.cpp)
		IpDrvTest: Complete TCP/UDP/IP driver test program written in C++, created with Microsoft Visual C/C++ 6.0
	MANUALS	TCP/UDP/IP driver manual

CD content

Windows 9x, Windows NT and Windows 2000 driver files:

HILIP32.DLL	Dynamic link library of the driver interface, created for use with Windows 9x, Windows NT and Windows 2000
HILIP32.LIB	Definition file with the exported function of the HILIP32.DLL.
IPDRVUSR.H	Definition header file for the user interface.
IP32DRV.DLL	TCP/UDP IP driver DLL

Applications:

IpDrvSetup.EXE	Driver Setup program for registry entries
IpDrvTest.EXE	Driver Test program to run the various device driver functions

Development platform:

Windows 9x	Microsoft Visual C++, V 6.x
Windows NT 4.0	Microsoft Visual C++, V 6.x
Windows 2000	Microsoft Visual C++, V 6.x
Windows XP	Microsoft Visual C++, V 6.x

ATTENTION:

The TCP/UDP IP Interface DLL and the driver files are installed during the Hilscher IP driver installation and not included in the development directories.

3.5 Installation of the Hilscher IP Driver

The driver will be installed by an installation program. This will guide you to the installation process. The installation program will run the following steps:

- Creating the standard registry entries for the TCP/UDP IP Driver
- Copying the device driver / interface DLL files
- Copying the device driver setup and test program

3.5.1 Standard Registry Entries Windows 9x, Windows NT/2000/XP

Registry Path:

\HKEY_LOCAL_MACHINE\Software\Hilscher GmbH\

TCP/UDP IP Driver Entry:

IP Driver	- Company	// Hilscher GmbH
	- CurrentFolder	// Installation folder of driver
	- CurrentVersion	// Current version of driver
	- Directory	// Installation directory of driver
	- Name	// Name of driver
	\Connection0	
	\Connection1	
	\Connection2	
	\Connection3	

The default entries are

```
Connection0 - IpAddress      0x00000000 // TCP/IP Address of connection 0
              - PortNumber   0x0000044B // Port Number of TC/IP connection
              - Mode         Client // Client Mode
              - Protocol     'TCP' // TCP protocol is used
              - ConnectTimeout 0x00002710 // Timeout for connection in msec.
              - Ack          0x00000001 // Acknowledge message enabled
Connection1 - IpAddress      0x00000000 // TCP/IP Address of connection 1
              - PortNumber   0x0000044B // Port Number of TC/IP connection
              - ClientMode   TRUE // Client Mode = TRUE
              - Protocol     'TCP' // TCP protocol is used
              - ConnectTimeout 0x00002710 // Timeout for connection in msec.
              - Ack          0x00000001 // Acknowledge message enabled
Connection2 - IpAddress      0x00000000 // TCP/IP Address of connection 2
              - PortNumber   0x0000044B // Port Number of TC/IP connection
              - ClientMode   TRUE // Client Mode = TRUE
              - Protocol     'TCP' // TCP protocol is used
              - ConnectTimeout 0x00002710 // Timeout for connection in msec.
              - Ack          0x00000001 // Acknowledge message enabled
Connection3 - IpAddress      0x00000000 // TCP/IP Address of connection 3
              - PortNumber   0x0000044B // Port Number of TC/IP connection
              - ClientMode   TRUE // Client Mode = TRUE
              - Protocol     'TCP' // TCP protocol is used
              - ConnectTimeout 0x00002710 // Timeout for connection in msec.
              - Ack          0x00000001 // Acknowledge message enabled
```

3.5.2 Driver File Installation

TCP/UDP IP Interface DLLs:

Windows 9x	The interface DLL HILIP32.DLL is copied to the %System Root%\System directory.
Windows NT	The interface DLL HILIP32.DLL is copied to the %System Root%\System32 directory.
Windows 2000	The interface DLL HILIP32.DLL is copied to the %System Root%\System32 directory.
Windows XP	The interface DLL HILIP32.DLL is copied to the %System Root%\System32 directory.

TCP/UDP IP Driver DLLs:

Windows 9x	The driver DLL IP32DRV.DLL is copied to the %System Root%\System directory.
Windows NT	The driver DLL IP32DRV.DLL is copied to the %System Root%\System32 directory.
Windows 2000	The driver DLL IP32DRV.DLL is copied to the %System Root%\System32 directory.
Windows XP	The driver DLL IP32DRV.DLL is copied to the %System Root%\System32 directory.

Device Driver Utilities:

Installation path	<System>\Program Files\Hilscher GmbH\IP Driver
IpDrvSetup	Driver setup programm
IpDrvTest	Driver test programm

3.5.3 Driver Utilities

The driver includes a driver setup (IPDRVSETUP.EXE) and a driver test (IPDRVTEST.EXE) program. These files are also installed during the installation procedure. Therefore, the installation program creates a Hilscher GmbH\IP Driver directory below the standard PROGRAM directory where the files are copied.

3.6 Configure the Windows 9x/2000/NT/XP Driver

The user must configure the IP address, the port number, the kind of connection and the used protocol of each connection. All these informations are written to the registry data base of the operating system.

To get an easy access to this data the device driver gets its own setup program IPDRVSETUP.EXE. This program will help you to change the registry entries without using REGEDIT.EXE.

Parameter	Description
IP address	IP address of device to which the connection should be established, maybe 0 if application works in server mode
Port Number	Port Number on which the IP connection should be established
Client / Server Mode	Determines if TCP connection should work as client or server
Protocol	Determines which protocol (TCP or UDP) is used
Connect Timeout	Determines how long the driver should try to establish a connection on DevInitBoard() function call
Ack	Determines if the acknowledge message is sent or not, refer to section <i>Avoiding TCP Send Delay</i> on page 15

Driver parameters

3.7 Programming Instructions

3.7.1 Include the Interface API in your Application

For the user API there is only one include file IPDRVUSR.H which contains all the necessary information like structure, constant and prototype definitions. A complete function description is given in the Device Driver manual (Dd:DevDrv). Link the device API-DLL (HILIP32.LIB) to your program. Make sure you have installed the driver if this one is used.

Further programming instructions can be found in the Device Driver Manual (Dd:DevDrv), since the API is same.

3.7.2 The Application Programming Interface

Since the TCP/UDP/IP driver has the same API as the CIF Device Driver, please refer to Device Driver manual for further information.

3.7.2.1 Differences

- The meaning of the parameter usDevNumber is changed from board number to connection number
- DevGetBoardInfo delivers IRQ number always as 0, Physical Address is changed to IP address

```
typedef struct tagBOARD_INFO{
    unsigned char abDriverVersion[16]; // DRV version information
    struct {
        unsigned short usBoardNumber; // DRV connection number
        unsigned short usAvailable; // DRV board is available
        unsigned long ulIpAddress; // DRV IP address
        unsigned short usIrqNumber; // DRV irq number, always 0
    } tBoard [MAX_DEV_BOARDS];
} BOARD_INFO;
```

- DevGetBoardInfo may need a long time to return, if an configured device is not available (TCP/IP Timeout)
- DevGetBoardInfo returns always usBoardAvailable = TRUE on UDP connections
- Function DevInitBoard tries to establish an IP connection to the configured device, no further testing actions are done
- DevGetMBXState delivers the state, if sending or receiving data over the configured IP connection is possible.
- DevGetInfo(), Info Area GET_DRIVER_INFO: Parameters IRQCnt, bHostFlags, bMyDevFlags, bExIOFlag are not supported, set to zero
- DevGetInfo(); Info Area GET_RCS_INFO: Parameters: bRcsError, bHostWatchdogState, bDevWatchdogState, bSegmentCount, bDeviceAddress, bDriverType are not supported, set to zero
- DevGetInfo(); Info Area GET_DEV_INFO: not supported, set to zero
- DevGetInfo(); Info Area GET_IO_INFO: not supported, set to zero

- DevExchangeIOEx(); not supported
- DevReadSendData(); not supported
- DevReadWriteDPMRaw(); not supported
- DevExtendedData(); not supported
- DevGetMbxData(); not supported
- DevSpecialControl(); not supported
- DevDownload(); not supported
- DevReadWriteDPMDData(); not supported

3.7.3 Important Hints

- Use DevGetMBXState() function to get information about a receive message is available. Do not use DevGetMessage() without timeout to poll the receive mailbox. This will cause a lot of Network traffic!
- Timeouts with TCP/UDP IP driver may be much longer than using the CIF Device Driver.

3.8 System Messages

In general these system messages are diagnostic functions or global commands which are sent not to a specific task but to the Operational System called Realtime Communication System (short RCS). The following messages are directed to the RCS directly which has the message receiver always set to 0 (msg.rx = 0). The answer message will be returned to the sender given in msg.tx.

3.8.1 Message DevReset

This message is used by the user application in order to perform a Reset of the DEVICE. All DEVICES distinguish two types of Resets. While the so called “Coldstart” instructs the DEVICE to start right from the scratch, to renew the RAM-Test and take the Static Parameter of the FLASH, the “Warmstart” command will be more soft and instructs to take next to the FLASH parameter also those which were online configured by the DevPutTaskParameter message.

Note If the function (no matter of Coldstart or Warmstart) is performed during operation, the fieldbus communication will be stopped.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msd.d[0]	UINT8	1 2	<u>Type of Reset</u> Coldstart Warmstart

The DEVICE replies with the following message:

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	0	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	J	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard

Note 1 Under some circumstances it becomes necessary to establish the TCP/IP connection again because this message may affect the DEVICES TCP/IP software stack as well.

Note 2 The reset can take a long time.

3.8.2 Message DevSetHostState

This message has to be used by the HOST application in order to signal the DEVICE if the application is running or not. This is normally used in cases the DEVICE is configured not to run any fieldbus activity until an application has given its permission to. A DEVICE can consist of 3 different communication links which can be enabled and disabled. All 3 links must be instructed with the same command in msg.d[1-3]. See the following Command Message:

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	2	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	17	<u>Sub Command</u> Set Host State
	msg.d[1]	UINT8	1 0	<u>Mode Communication Link 1</u> Host Not Ready, disables link Host Ready, enables link

The DEVICE replies with the following message:

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	0	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard

3.8.3 Message DevPutParameter

This message has to be used by the HOST application in order to write parameter into the specified TaskParameter (1 or 2) within the DEVICE. Giving task parameter online during startup is necessary for some DEVICES to overwrite static configured parameter. The lifetime of this parameter will be as long as the DEVICE is not resetted.

Note After the DevPutParameter Message is processed the application has to perform the Warmstart command (DevReset-Message) afterwards, so that the new parameter becomes valid.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	4 + m	<u>Message Length</u> Length in Octets (m = 1 .. 64)
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	20	<u>Sub Command</u> Get/Put Parameter
	msg.d[1]	UINT8	1	<u>Function</u> Put Parameter
	msg.d[2]	UINT8	1 .. 64	<u>Size</u> Size of TaskParameter Area
	msg.d[3]	UINT8	1, 2	<u>Area</u> TaskParameter Area
	msg.d[4 .. 67]	UINT8	0 .. 255	<u>Data</u> TaskParameter Data

The content of the TaskParameter Data depends on the type of the DEVICE. For detailed structure information refer to its corresponding Protocol Interface manual.

The DEVICE replies with the following message:

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	0	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard

3.8.4 Message DevGetParameter

During power up the DEVICE writes into the parameter areas some elected and changeable parameter of the DEVICES internal FLASH configuration. Values could be for example an IP-Address or Bus Address. The following message has to be used by the HOST application in order to read parameter data from the specified TaskParameter (1 or 2) area. Each area has in total 64 bytes.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	4	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	20	<u>Sub Command</u> Get/Put Parameter
	msg.d[1]	UINT8	2	<u>Function</u> Get Parameter
	msg.d[2]	UINT8	1 .. 64	<u>Size</u> Size of TaskParameter Area
	msg.d[3]	UINT8	1, 2	<u>Area</u> TaskParameter Area

The content of the returned TaskParameter Data depends on the type of DEVICE. For detailed structure information refer to its corresponding Protocol Interface manual.

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	0 .. n+1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> 0 No Error else see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0 .. n]	Array UINT8	0 .. 255	<u>Data</u> Content of the Requested Task Parameter Area

3.8.5 Message DevGetTaskState

Each DEVICE holds some compact information of its internal state in 2 separated structures which have in maximum 64 bytes each. The following message has to be used by the HOST application in order to read the specified TaskState area.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	4	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	20	<u>Sub Command</u> Get/Put Parameter
	msg.d[1]	UINT8	3	<u>Function</u> Get State
	msg.d[2]	UINT8	1 .. 64	<u>Size</u> Size of TaskState Area
	msg.d[3]	UINT8	1, 2	<u>Area</u> TaskState Area

The content of the returned TaskState Data depends on the type of specific fieldbus system. For detailed structure information refer to the corresponding Protocol Interface manual.

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	0 .. n+1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0 .. n]	UINT8	0 .. 255	<u>Data</u> Content of the Requested TaskState Area

3.8.6 Message DevTriggerWatchDog

This message is used by the HOST application in order to control the watchdog function of the DEVICE. By default the HOST watchdog function is disabled in the DEVICE until the first trigger is received through the DevTriggerWatchdog Message. The time that is supervised by the DEVICE between each life signaling Watchdog-Message is configured normally through the FLASH configuration or can be changed by the DevPutTaskParameter-Message. Once the Watchdog is started the application must retriggering it within the lifetime, else the DEVICE will stop the fieldbus activity.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	2	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	23	<u>Sub Command</u> Trigger Watchdog
	msg.d[1]	UINT8	0 1	<u>Function</u> 0 Stop Watchdog 1 Retrigger Watchdog

The DEVICE replies with the following message:

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	2	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0, 1]	UINT16	0 .. 65535	<u>Data</u> Value internal Watchdog Counter before the trigger message was received

3.8.7 Message DevGetInfo

This chapter covers several helpful functions that can be used to obtain internal diagnosis and information structures.

3.8.7.1 Function GetVersionInfo

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	15	<u>Sub Command</u> Get Version Info

The DEVICE replies with the following message:

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	32	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	1	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0 .. 3]	UINT32		<u>Manufacturing Date</u> BCD coded Example: 30012003 as value will mean produced 30. January 2003
	msg.d[4 .. 7]	UINT32		<u>Hilscher Device Number</u> BCD code
	msg.d[8 .. 11]	UINT32		<u>Serial Number</u> BCD coded
	msg.d[12 .. 15]	UINT32		Reserved
	msg.d[16 .. 19]	UINT32		<u>License Code1</u> ASCII coded
	msg.d[20 .. 23]	UINT32		<u>License Code2</u> ASCII coded
	msg.d[24 .. 27]	UINT32		<u>License Code3</u> ASCII coded
	msg.d[28 .. 31]	UINT32		Reserved

3.8.7.2 Function GetFirmwareInfo

This function is used by the HOST application in order to obtain the DEVICES firmware name and its version.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	4	<u>Sub Command</u> Get Firmware

The function returns firmware name its version and the checksum of the firmware file.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	34	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	1	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0 .. 15]	INT8	0 .. 127	Name of Firmware (ASCII)
	msg.d[16 .. 31]	INT8	0 .. 127	Version of Firmware (ASCII)
	msg.d[32, 33]	UINT16	0 .. 65535	Checksum of Firmware

Example

The DeviceNet Master firmware on the PCI board returns the following:

```
DNM      CIF50DNM
V01.087 13.12.02
```

3.8.7.3 Function GetTaskInfo

This function is used by the HOST application in order to obtain names of the tasks and their versions running on the DEVICE.

Command Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	0	<u>Identification of Receiver</u> RCS
	msg.tx	UINT8	255	<u>Identification of Transmitter</u> Driver Tx
	msg.ln	UINT8	1	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	0	<u>Reply Identification</u> No Reply
	msg.f	UINT8	0	<u>Error Code</u> No Error
	msg.b	UINT8	2	<u>Command Identification</u> RCS Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0]	UINT8	2	<u>Sub Command</u> Get Task Information

The function returns various information of the running task.

Answer Message				
	Parameter	Type	Value	Description
Message Header	msg.rx	UINT8	255	<u>Identification of Receiver</u> Driver Tx
	msg.tx	UINT8	0	<u>Identification of Transmitter</u> RCS
	msg.ln	UINT8	104	<u>Message Length</u> Length in Octets
	msg.nr	UINT8	j	<u>Message Identification</u> Unique Number
	msg.a	UINT8	2	<u>Reply Identification</u> RCS Reply
	msg.f	UINT8	0 else	<u>Error Code</u> No Error see chapter <i>Error Numbers of System Messages</i> on page 46
	msg.b	UINT8	0	<u>Command Identification</u> No Command
	msg.e	UINT8	0	<u>Extension</u> Standard
Message Data	msg.d[0 .. 7]	INT8	0 .. 127	Name of Operation System
	msg.d[8 .. 9]	INT16	-32768 ... 32767	Version of Operation System Version = Value • 0.01
	msg.d[10]	UINT8	0	Reserved
	msg.d[11]	UINT8	0	Reserved
	msg.d[12]	UINT8	0	Reserved
	msg.d[13 .. 20]	INT8	0 .. 127	Name of Task 1
	msg.d[21]	INT16	-32768 ... 32767	Version of Task 1 Version = Value • 0.01
	msg.d[23]	UINT8	0 .. 7	Priority of Task 1
	msg.d[24]	UINT8	0 .. 7	Start Index of Task 1
	msg.d[25]	UINT8	0 .. 255	State of Task 1 see table below
	msg.d[26 .. 33]	INT8	0 .. 127	Name of Task 2
	etc.		...	etc.
	msg.d[103]	UINT8	0 .. 255	State of Task 7 see table below

State of Task	
Value	Description
0	Okay/Success
1	Locked
2	Waiting
3 .. 249	Init Faults (see: fieldbus specific Protocol Interface manuals)
250	Initialized
251	Ready
252	Called
253	Available
254	Configured
255	Empty

4 Error Numbers

4.1 Error Numbers of Driver Functions (Return Values)

4.1.1 Error Numbers -1 .. -70

Error numbers are compatible with error numbers from CIF device driver. Please see device driver manual for further details.

New error numbers are defined for TCP/UDP IP as documented in the following section

4.1.2 Error Numbers -1000 .. -1200

Error Number	Description
-1000	Function not implemented
-1001	Error by getting resources from Host PC
-1100	Internal NULL pointer exception
-1101	Error by accessing registry
-1102	Error reading key in registry
-1103	Unknown mode (not Client or Server)
-1104	Unknown protocol (not TCP or UDP)
-1105	Connection number is invalid
-1200	Wrong answer for sent command received

Error Number -1000 .. -1200

4.1.3 Error Numbers 10000 .. 10092

Error numbers of 10000 and higher are Windows socket system specific error numbers. You will find detailed information in the documentation of your operating system.

Important Windows socket error numbers:

Error Number	Description
10036	Operation now in progress. A blocking operation is currently executing. Windows Sockets only allows a single blocking operation—per- task or thread—to be outstanding
10051	Network is unreachable. A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host
10054	Connection reset by peer. An existing connection was forcibly closed by the remote host.
10057	Socket is not connected. A request to send or receive data was disallowed because the socket is not connected
10058	Cannot send after socket shutdown. A request to send or receive data was disallowed because the socket had already been shut down
10060	Connection timed out. A connection attempt failed because the connected party did not properly respond after a period of time, or the established connection failed because the connected host has failed to respond
10061	Connection refused. No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host—that is, one with no server application running.
10091	Network subsystem is unavailable. This error is returned by WSASStartup if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable
10092	Winsock.dll version out of range. The current Windows Sockets implementation does not support the Windows Sockets specification version requested by the application.

Error Numbers 10000 .. 10092 (WinSock error numbers)

4.2 Error Numbers of System Messages

Error Number	Description
4	Task does not exist
5	Task is not initialized
6	The MCL is locked
7	The MCL rejects a send command because of an error
20	The user will download a database into the device that is not valid for this device type.
21	Data base segment not configured or not existed
22	Number for message wrong during download
23	Received number of data during download does not match to that in the command message
24	Sequence identifier wrong during download
25	Checksum after download and checksum in command message do not match
26	Write/Read access of data base segment
27	Download/Upload or erase of configured data base type is not allowed
28	The state of the data base segment indicated an error. Upload not possible
29	The access to the data base segment needs the bootstraploader. The bootstraploader is not present
30	Trace buffer overflow
31	Entry into trace buffer too long
37	No or wrong license. The OEM license of the System Configurator allows only communication to devices that have the same license inside
38	The data base created by the System Configurator and the data base expected by the firmware is not compatible
39	DBM module missing
40	No command free
41	Command unknown
42	Command mode unknown
43	Wrong parameter in the command
44	Message length does not match to the parameters of the command
45	Only a MCL does use this command to the RCS
50	FLASH occupied at the moment
51	Error deleting the FLASH
52	Error writing the FLASH
53	FLASH not configured
54	FLASH timeout error
55	Access protection error while deleting the FLASH
56	FLASH size does not match or not enough FLASH memory

Error Numbers 4 .. 56 (System messages)

Error Number	Description
60	Wrong structure type
61	Wrong length of structure
62	Structure does not exist
70	No clock on the device
80	Wrong handle for the table (table does not exist)
81	Data length does not match the structure of this table
82	The data set of this number does not exist
83	This table name does not exist
84	Table full. No more entries allowed
85	Other error from DBM
90	The device info (serial number, device number and date) does already exist
91	License code invalid
92	License code does already exist
93	All memory locations for license codes already in use

Error Numbers 60 .. 93 (System messages)

5 Contacts

Headquarter

Germany

Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support

Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China

Hilscher Ges.f.Systemaut. mbH
Shanghai Representative Office
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support

Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France

Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support

Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

Italy

Hilscher Italia srl
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support

Phone: +39 / 02 25007068
E-Mail: it.support@hilscher.com

Japan

Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support

Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Switzerland

Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support

Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA

Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support

Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com