SYS TEC ELECTRONIC

CAN-Ethernet-Gateway

System Manual

Edition July 2010

system house for distributed automation

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

The information in this document has been carefully checked and is believed to be entirely reliable. However, SYS TEC electronic GmbH assumes no responsibility for any inaccuracies. SYS TEC electronic GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. SYS TEC electronic GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.

Additionally, SYS TEC electronic GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. SYS TEC electronic GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

	EUROPE	NORTH AMERICA
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (3661) 6279-0 info@systec-electronic.com	1 (800) 278-9913 info@phytec.com
Technical Support:	+49 (3661) 6279-0 support@systec-electronic.com	1 (800) 278-9913 support@phytec.com
Fax:	+49 (3661) 62 79 99	1 (206) 780-9135
Web Site:	http://www.systec-electronic.com	http://www.phytec.com

1st Edition July 2010

1 Introduction

1.1 Fundamentals

Internet communication via TCP/IP is finding ever-increasing implementation in industrial applications. The CAN-Ethernet Gateway from SYS TEC electronic GmbH is a solution that enables CAN networks to be coupled together over the Internet/Ethernet, whereby remote monitoring and control is possible. The CAN-Ethernet Gateway controls networked communication and makes a transparent CAN-based application interface available to the user.

The device supports a transparent, protocol-independent transfer of the CAN messages, thus allowing its implementation into a wide range of possible applications. Furthermore, the CAN-Ethernet Gateway can be used with various higher layer CAN protocols (e.g. CANopen, SDS, J1939, DeviceNet or other proprietary protocols).

The CAN-Ethernet Gateway can be used in CAN networks with a transfer rate of up to 1 MBit/s corresponding to CAN Specification 2.0A (11-bit CAN identifier) and 2.0B (29-bit CAN identifier). For each CAN message a time stamp can be created by the CAN-Ethernet Gateway that is transferred along with the data.

The CAN-Ethernet Gateway can be configured via an asynchronous serial interface (UART with RS-232 using hardware flow control) or via a Telnet connection. The user can therefore adapt the functions of the CAN-Ethernet Gateway to the specific application environment.

For communication between the CAN-Ethernet Gateways a BTP/IP-based network protocol (BTP = Block Transfer Protocol) is used. This enables the CAN messages to be routed over the Ethernet with minimal delay time. The TCP/IP protocol time for establishing and ending a network connection is thereby eliminated.

As an option, the CAN messages can also be transferred using a TCP/IP network protocol.

The Gateway firmware is designed for high data throughput. The optimized buffer management requires minimal resources for copying and temporarily storing data. Transmission rate spikes that may occur in the CAN network will be easily handled. If a large amount of data is transmitted, multiple CAN messages are combined in one UDP or TCP package and transmitted as a single block.

The CAN-Ethernet Gateway identifies errors and sends CAN messages (error messages) that contain the reason for the error. The error message's CAN identifier is configurable (*refer to section 6.3*).

1.2 Application Fields

1.2.1 Connecting Two CAN Networks over Ethernet

A typical application is the connection of two CAN networks via Ethernet over a great distance. A CAN-Ethernet Gateway is present in each CAN network. CAN messages are transferred transparently between the CAN-Ethernet Gateways.

The firmware of the CAN-Ethernet Gateway allows for filtration of the CAN messages to be sent on, thus only the relevant data gets transmitted over the Ethernet.

The principle possibilities for building a network structure using CAN-Ethernet Gateways are represented in *Figure 1*.

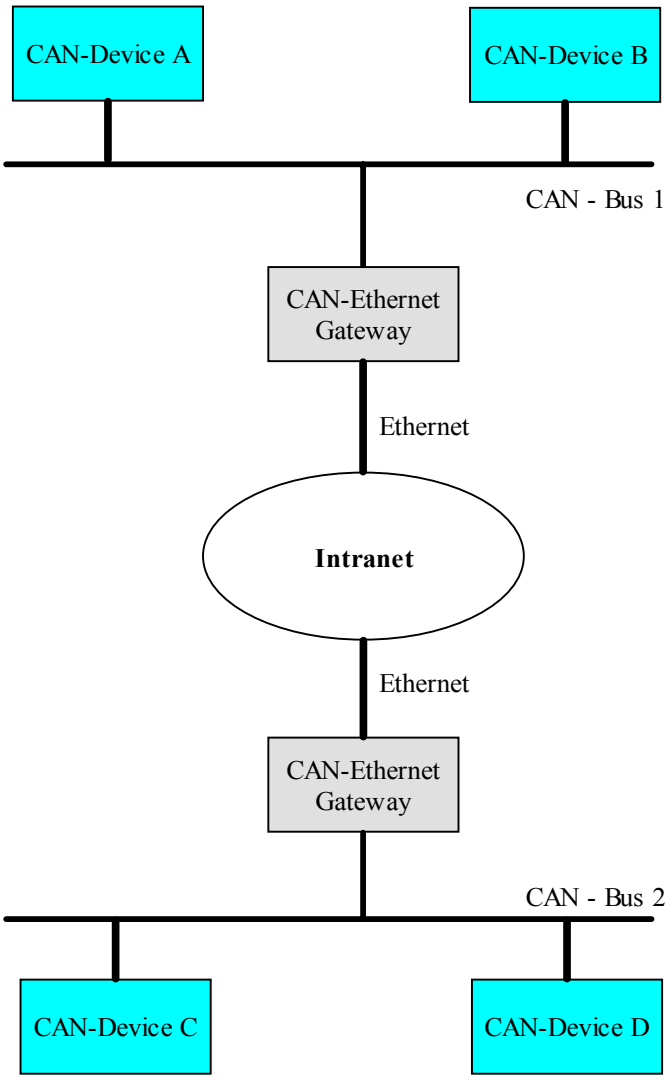

Figure 1: Application Example #1: Transparent Connection of two CAN Networks using Intranet/Ethernet

CAN-Ethernet Gateways are represented in *Figure 2*.

1.2.2 Remote Diagnostics and Configuration of CAN Networks

Another possible application is the connection of a CAN network with a computer. The user only requires a network connection over the Ethernet in order to establish a connection with the remote CAN network. No CAN hardware is required on the host computer.

A virtual CAN-Ethernet Gateway in the form of PC software (DLL) is available for MS Windows systems. The interface of the virtual CAN-Ethernet Gateway corresponds to a CAN driver.

Thus it is possible to use standard CAN programs, which use a CAN driver (e.g. CANopen configuration tools like CANsetter™ and ProCANopen™ or CAN analyzer tools like PCAN-Explorer™ or PCANview™).

The virtual CAN-Ethernet Gateway for PC extends the CAN network over the Ethernet/Intranet/Internet into the office and offers new possibilities for configuration and diagnosis of CAN networks in the field level. The computer in the communication management level requires an Ethernet connection to the field level, however it offers the convenience of familiar CAN and CANopen tools.

The functions and parameters of the Gateway itself can be accessed and modified remotely using the Telnet protocol.

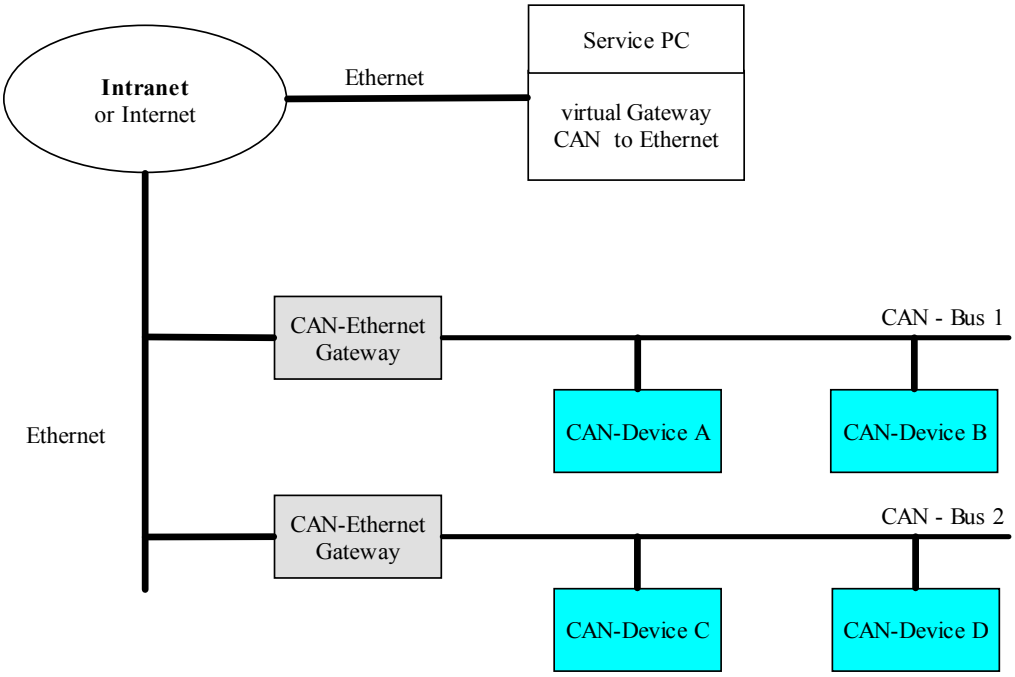

Figure 2: Application Example #2: Remote Diagnostics of CAN Networks on a Service Computer over the Internet

1.3 Delivery Contents

The following parts are included in the CAN-Ethernet Gateway's delivery contents:

- GW-003 CAN-Ethernet Gateway (1* CAN) in housing for DIN rail assembly, including a 2-pin and a 5-pin removable screw clamp connector
- L-1032 System Manual *(this manual is on CD-ROM)*
- CD-ROM with installation program for PCANView (SO-1010), documentation, example configuration files (SO-1027)
- WK041 null-modem cable for configuration of the CAN-Ethernet Gateway via RS-232
- GW-003-2 CAN-Ethernet Gateway (2* CAN) in housing for DIN rail assembly, including a 2-pin and a 5-pin removable screw clamp connector

2 Technical Data

The CAN-Ethernet Gateway has the following technical features and functions:

- Monitors and controls remote CAN networks over the Internet
- Couples two CAN networks
- Gateway configurable via Telnet (remote maintenance) or RS-232
- based on internal file system for configuration data
- capability of executing scripts (e.g. upon start up of the Gateway)
- flexible configuration through implementation of multiple interfaces *(refer to section 4.2)*
- multiple filter mechanisms for CAN messages with the possibility of prioritization
- Generation of a time stamp for CAN messages
- connection to Windows application programs for CAN and CANopen
- 7 LEDs for visualization of the Gateway's state
- generation of CAN error messages
- high data throughput
- 10Base-T interface (10 MBit/s) with RJ45 socket, galvanic isolated
- CAN interface according to CiA[1] DS102, up to 1 MBit/s, high-speed CAN according to ISO11898-1/2, galvanic isolated
- CAN bus connection, D-Sub-9 plug and 5-pin removable screw clamp connector according to CiA DS102 or DeviceNet standard
- Supports CAN specification 2.0A (11-bit CAN identifier) and 2.0B (29-bit CAN identifier)
- RS-232 interface via D-Sub-9, hardware flow control
- Supply voltage 24 VDC +20% -60%, reversed polarity protection
- Current draw approximately 90 mA
- Power connector, 2-position removable screw clamp connector
- Dimensions without connectors, 70 x 100 x 61 (L x B x H) mm³, suitable for DIN/EN rail assembly

[1] CiA, CAN in Automation, international users and manufacturers group

- Protection level: IP20
- Operational temperature range 0°C to +70°C

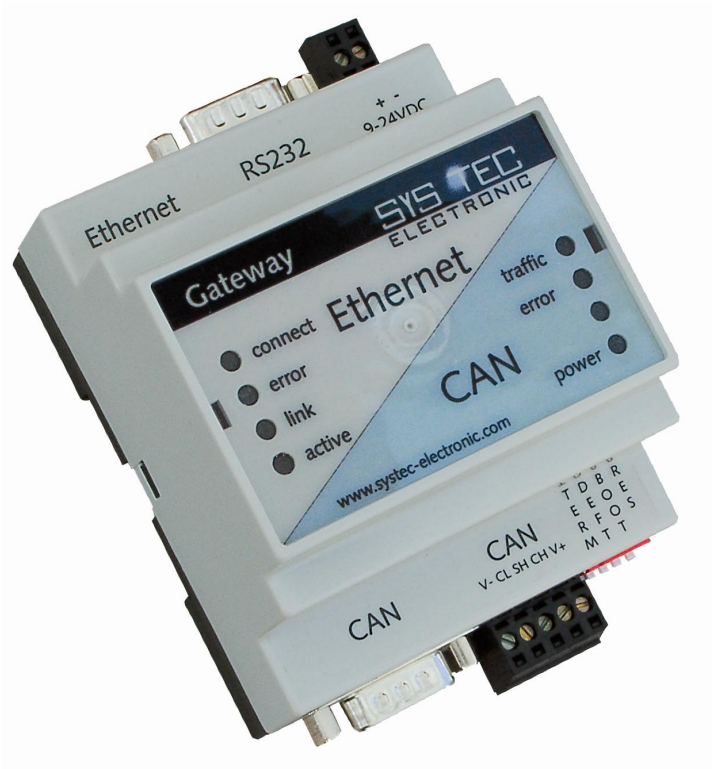

Figure 3: View of the CAN-Ethernet Gateway

3 Getting Started

3.1 Power Supply

A direct current voltage of 24 V –60% to +20% is required to operate the device. The current draw of the device amounts to approximately 90 mA. The supply voltage is connected over a 2-pin, removable screw clamp connector. Labels for the "+" and "–" polarity are printed on the device's connector. Correct connection of the supply voltage is indicated with the green "power" LED.

3.2 Network Connection

3.2.1 Connecting to a CAN Bus

A D-Sub-9 plug is provided for connection of the device to a CAN network. Alternatively a connection via the 5-pin removable screw clamp connector is also possible (suitable connector Weidmueller ‘Omnimate Range – raster 3,5mm’). This connector is wired parallel to the D-Sub-9 plug. Its configuration corresponds to the DeviceNet and CANopen standard.

The supply voltage for the CAN bus (pin 9 at D-Sub-9 or pin 5 at 5-pin socket connector) is not connected in the Gateway. The CAN shield potential is only connected between the two CAN connectors. The CAN bus is galvanic isolated (optically isolated) from the Gateway's internal circuitry.

D-Sub-9 Plug	5-pol.	Signal Name	Description
1		n.c.	not connected
2	2	CL (CAN_L)	CAN_L bus line
3	1	V- (CAN_GND)	CAN Ground
4		n.c.	not connected
5	3	SH (CAN_SHLD)	CAN Shield
6		GND	CAN Ground (optional)
7	4	CH (CAN_H)	CAN_H bus line
8		n.c.	
9	5	V+ (CAN_V+)	not connected

Table 1: CAN Connector Pin Assignment

3.2.2 Ethernet Connection

The Ethernet (10Base-T) signals are routed to an RJ45 socket for easy connection using a standard CAT 3 or CAT 5 network cable. For direct connection (without a hub or switch) of a CAN-Ethernet Gateway and a PC, a crosslink cable is required.

The Ethernet connection is galvanic isolated from the CAN-Ethernet Gateway

Pin	Name	Description
1	TX+	Transmit Data +
2	TX-	Transmit Data -
3	RX+	Receive Data +
4	n.c.	not connected
5	n.c.	not connected
6	RX-	Receive Data +
7	n.c.	not connected
8	n.c.	not connected

Table 2: Ethernet Connector Pin Assignment

3.2.3 RS-232 Interface

The CAN-Ethernet Gateway provides an RS-232 interface with hardware flow control. This interface is connected to a D-Sub-9 plug. This interface allows for configuration of the CAN-Ethernet Gateway. This connector is specifically intended for initial configuration *(refer to section 3.5)*. The RS-232 interface is **not** galvanically separated.

DB-9 Plug, Pin#	Name	Signal Description
1	CD	Carrier Detect
2	RXD	Receive Data
3	TXD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	System Ground
6	DSR	Data Set Ready
7	RTS	Request to Send
8	CTS	Clear to Send
9	RIN	Ring Indicator

Table 3: RS-232 Interface Connector Pin Assignment

The CAN-Ethernet Gateway is connected to the PC via a null modem cable.

3.3 Device Status Display

There are a total of 7 LEDs *(refer to Table 4)* for displaying the operational state of the device. The displays are arranged according to their meaning to the networks *(refer to Figure 3)*. One red and one green LED show the state of the CAN or Ethernet network. Detailed description of the "error" LEDs can be found in *section 6.1*.

LED Name	Description
power	Supply Voltage OK [green]
connect	A connection to the other Gateway is established over UDP or TCP [green]
error (Ethernet)	Error during data transfer on the Ethernet connection (*refer to section 6.1*) [red]
link	Connection to Ethernet established, cabling OK [green]
active	Data transfer over Ethernet [yellow]
traffic	Indicates data traffic on the CAN bus [green]
error (CAN)	Error during data transfer on the CAN connection *(refer to section 6.1)* [red]

Table 4: Meaning of the Device Status LEDs

3.4 Switches

Four DIP-switches are available on the CAN-Ethernet Gateway for configuration purposes of the device. Their function is described in the following table:

Switch #	Name	Description
1	TERM	Defines whether the CAN terminating resistor of 120 Ohm is active or not ON → terminating resistor active/enabled OFF→ terminating resistor not active/disabled
2	DEFT	Defines the start initialization of the CAN-Ethernet Gateway *(refer to section 3.5)* ON → factory default configuration will be loaded OFF→ user configuration will be loaded
3	BOOT	Activate the "Bootstrap-Mode" of the CAN-Ethernet Gateway *(refer to section 8)* ON → "Bootstrap-Mode" will be entered OFF→ Firmware of the CAN-Ethernet Gateway starts
4	RES	Reset the CAN-Ethernet Gateway *(refer to section 8)* ON → Reset active/enabled OFF→ Reset not active/disabled

Table 5: Meaning of the Switches

3.5 Initial Setup and Operation

3.5.1 Standard Configuration

The CAN-Ethernet Gateway has the following standard (factory default) configuration *(Setting the Configuration Script, refer to section 5.1):*

Ethernet/Internet settings

IP address of the CAN-Ethernet Gateway:	192.168.10.111
Subnet- Mask:	255.255.255.0
Standard Gateway:	192.168.10.1
one UDP[2] server	
one TCP server	

CAN settings

CAN bit rate:	1 MBit/s
CAN identifier for error messages:	0xFE

RS-232 interface

Baud rate:	9600 baud
Data bits:	8
Parity:	none
Stop bits:	1
Protocol/flow control:	hardware

[2] BTP: Block Transfer Protocol for transmitting CAN messages via UDP/IP

3.5.2 Initial Configuration using the RS-232 Interface

The CAN-Ethernet Gateway must be configured to meet the specific application requirements before CAN messages can be transmitted. The following steps are required:

- Connect the included null modem cable to the RS-232 interface of the CAN-Ethernet Gateway and a free serial interface on the PC (e.g. COM1).
- Start a terminal program on the PC, the program "Tera Term" will be used for the following examples. This program is available under http://ttssh2.sourceforge.jp/. (If you use a different terminal program, appropriate configuration must be made).
- Set the baud rate and the protocol for the serial interface (e.g. COM2) in Tera Term under the menu item **Setup\Serial Port** *(refer to Figure 4 and Figure 5).*

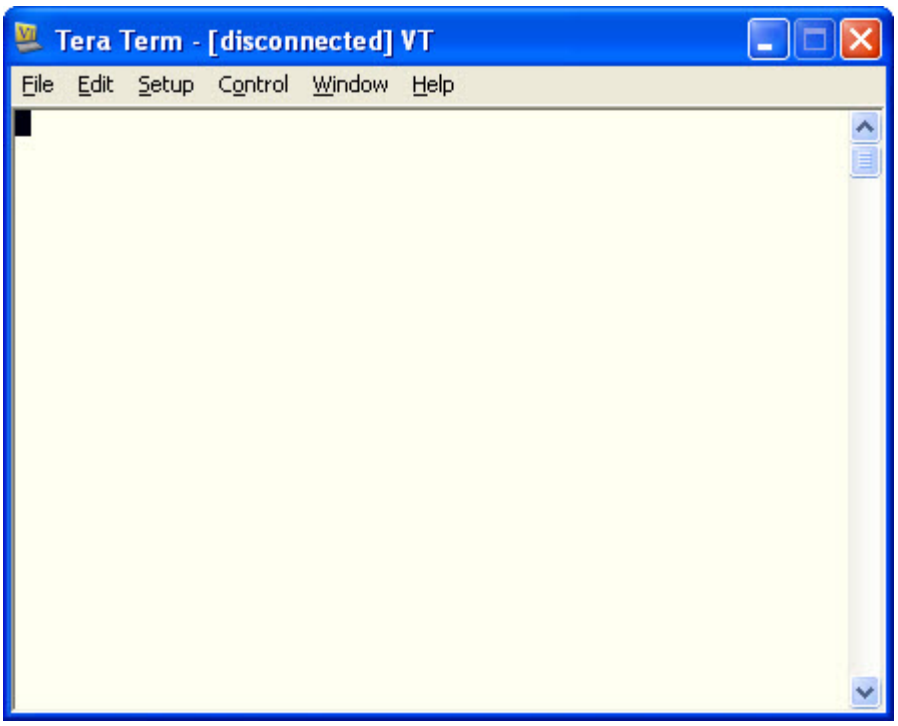

Figure 4: Tera Term Serial Port Configuration (1)

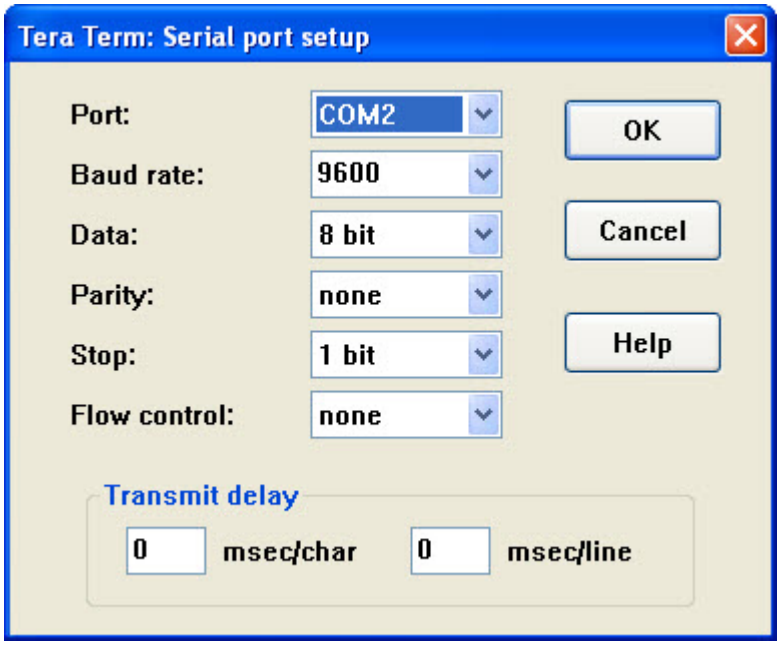

Figure 5: Tera Term Terminal Configuration (2)

- Be sure that you have correctly connected the supply voltage (polarity) and then turn on the power.

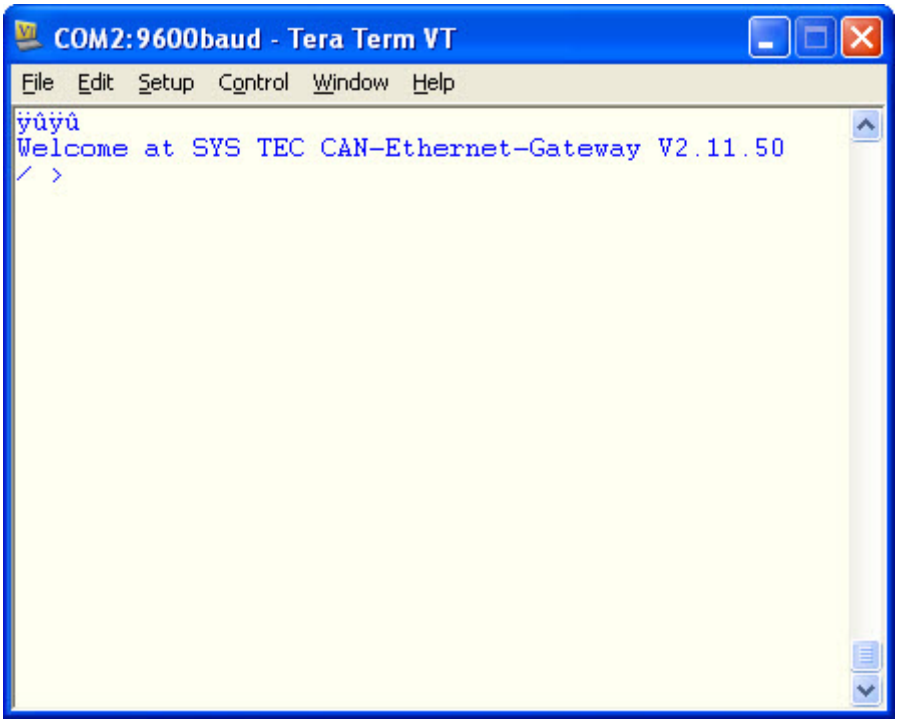

Figure 6: CAN-Ethernet Gateway Start Message

Proper startup of the device is indicated by a (*refer to Figure 6*)

```
Welcome at SYSTEC CAN-Ethernet Gateway V2.11.50
/ >
```

message.

The device then waits for the input of commands. The startup message portion "V2.11.50" represents the current version number and may vary.

Since the CAN-Ethernet Gateway is based on an internal file system, there are commands for navigation. These commands are *(refer to section 4.5)*:

`ls`	to show the contents of the current directory,
`cd`	to switch the current directory
`write`	to create a new file
`rm`	to delete a directory/file
`sync`	in order to store a file in non-volatile memory (EEPROM)

Predefined files are included for initial configuration. After the installation of the software `SO-1046` into the default directory, these files are located under:

`C:\Program Files\SYSTEC-electronic\CAN-Ethernet-Gateway_Utility_Disk\Rc-Files\UDP` (for the UDP transfer) or `C:\Program Files\SYSTEC-electronic\CAN-Ethernet-Gateway_Utility_Disk\Rc-Files\TCP` (for the TCP transfer).

Each folder contains a file for a server configuration `UDP_Server_1CAN.txt` or `TCP_Server_1CAN.txt` and a file for a client configuration `UDP_Client_1CAN.txt` or `TCP_Client._1CANtxt`.

- Before you transfer one of these files to the CAN-Ethernet Gateway you must modify the file. Open the file `TCP_Client_1CAN.txt` with an editor.

- Modify the local IP address, the subnet mask and the standard Gateway address to the values that correspond to your application requirements *(refer to section 4.5.12)*.
  For example:
  `ipcfg 192.168.10.179 255.255.255.0 192.168.10.1`

- Set the CAN bitrate and the CAN identifier for error messages *(refer to section 4.2.4)*.
  For example: to 1000 kBit/s and CAN identifier 0xFD
  `/if/can0 bus:0 baud:0 canid:FD on`

- Save the configuration file under a different name.
  For example:
  `TCP_Client_1CAN_179.txt`

The generated configuration file is written to the CAN-Ethernet Gateway as follows:

1. Change to the directory `/save`
   `/ >cd /save↵`
   `/save >`

2. Erase the existing configuration file `rc`
   `/save >rm rc↵`
   `/save >sync↵`
3. Write a new configuration file `rc`
   `/save >write rc↵`
   After you've input the command you can send a file to the CAN-Ethernet Gateway using the HyperTerminal tool. The following images show how to do this:

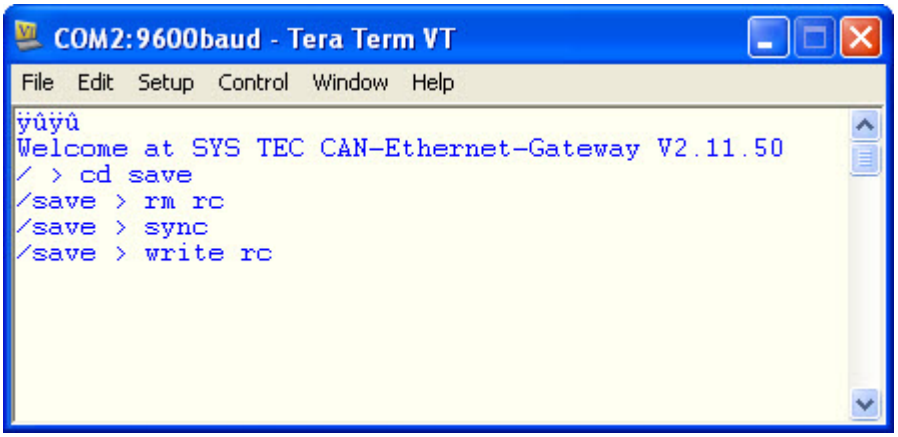

Figure 7: Sending a Configuration File via Tera Term

Please select the menu item **File\Send File**

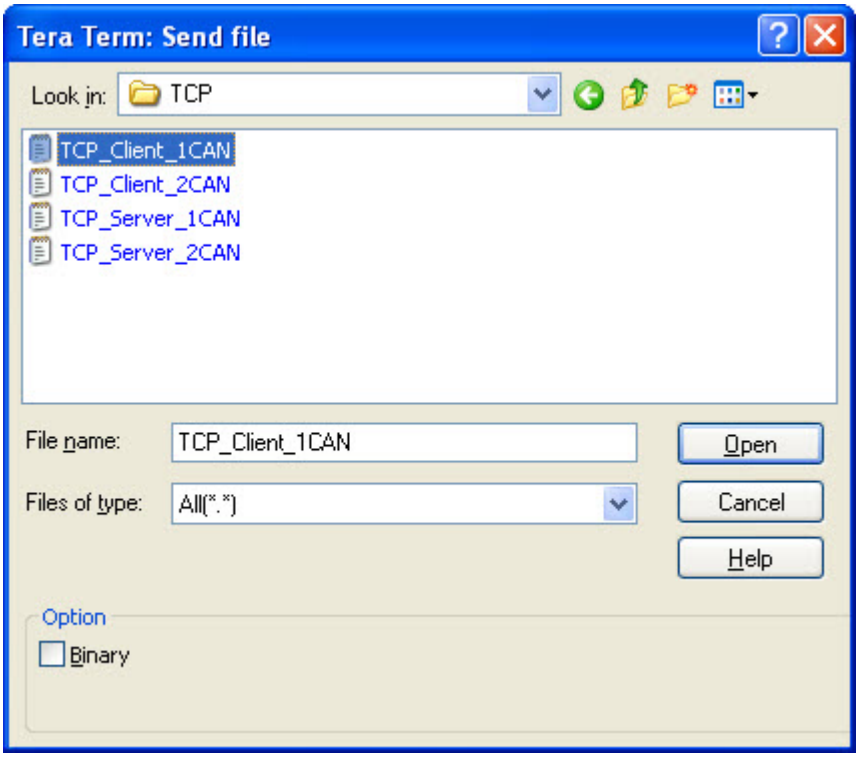

Figure 8: Selecting a Configuration File

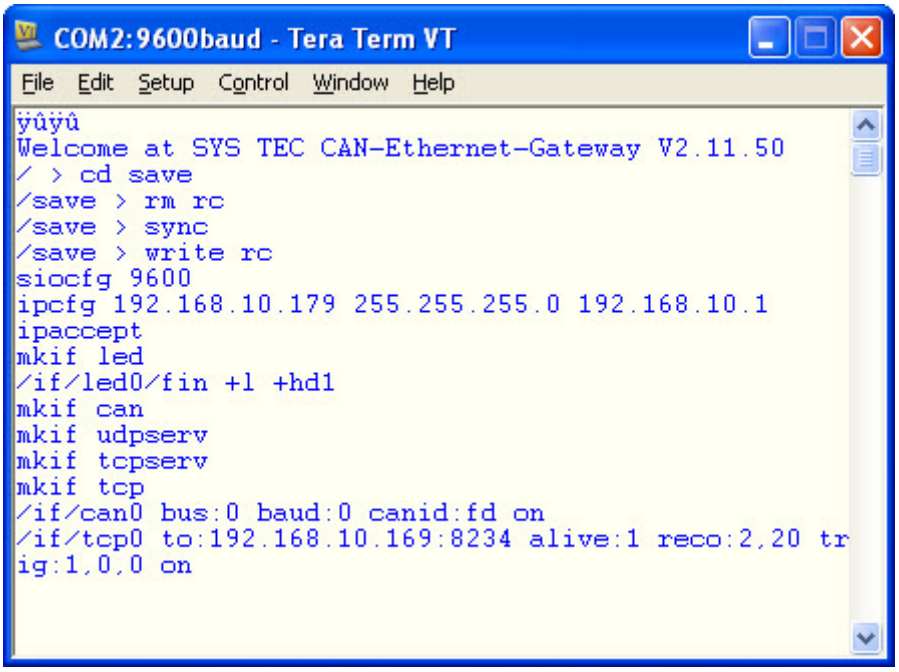

Figure 9: Successful Transmission of a Configuration File

4. To end the transfer push `Ctrl+D`. The Gateway will show the execution with the prompt
   `/save >`

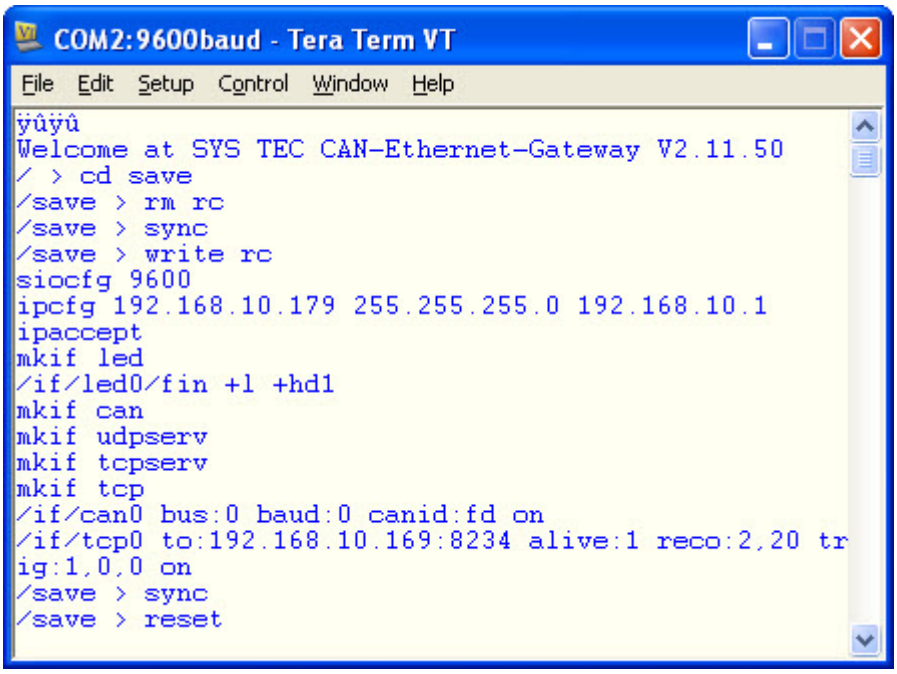

Figure 10: Finishing the Configuration

5. Save the file `rc` with the command `sync` in EEPROM.

6. Set the switch 2 "DEFT" to OFF and restart the CAN-Ethernet Gateway. Upon restart of the CAN-Ethernet Gateway with power-on or via the command `reset`, the saved configuration file will be executed and the CAN-Ethernet Gateway will be configured.
   `/save >reset↵`

7. Verify the new configuration by inputting the command `ipcfg` *(see **Fehler! Verweisquelle konnte nicht gefunden werden.**)*. The parameter IP address, subnet mask and standard gateway stored in the configuration file all correspond to their settings. Now the Gateway is configured and can be accessed over the Ethernet (e.g. Telnet).

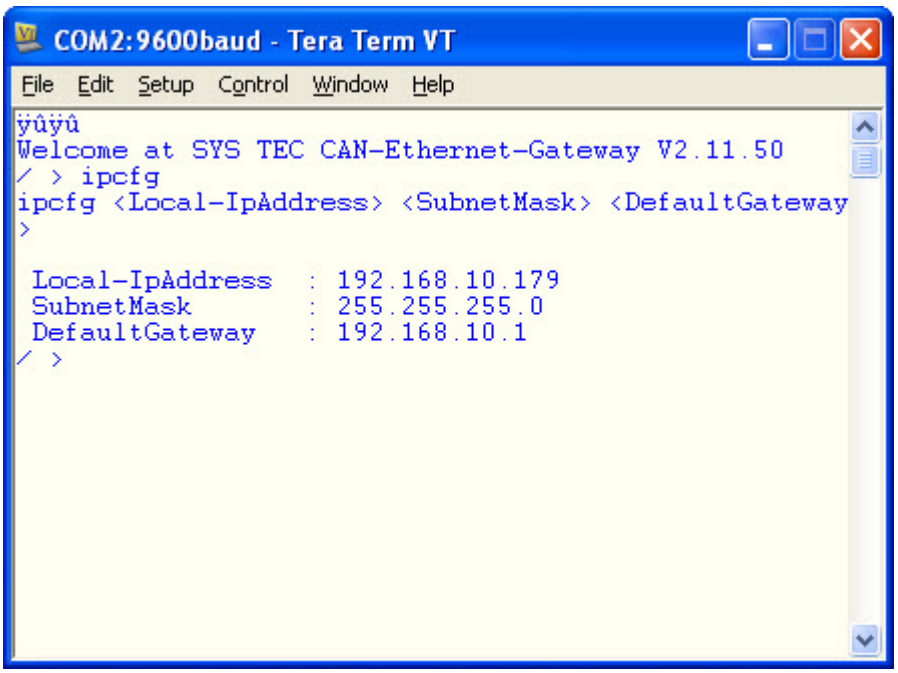

Figure 11: Verifying the Selected Configuration

This concludes the initial start up of the device.

The CAN-Ethernet Gateway functions with different interfaces *(refer to section 4.2),* which enable the data exchange over the Ethernet and CAN. Interfaces are created with the command `mkif` and deleted with the command `rm`. All created interfaces appear in the folder `/if`. An interface can be addressed and configured via a direct call to the directory *(refer to section 4.2.4)*.

3.5.3 Configuration and Operation using Telnet

Configuration of the CAN-Ethernet Gateway during operation is also possible via Telnet (TCP port 23). The function parameters are the same as those of the RS-232 interface. Use of Telnet also enables configuration of remote CAN-Ethernet Gateways. A condition for this, however, is the initial configuration of the IP address with the command `ipcfg` *(refer to section 4.5.12)*. Without this initial configuration of the IP address the CAN-Ethernet Gateway can only be addressed via its standard configuration *(refer to section 3.5.1)*.

A Telnet client is already included in the Windows package. This Telnet client can be called via `telnet <Address>`. Under Linux the programs `telnet` or `netcat` can be used.

4 Device Functions

4.1 Overview

The CAN-Ethernet Gateway has multiple interfaces for operation and control. The principle structure is shown in the following figure:

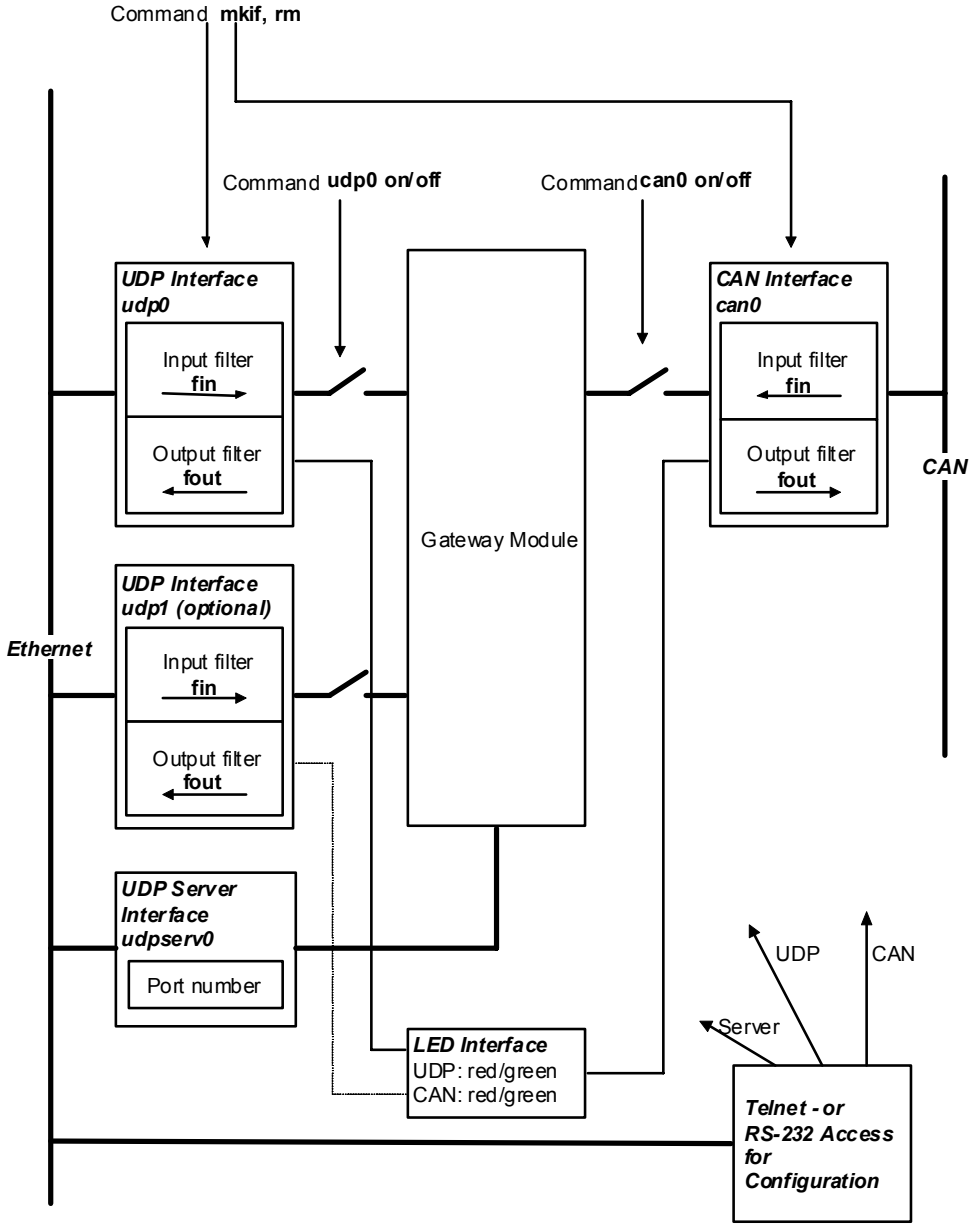

Figure 12: CAN-Ethernet Gateway Functional Overview

4.2 Interfaces

4.2.1 Fundamentals

CAN messages are exchanged over interfaces. An interface establishes a connection between a CAN message input/output and the central distributor (data pool in the Gateway module) in the CAN-Ethernet Gateway. Multiple interfaces can be activated, as long as enough memory is available on the Gateway.

The two most important interfaces are `can` for the CAN interface and `udp` or `tcp` for an Ethernet interface according to the block transfer protocol. While the CAN interface sends and receives CAN messages to and from the CAN network, the UDP interface is responsible for tunneling the messages over UDP/IP/Ethernet.

Available Interfaces	Type	Maximum amount	Requirements
CAN	can	1	required, mandatory
UDP Client	udp	3	required based on application needs
UDP Server	udpserv	1	required, mandatory
TCP Client	tcp	3	required based on application needs
TCP Server	tcpserv	1	required, mandatory
LED	led	1	required, mandatory

Table 6: Interface Overview

The UDP interface transports the CAN messages based on the UDP protocol while the TCP interface uses TCP as the transport protocol. Both interface types are functionally identical but vary in terms of transfer rate.

Interfaces are created with the command `mkif` and deleted with the command `rm`. All interfaces that have been created appear in the `/if` directory. Within the interface directory there are 3 files. The files `fin` and `fout` are responsible for CAN message filtering *(refer to section 4.3)*. The file `conf` is intended for future expansions and currently remains unused.

An interface can be addressed by opening the interface directory.

```
/if/<if-name> {<option>}
```

e.g.

```
/if/can0↵
```

or also

```
cd /if/<if-name>
. {<option>}
```

e.g.

```
/if/can0 canid:123↵
```

After an interface has been created, the current settings can be queried by entering the command interface name + number.

e.g.

```
/if/can0↵
Usage: /if/can0 [bus:<num>] [baud:<num>]
       [userbaud:<hex>] [canid:<hex>] [on] [off]

Current settings: bus:0 baud:0(1MBit/s) userbaud:0x0000
                  canid:0xF6 on state:0x0
```

4.2.2 UDP/TCP Server Interface

The UDP server interface (`udpserv`) awaits UDP connection queries from another Gateway and then creates a new UDP connection, which is used for the CAN message exchange. Likewise, the TCP server (`tcpserv`) awaits TCP connection queries and then creates a TCP interface for communication if required.

The UDP/TCP interfaces created in this manner are removed by the UDP/TCP server when the connection is ended or aborted. In order to be able to react to TCP- as well as UDP-based queries, a TCP and a UDP server interface both have to exist on the Gateway.

A UDP server interface is created with the command `mkif udpserv` using the next available interface number. The directory name is chosen automatically and is `/if/udpserv0` for the first interface.

A TCP server interface is created with the command `mkif tcpserv`↵ within `/if/tcpserv0`.

Possible options:

```
<option> ::= {port:<num>} |
             {trig:<cnt>[,<time>][,<inhibit>]}
```

`<num>` contains the port number where the interface awaits queries (default: 8234)

```
<trig> ::= trig:<cnt>[,<time>][,<inhibit>]
```

These parameters can be used to control the transmission of CAN messages in TCP/UPD packets. A trigger threshold is set which enables the transmission of a specific minimum number of CAN messages in a UDP/TCP packet. To ensure that the packets are transmitted after a maximum amount of time, even if the minimum number of CAN messages has not been reached, an additional timer event can initiate the transmission of the UDP/TCP packet. These parameters are applicable for

connections, which are established automatically over this UDP/TCP server.

`cnt` Number of CAN messages that have to at least be present in the internal buffer for transmission via Ethernet, before an Ethernet packet is sent
Value range: 1 - 127
Default: 1

`time` time in milliseconds [ms] that can have passed since the last transmission before the next Ethernet packet is transmitted.
Value range: 0 - 4294967295 [ms]
Default: 0

`inhibit` time in milliseconds [ms] to delay the transmission of the next Ethernet packet (used for slower connections or slower network devices such as router or switches)
Value range: 0 - 100 [ms]
Default: 0

The following conditions apply for the transmission of an Ethernet packet:

Number of received CAN messages >= `cnt`
OR
time difference between the last transmission time and the current time > `time`
OR
time difference between the last Ethernet transmission time and the current time > `inhibit`.

Output of the current configuration:

```
/if/udpserv0↵

Usage:
udpserv0 port:<num> [trig:<cnt>[,<time>]>[,<time>]]]
Current settings: port:8234 trig:1,0,0
```

4.2.3 UDP/TCP Client Interface

The UDP client interface makes a tunnel over UDP/IP/Ethernet available in order to send or receive CAN messages. With the TCP client interface the tunnel operates over TCP/IP/Ethernet. The UDP/TCP client interface is instanceable, so that multiple, but not more than 3 connections are possible simultaneously.

A UDP client interface is created with `mkif udp 0` using the interface number 0 or the next free number. It is used to establish an active connection to another Gateway. The directory name is selected automatically and is `/if/udp0` for the first interface.

A TCP client interface is created with the command `mkif tcp 0`.

Possible options:

```
<option> ::= { <addr> | <switch> | <alive> |
               <reco>}|<trig>|<inhibit> }

<addr> ::= to:<ipaddr>[:<port>]
```

setting the target IP address and the port of the UDP server to be communicated with (default: IP address 192.168.10.111, port 8234) for an active connection established by the Gateway

```
<switch> ::= {on | auto | off}
```

- `on` starts active connection establishment
- `auto` starts establishing a connection as soon as there are CAN messages to be sent.
- `off` stops the connection, any data that has not been transferred will be discarded (default).

`<alive> ::= alive:<num>`

0 switches off the cyclical testing of the communication paths. An absence of communication parameters is not recognized, the effected interface must then be removed manually.

1 turns on the cyclical testing of the command paths (default for UDP and TCP).

`<reco> ::= reco:<num>,[<time>]`

`0` switches off the automatic reconnection function

`1` switches on the automatic reconnection function after some CAN messages are enqueued.
The connection is established respectively reestablished after some CAN messages are enqueued in the CAN receive buffer.

`2` switches on the immediately automatic reconnection function (default).
The connection is established respectively reestablished immediately, CAN messages do not have to receive.

`time` number of seconds waited until a new connection is attempted (default: 120)

`<trig> ::= trig:<cnt>,[<time>],[<inhibit>]`

These parameters can be used to control the transmission of CAN telegrams within TCP/UPD packets. A trigger threshold is set which enables a specified minimum number of CAN messages to be transmitted in a single UDP/TCP packet. In order to make sure that the packets are transmitted after a maximum time, even if the minimum number of CAN telegrams has not been reached, an additional timer event can initiate the transmission of the UDP/TCP packets. These parameters are applicable for

connections, which are established automatically over this UDP/TCP server.

`cnt` number of CAN messages which have to at least be present in the internal buffer for transmission via Ethernet before an Ethernet packet is sent
Value range: 1 - 127
Default: 1

`time` time in milliseconds [ms] that can have passed since the last transmission before the next Ethernet packet is transmitted.
Value range: 0 - 4294967295 [ms]
Default: 0

`inhibit` time in milliseconds [ms] to delay the transmission of the next Ethernet packet (used for slower connections or slower network devices such as router or switches)
Value range: 0 - 100 [ms]
Default: 0

The following conditions apply for transmitting a UDP/TCP packet:

Number of received CAN telegrams >= `cnt`
OR
time difference between the last transmission time and the current time > `time`
OR
time difference between the last Ethernet transmission time and the current time > `inhibit`.

Output of the current configuration:

```
/if/udp0↵

Usage:
udp0 [to:<ipaddr>[:<port>]] [on|auto|off] [alive:<num>]
[reco:<num>[,<time>]] [trig:<cnt>[,<time>[,<inhibit>]]]

0 frames sent, 0 frames recv'd, state: Connection
establishment in process

Current settings:
to:192.168.10.118:8234 alive:1 reco:2,120 trig:1,0,0
```

4.2.4 CAN Interface

A CAN interface is generated by `mkif can 0` with the interface number 0. The directory name is selected automatically and is `/if/can0` for the first interface. Only one CAN interface can be active.

Possible Options:

```
<option> ::= { <busid>|<bauidx>|<userbaud>|<canid>|
               <switch> }

<busid> ::= bus:<num>
```

`num` determines the CAN bus connection that is to be used for the interface (default: 0)
GW-003 supports a CAN-Bus: num = 0
GW-003-2 CAN-Ethernet Gateway supports two CAN bus interfaces, value range: 0 <= num <=1

```
<baudidx> ::= baud:<num>
```

`num` sets the transfer rate corresponding to the baudrate index according to CiA DSP305[3]

```
0     = 1000 kBit/s  (default)
1     =  800 kBit/s
2     =  500 kBit/s
3     =  250 kBit/s
4     =  125 kBit/s
5     =  100 kBit/s
6     =   50 kBit/s
7     =   20 kBit/s
8     =   10 kBit/s
FF    = Use value from user baudrate
```

[3] CiA DSP305: CAN in Automation Draft Standard Proposal CANopen Layer Setting Services and Protocol

`<userbaud> ::= userbaud:<hex>`

`hex` Sets the user specific baudrate value `<hex>` for initialization of the CAN controller. This enables to set baud rates which are not standard baud rates according to CiA DSP305. To use the user baudrate it is necessary to set the baudrate index to value ***FF***.
Value range: 0 < `hex` <= FFFF
Default: 0

CAUTION!:
The CAN baudrate must be given as a hexadecimal value without special characters.
e.g. ***2F41*** corresponds to a resulting CAN baudrate value of ***0x2F41*** or ***12097*** dec.

`<canid> ::= canid:<hex>`

Sets the CAN identifier `<hex>`, which is used when error messages are sent.
Value range: 0 <= `hex` <= 7FF
Default: FE

CAUTION!:
The CAN identifier must be given as a hexadecimal value without special characters.
e.g. ***123*** corresponds to a resulting CAN identifier of ***0x123*** or ***291*** dec.

`<switch> ::= {on | off}`

`on` initializes the CAN driver
`off` switches off the CAN driver (default)

Only 11-bit CAN identifiers are supported.
Construction of the error message is described in *section 6.3*.

`0` switches off the transmission of error messages in the CAN-Ethernet Gateway

`1`

through

`7FF` sets the CAN identifier to the given value.

Output of the current configuration:

```
/if/can0↵
Usage: can0 [bus:<num>] [baud:<num>] [userbaud:<hex>]
       [canid:<hex>] [on] [off]

Current settings: bus:0 baud:0(1MBit/s) userbaud:0x0000
                  canid:0xF6 state:0x0 on
```

4.2.5 LED Interface Status Display

The LED interface is used to indicate the operational states for CAN and Ethernet using the LEDs. The UDP or TCP interfaces displayed on the status LEDs have to be selected. No selection is possible for CAN, but the interface has to be configured.

An LED interface is generated with `mkif led`. The directory name is selected automatically and is `/if/led0` for the interface.

It is possible to select which LEDs go with which interface using the adjustable filter defines (*refer to section 4.3*). Example:

With the setting

```
/if/led0/fin +hd1 +ld23
```

the states of interface 1 are assigned to the LEDs for CAN error and CAN traffic (identified by `hd`) and the states of interface 2 and interface 3 are displayed on the BTP error and BTP connection LEDs. The states of other interfaces are not displayed. The interface IDs can be accessed with the command `ls` *(refer to section 4.5.2)*.

Default setting (the CAN interface has the interface number 1):

```
/if/led0/fin +hd1 +l
```

Possible options:

```
<option> ::={<reset>}
```

`reset` re-initializes the LEDs

4.3 Filtering

4.3.1 Concept of Message Filtering

The filtering is based on the screening of CAN identifiers. The CAN-Ethernet Gateway processes the data from the Ethernet or CAN in a data pool. The filter rules determine which messages will be passed on from this data pool. Thus data traffic can be reduced if, for example, only messages from a certain group of CAN identifiers (CAN IDs) are passed on. Furthermore, individual CAN identifiers can be selected for highly specific filtering schemes.

The filter rules can be determined by calling an interface's filter files. If no filter rules are present then all messages are passed on.

4.3.2 Input Filter

The input filter determines which messages will be accepted by the interface input (`fin`). For example, with a CAN interface only the messages whose CAN IDs are established in the `fin` function will reach the CAN-Ethernet Gateway.

4.3.3 Output Filter

The output filter determines which messages from the Gateway’s data pool will be passed on (`fout`).

4.3.4 Filter Description (Syntax)

Format: `fin <default> {<rule>}*`

```
<default> ::= {+ | -}[{l | h}]
<rule> ::= {+ | -}[{l | h}][r][d]{0-9}* [<idl> [<idh>]]
```

Definition of filter rules:

`+`	the message is accepted
`-`	the message is discarded
`h`	the message is assigned a high priority, as long as it is supported by the interface (LED interface: errors are signaled by CAN LEDs)
`l`	the message is given a low priority (LED Interface: errors are signaled by Ethernet LEDs)

Determining the area of application for a filter rule:

`<idl>`	first CAN identifier (as hexadecimal number) for which the filter rule applies. If no CAN identifier is given, the filter rule can apply for any CAN identifier.
`<idh>`	last CAN identifier (as hexadecimal number) for which this filter rule applies. If only one CAN identifier is given, then the filter rule only applies for this CAN identifier.
`r`	this rule applies for CAN RTR[4] frames
`d`	this rule applies for CAN data frames
`0-9`	this rule applies for messages that come from the interface with the interface ID "x" *(to determine the interface ID with the command **`ls`**, refer to section 4.5.2)*. If no number is given then all interfaces are accepted.

If individual CAN identifiers are filtered then `<idh>` is not required.

[4] RTR: Remote Transmission Request

Example 1:

```
cd /if/can0
/if/can0 > fout +l +hrd0123 10A 200 -r23 300
```

This rule results in RTR frames and data frames, which are received by the CAN-Ethernet Gateway and originate from interfaces 0 through 3 and have a CAN identifier between 0x10A and 0x200 (inclusive), are sent as high priority. Furthermore, all RTR frames from interfaces 2 and 3 with the identifier 0x300 are discarded. All remaining messages originating from the CAN-Ethernet Gateway are sent with low priority.

Example 2:

```
cd /if/can0
/if/can0 > fin - +ld 400 400 +ld 2c0 2c0
```

Only CAN messages with the CAN identifiers 0x400 and 0x2C0 are passed on from the CAN interface to the other interfaces.

Example 3:
File rules can be combined.

```
/if/can0 >fin + -ld 180 57F
/if/can0 >fin -ld 700 700
```

All CAN messages from the CAN interface are passed on to the other interfaces, with the exception of the CAN identifiers between 0x180 and 0x57F and those with the CAN identifier 0x700.

By deleting the filter file with the command `rm`, the filter rules are also deleted. The file `fin` is retained.

The configured filter rules can be displayed by calling the filter file.

Example output for the configuration of example 3:

```
/if/can0 >fin
Usage:
     fin
     if-id: 1
     Current filter:
     -lds 180 57f
     -lde 180 57f
     -lds 700 700
     -lde 700 700
     default: +l
     /if/can0 >
```

Note:
The CAN identifier is given in hexadecimal format, without indication of a preceding 0x or a subsequent h.

4.4 File System

4.4.1 Structure

The CAN-Ethernet Gateway works with an internal file system. This allows for configuration modifications on the Gateway during runtime and to control the automatic configuration via script files. Furthermore, there is the possibility of storing the files as non-volatile data in an EEPROM.

The structure of the file system is shown in *Figure 13* and has the following predefined structure:

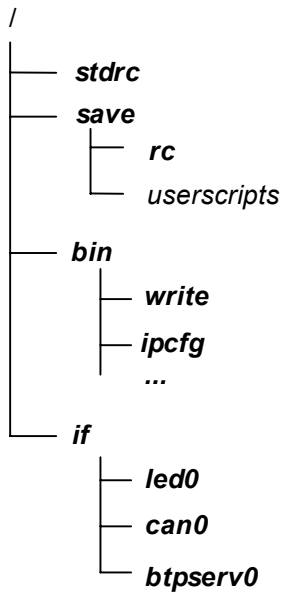

Figure 13: File System Structure

The file system is located in the RAM. After a power failure or reset the file system is initialized with the default configuration *(refer to section 5)*.

The `/save` directory represents a special case. All files contained therein can be stored in EEPROM. There are commands available for navigation within the file system *(refer to section 4.5)*.

4.4.2 Data Storage in EEPROM

There is an EEPROM available for storage of the configuration data. All files that are to be stored as non-volatile data must be located in the `/save` directory. To store files in the EEPROM, the process must be started manually by the user with a call of the `sync` command *(refer to section 4.5.8)*.

The files in the `/save` directory can be modified by the user. The most important file in this directory is the configuration script `rc`. This file is executed when the firmware is started (under the condition that the switch "DEFT" = OFF) and contains all of the configuration data for the CAN-Ethernet Gateway *(refer to section 5.3)*.

4.5 Command Set Description

For general functions like changing directories or displaying data, Unix commands are used. The following section describes the available commands. Options (e.g. `ls -l`) are not available however.

4.5.1 cd

Format: `cd <dir>`

`cd ..` changes into the specified directory or into the next higher level

Description: The command `cd` is used to switch into the specified directory `<dir>` or into the next higher directory level.

4.5.2 ls

Format: `ls [<dir>]`

Description: The command `ls` shows the files of the indicated or current folder. The interface ID, which is required for displaying the filter rules, is given along with the command.
See example, the interface can0 has the interface ID 1 (`id:1`), the interface udp0 was assigned the interface ID 4 (`id:4`).

Example:

`/ >ls /if↵` displays the folder `/if`

```
 ..                typ:0xc2
 led0           use:0 typ:0xa0 ch:4 id:0
 can0           use:0 typ:0xa0 ch:4 id:1
 udpserv0       use:0 typ:0xa0 ch:4 id:2
 tcpserv0       use:0 typ:0xa0 ch:4 id:3
 udp0           use:0 typ:0xa0 ch:4 id:4
```

4.5.3 mkif

Format: `mkif <type> <if-num> [<if-name>]`

Description: The command `mkif` creates a new interface of type `<type>` and integrates it in the Gateway's processing chain. The interface appears under the given name `<if-name>` or under the type name with a number suffix within the `/if` folder. The number `<if-num>` is given continuously. The available interfaces are described in *section 4.2*.
The interface can be removed using the `rm` command *(refer to section 4.5.5)*.

The following files are created:

`/if/<if-name>/..` Link to parent directory

`/if/<if-name>/conf` Static configuration of the interface (format is interface specific). This file is intended for future expansions and is currently unused.

`/if/<if-name>/fin` Configuration of the input filter, i.e. for messages from the interface to the Gateway

`/if/<if-name>/fout` Configuration of the output filter, i.e. for messages from the Gateway to the interface

Example:

`/ >mkif tcp 0 tcpclient` creates an interface with the name `tcpclient`

`/ >mkif can` creates a CAN interface with the name `can0`

4.5.4 mem

Format: `mem`

Description: The `mem` command returns the available or free memory resources. Care must be taken when creating new interfaces to ensure that sufficient memory resources are available.
The following memory areas are available:

`Eeprm`	data in EEPROM for non-volatile storage of files
`lwIP`	internal memory of the TCP/IP stack
`SysSt`	system stack
`UsrSt`	user stack
`Gcm`	gateway application

Example:

```
/ > mem↵

Free file handles: 40
Memory  total   free   max  blocks
Eeprm:   2026   1668   1442  2
lwIP :   8192   8090   8090  1
SysSt:   2410   2148   1908  2
UsrSt:   9000   7058   6658  2
Gcm  : 150000 144792  65528  5
```

4.5.5 rm

Format: `rm <fname>`

Description: The `rm` command is used to delete files, directories and interfaces. The descriptor `<fname>` specifies the name of the item to be removed. When deleting files in the EEPROM, the `sync` command (*refer to section 4.5.8*) must be called following the call of `rm` in the `/save` directory.

Example:

`/ >rm /save/rc↵`	deletes the configuration file `rc` in the `/save` directory
`/ >sync`	writes `/save` directory into EEPROM

4.5.6 write

Format: `write <fname>`

Description: With the command `write`, an ASCII file is created and given the name `<fname>`. If an existing file is to be modified, this should be indicated with `cat`, a file is deleted with `rm` and modified accordingly via the intermediate file with `write`. The command is closed with two empty lines and the key combination `Crtl+D`. The use of the command is described in *section 3.5*.

4.5.7 cat

Format: `cat <fname>`

Description: The `cat` command returns the contents of the file `<fname>`.

Example: display the contents of the standard configuration file

```
/ > cat stdrc↵

siocfg 9600
ipcfg 192.168.10.111 255.255.255.0 192.168.10.1
mkif led
/if/led0/fin +l +hd1
mkif can
mkif udpserv
mkif tcpserv
/if/can0 baud:0 canid:fe on
/ >
```

4.5.8 sync

Format: `sync`

Description: The `sync` command writes files located in the `/save` directory that are not yet saved into the EEPROM. Thus the files are available again following power-on or a reset *(refer to section 4.4.2)*.

4.5.9 version

Format: `version`

Description: The `version` command returns the current firmware version number for the CAN-Ethernet Gateway and the MAC[5] address.

Example:

```
/ > version↵
 SYSTEC CAN-Ethernet-Gateway V2.11.50
 MAC: 0x00:0x40:0xDC:0x00:0x0E:0xD3
/ >
```

4.5.10 exit

Format: `exit`

Description: The `exit` command terminates an ongoing Telnet session. The same can be achieved by entering `Crtl+D`.

4.5.11 reset

Format: `reset`

Description: The `reset` command resets the CAN-Ethernet Gateway (software reset). Depending on the "DEFT" DIP switch position the corresponding configuration script will be loaded.

[5] MAC: Media Access Control: Unique Ethernet address for each CAN-Ethernet Gateway. Each device with an Ethernet controller has its own MAC address.

4.5.12 ipcfg

Format: `ipcfg [<local-ip> <mask><gw>]`

Description: The `ipcfg` command configures the IP address, the subnet mask and the standard gateway for the Ethernet connection. The TCP/IP stack is initialized with the provided values. The command `ipcfg` must be called prior to use of the Telnet and the BTP interface, since otherwise no communication is possible over the Ethernet interface. It therefore has to be the first command in the configuration script `/save/rc` and must also have a unique IP address in the network. The current configuration can be displayed by entering the command without parameters.

Example:
Configuring the device parameters

```
/>ipcfg 192.168.10.117 255.255.255.0 192.168.10.1↵
```

Displaying the current configuration

```
/>ipcfg ↵

    ipcfg ipcfg <local-ip> <mask> <gw>
    local-ip:  192.168.10.117
    mask:      255.255.255.10
    gw:        192.168.10.1
```

4.5.13 siocfg

Format: `siocfg <Baudrate>`

Description: The command `siocfg` switches on the RS-232 interface for the command input and configures the baud rate `<Baudrate>` being used. The following baud rates are supported:
4800Baud
9600Baud
19200Baud
38400Baud
57600Baud
115000Baud
The parameters hardware flow control, 8 data bits, one stop bit and no parity are preset.

Example:
```
/ >siocfg 57600↵
```

4.5.14 ipaccept

Format: `ipaccept -<param> [<IP address>]`

Description: The command `ipaccept` allowed the configuration of an IP address filter. From each IP address in the filter the gateway can accept incoming connections. If the IP address filter is empty, the gateway can accept incoming connection from every IP address in the network with the same default gateway and relevant subnet mask and network class (Class A, Class B, Class C network).

`ipaccept -a <IP address>.`

Use this command to add an IP address to the filter.

`ipaccept -d <IP address>.`

Use this command to determined IP address from the filter.

`ipaccept –all`

Use this command to show the complete IP address filter. All the configured IP addresses in the filter will be printed out to the console window.

Example:
Add an IP address to the filter

```
ipaccept –a 192.168.10.111
```

Delete an IP address from the filter

```
ipaccept –d 192.168.10.156
```

Show the complete IP address filter

```
ipaccept –all

IpAddress table

192.168.10.116
192.168.10.178
…
…
```

5 Gateway Configuration

5.1 Fundamentals

Configuration of the CAN-Ethernet Gateway can be accomplished in two ways; using the RS-232 interface with a terminal program *(refer to section 3.5.2)* or a Telnet connection over the Ethernet connection *(refer to section 3.5.3)*.

The CAN-Ethernet Gateway is configured via configuration scripts. There are the following two configuration scripts:

`/stdrc` Standard configuration,

- ✓ it is fixed in the firmware;
- ✓ is executed if the switch "DEFT" is set to ON or if no ***`/save/rc`*** exists
  or if an error has been detected in the EEPROM or if a firmware update has been performed
- ✓ it contains the following entries:

```
siocfg 9600
ipcfg 192.168.10.111 255.255.255.0
192.168.10.1
mkif led
/if/led0/fin +l +hd1
mkif can
mkif udpserv
mkif tcpserv
/if/can0 baud:0 canid:fe on
```

The CAN-Ethernet Gateway is in a passive state and waits for connection queries following a start with the standard configuration.

`/save/rc` customer specific configuration;

- ✓ can be established with the commands `write` and `sync` and is saved in EEPROM
- ✓ is executed if the switch "DEFT" is set to OFF

- ✓ is established by the customer based on the application requirements
- ✓ there are example configuration scripts for client and server applications *(refer to section 3.5.2)*

The CAN-Ethernet Gateway is in the set state following a start with the customer specific configuration and can attempt establishing a connection or wait for a connection to be established.

The corresponding configuration script is selected and executed in the event of a power-on or software reset, according to the requirements listed above.

5.2 Example for a Customer-Specific Configuration Script

The following overview shows a series of commands and parameters as used in the example file `rc`.

`siocfg 9600`	Switching on the RS-232 interface and setting the baud rate
`ipcfg 192.168.10.117 255.255.255.0 192.168.10.1`	Setting the IP address, subnet mask and standard Gateway
`mkif led`	Creating the LED interface for status display
`/if/led0/fin +l +hd1`	Configuration of the LED interface
`mkif can`	Creating a CAN interface
`mkif udp`	Creating a UDP client interface for establishing an active connection to another Gateway
`mkif udpserv`	Creating a UDP server interface
`/if/can0 baud:0 canid:fe on`	Configuring the CAN interface for 1 Mbit/s with CAN error message, CAN interface is switched on
`/if/udp0 to:192.168.10.115 on`	Configuring the UDP client interface with target IP address entry, the interface is switched on and starts the connection

5.3 Creating a Configuration Script

The following steps are necessary for creating a configuration script.

Output of the existing configuration file:

```
cd /save
cat rc
```

The output can be marked and added in an editor. "Notepad" is a particularly good editor. The configuration can then be modified in the editor and adapted to the specific requirements.

Before writing the configuration back to the device, the file `/save/rc` must be deleted with command `rm rc`.

With `write rc` the input of the configuration is started. It is transferred to the Gateway by being marked in the editor and then copied to the terminal window. The input of the command `write` is finalized with `Crtl+D` key combination.

Using the command `cat rc` the configuration can be checked one more time. Finally the data must be stored in the EEPROM with the command `sync`. Restarting the Gateway activates the new configuration (the switch "DEFT" must be set to OFF).

5.4 Resetting the Device to its Standard Configuration

For local maintenance tasks, if an unknown state of the CAN-Ethernet Gateway or a faulty configuration is detected, then it is necessary to be able to communicate with the CAN-Ethernet Gateway at a fixed IP address. The IP address 192.168.10.111 is used for the standard configuration.

In order to set the device to its default IP address 192.168.10.111, the switch "DEFT" must be set to ON. Now perform a power-on reset by briefly interrupting the supply voltage.

Note: Saved configurations (`/save/rc`) are not deleted.

As an alternative, access over the RS-232 interface is possible, whereby access to the CAN-Ethernet Gateway occurs independently of the IP address.

5.5 Assigning Passwords

There is also the possibility of creating a password for access via Telnet and RS-232. To do this the file `passwd` located in the `/save` directory is used. The content of this file corresponds to the password. The file is created using the commands `write` and `sync`.

Example:

`/ >cd save`	change to the `/save` directory
`/save >write passwd`	creates the file `passwd`
`xyz0815`	sets the password, finish the entry with `Ctrl+D`
`/save >sync`	saves the password in EEPROM
`/save >reset`	software reset
`Bye. Enter password:`	prompts to enter the password when starting a new session over Telnet or RS-232

Entry of an incorrect password will be answered with `Access denied`.

6 Error Handling

6.1 CAN-Ethernet Gateway Error Signals

The following error states are indicated:

- Problem: Establishing a connection over BTP interface not possible

  Possible causes:

  No connection to configured server possible or connection denied

  Indication on Gateway:

  Ethernet error LED illuminates

- Problem: CAN RxBuffer overflow, CAN message loss

  Possible causes:

  CAN bus load too high
  receive buffer too small *(refer to section 4.2.4 for settings)*

  Indication on Gateway:

  CAN Error LED illuminates briefly

- Problem: CAN TxBuffer overflow

  Possible causes:

  CAN messages are being incorporated too quickly into the CAN send buffer (for example due to high bus load and low priority of the CAN ID to be sent)

  General: incorporation of error messages possible, if CAN internal high-priority buffer has not overflowed
  Incorporation of the original CAN message is attempted again.

  Indication on Gateway:

  CAN Error LED illuminate briefly

- Problem: BTP TxBuffer overflow
  Possible causes:
  no CAN to UDP buffer available -> message loss

  Indication on Gateway:
  Ethernet error LED illuminates

- Problem: BTP RxBuffer overflow
  Possible causes:
  no CAN to UDP buffer available -> message loss

  Indication on Gateway:
  Ethernet error LED illuminates

- Problem: Error during receipt or transmission via BTP
  Possible causes:
  e.g. if the recipient does not have a free buffer or in the event of a timeout
  General: CAN to TCP/IP send buffer is discarded -> message loss

  Indication on Gateway:
  Ethernet error LED illuminates

- CAN bus-off error
  Possible causes:
  incorrect CAN bus cabling,
  wrong CAN bitrate,
  hardware error
  Gateway: CAN error LED blinks with a 50:50 on/off ratio until the transmission or receipt of a CAN message was successful

- CAN- ACK error

  Possible causes:

  No other CAN device connected to the CAN bus
  Incorrect CAN bus cabling,
  hardware error,
  terminating resistors are missing or wrong value (120 Ohm at both ends are required)

  Gateway: CAN error LED blinks with a 25:75 on/off ratio until the transmission or receipt of a CAN message was successful

Ethernet Error LED	CAN Error LED	Error Description
illuminates		No connection established via BTP interface or connection interrupted
illuminates		BTP send buffer overflow (message loss)
illuminates		BTP receive buffer overflow (message loss)
illuminates		Error while sending or receiving over BTP (message loss)
	short flashes	Receive buffer overflow (message loss)
	blinks with 50:50 on/off ratio	CAN bus-off error detected
	blinks with 25:75 on/off ratio	CAN acknowledgement error detected
	short flashes	Send buffer overflow

Table 7: Error Display Overview

6.2 Error Indication under Windows

All of the error states listed above are indicated by pop-up windows.

6.3 Error Messages over CAN

Error messages can be sent according to the CANopen standard (emergency messages) in order to be able to evaluate the state of the Gateway from the CAN network.

The transmission of error messages with the identifier to be used in the CAN-Ethernet Gateway, can be activated with the option `canid:<id>` when the CAN interface is created.

In the standard configuration transmission of an error message with identifier 0xFE is enabled.

The format of the error message is specified as follows:

Byte	0	1	2	3	4	5	6	7
Contents	Emergency Code		Error Register	Interface Number	Error Code		reserved	reserved

Table 8: Structure of a CAN Emergency Message

The emergency code can assume the following values:

0x1000: if a new common error has occurred
0x8140: node comes out of CAN bus-off state
0x0000: the device works properly, no errors

The error register is 0x80 if there are still errors present and 0x00 if all errors have been corrected. In the next field the number of the interface where the error occurred is displayed, followed by a bit mask, which shows what errors are present. For the 2 byte long emergency and error codes (byte 0/1 and 4/5) the MSB portion is sent last, e.g. 00 10 for 0x1000.

The following bits are specified for the error code:

0x0000	no errors
0x0001	buffer overflow when receiving
0x0002	buffer overflow when transmitting
0x0004	buffer overflow in the CAN controller
0x0008	CAN acknowledge (ACK) error
0x0010	CAN warning limit reached
0x0020	CAN passive mode reached
0x0040	device is in CAN bus off state
0x0080	error during message transmission
0x0100	error during message receipt
0x0400	the interface connection has been terminated
0x2000	generic CAN controller error, internal hardware error of the Infineon TwinCAN (bit stuffing error, form error, CRC error)

These device error states can also be queried via Telnet or RS-232. The state (`state`) is located in the CAN interface. Along with the possible error states, the following additional bits provide information about the existing BTP connections.

0x0200	The interface is successfully connected
0x0800	Signalization of the data transfer

Example:

```
/if > can0↵

Usage: can0 [bus:<num>] [baud:<num>] [userbaud:<hex>]
            [canid:<hex>] [on] [off]

Current settings: bus:0 baud:0 (1MBit/s) userbaud:0000
                  canid:0xF5 on state:0x850
```

`state = 0x850` shows CAN bus-off and CAN warning limit as well as data transfer via BTP

7 Software Support

7.1 Interfacing the CAN-Ethernet Gateway to a PC

A WIN32 DLL, which allows for a number of export functions is available for connecting the CAN-Ethernet Gateway to a PC. With this DLL it is possible to develop your own applications under Windows. The CAN-Ethernet Gateway can be accessed directly from the PC over the driver DLL via Ethernet.

7.2 Driver Installation under Windows

The installation of the driver DLL for Windows is required. The corresponding setup program can be found on the included CD in the SO-1027 folder.

Start the setup program on the CD and follow the instructions that appear on the screen. The program will be installed in the following folder:

C:\Program Files\SYSTEC-electronic\CAN-Ethernet-Gateway_Utility_Disk

The installation path can be changed by the user as desired.

Note:
Be sure that you have administrator rights for installation under Windows 2000, XP, Vista and Windows 7!

During installation the driver-DLL (EthCAN.Dll) will be copied to the corresponding Windows system folder (depending on your operating system). The setup program will also create the following file structure in the installation folder, based on the default installation path:

Subfolder	Contents
Demo.Prj	"C" demo in source for MSVC 5.0 and 6.0
Doku	Systems Manual CAN-Ethernet-Gateway
Include	"C" header file for EthCan.Dll
Lib	EthCan.Lib and EthCan.Dll

Table 9: File Structure of the CAN-Ethernet Gateway Utility Disk

The folder ***"LIB"*** contains the library and the corresponding DLL. In the folder ***"Include"*** you will find the header file for the ***"EthCAN.Dll",*** which contains all prototypes for the public functions of the DLL as well as all data structures and data types used. This header file needs to be included in the development project along with the DLL for writing user applications. The folder ***"Doku"*** contains the Systems Manual for the CAN-Ethernet Gateway as a PDF file. The folder ***"Demo.Prj"*** contains a demo project in the form of a Visual Studio project. This project contains a "C" source file as well as a corresponding header file, which shows the use of the DLL functions in the form of a demo program.

7.3 Dynamic Linked Library *EthCan.Dll*

The Dynamic Linked Library (EthCAN.Dll) is a function library for application programs. It serves as an interface between the Windows socket and an application program. It manages the connected CAN-Ethernet Gateway devices and converts the CAN messages in IP packets and vice versa.

In order to include the DLL to a project, the EthCan.Lib must also be included. The DLL is then loaded automatically when the application program is started. If the LIB is not linked to the project, then the DLL has to be loaded with the Windows function *LoadLibrary()* and the library function has to be added with the function *GetProcAdress()*. The STDCALL directive of the DLL's call function ensures a standardized call interface for the user. This guarantees that someone familiar with another programming language (Pascal, ...) can also use these functions.

7.3.1 The Concept of the EthCan.Dll

With the ***EthCan.Dll*** up to 5 CAN-Ethernet Gateways can be addressed simultaneously within an application. Furthermore, it is also possible to access 5 CAN-Ethernet Gateways from a remote application as long as they have the same remote address (as shown in application 1). This is because multiple interfaces can be connected to the CAN-Ethernet Gateway at once, so that connections from multiple applications are possible. However, it is **not** possible to establish multiple connections to one and the same CAN-Ethernet Gateway from a single application.

When the DLL is used, two software states are generated for each CAN-Ethernet Gateway. After the application program has been started and the DLL has been loaded, the software will be in the state DLL_INIT, whereby all necessary resources for the DLL are created. If the library function ***EthCanInitHardware()*** is called, then the software will change to the HW_INIT state. All resources that are necessary for communication with the CAN-Ethernet Gateway are

provided here. With the library function ***EthCanDeinitHardware()*** it is possible to change from the HW_INIT state back to the state DLL_INIT. Only then it is possible to end the application program.

State	Available Functions
DLL_INIT	*EthCanGetVersion ()* *EthCanInitHardware ()*
HW_INIT	*EthCanGetVersion ()* *EthCanGetStatus ()* *EthCanDeinitHardware ()* *EthCanReadCanMsg()* *EthCanWriteCanMsg()* *EthCanResetCan()* *EthCanGetConnectionState()*

Table 10: Available Functions within the Software States

If multiple CAN-Ethernet Gateways are used in an application then these states apply to each CAN-Ethernet Gateway. While the first CAN-Ethernet Gateway is already in the DLL_INIT state, the second Gateway can be in the HW_INIT state.

7.3.2 *EthCan.Dll* Function Interface

This section describes the interface functions of the CAN-Ethernet DLL in terms of their use and return values. The use of the functions is illustrated with code examples. All function parameters are selected so that the DLL can be used with programming languages such as Pascal or Visual Basic.

7.3.2.1 EthCanGetVersion

Syntax:

DWORD STDCALL EthCanGetVersion (void);

Usage DLL_INIT, HW_INIT

Description:
The function returns the software version number of the ***EthCan.Dll***.

Parameters: none

Return value:
The return value is the software version number in *DWORD* format. It is structured as follows:

bit 0 - 7:	most significant portion of version number in binary format
bit 8 - 15:	least significant portion of version number in binary format
bit 16 - 31:	release version number in binary format

Application Example:

```
DWORD dwVersion;
char szVersion[16];
...
// Get version number
dwVersion = EthCanGetVersion ();

// convert to a string
wsprintf( szVersion, „V%d.%02d.r%d", (dwVersion&0xff),
         (dwVersion&0xff00)>>8, dwVersion>>16);
```

7.3.2.2 EthCanInitHardware

Syntax:

DWORD STDCALL EthCanInitHardware(
tEthCanHandle pEthCanHandle_p,*
tEthCanHwParam pEthCanHwParam_p,*
tEthCanCbConnectFct fpEthCanCbConnectFct_p
void pArg_p);*

Usage: DLL_INIT

Description:

This function initializes all necessary data structures and establishes a connection to the addressed CAN-Ethernet Gateway. The parameters required for this such as the IP address, port number etc. are transmitted in the form of an address to a hardware parameter structure (parameter #2).
This function differentiates between two call modes:

1. The function operates in the so-called ***"Blocked Mode"***, if a NULL pointer is given as the pointer for the callback function (parameter #3). The function will only return if a successful connection to the CAN-Ethernet Gateway was established or if an error has occurred, e.g. in the form of a timeout.
2. The function operates in the so-called ***"Non-blocked Mode"*** if a valid address has been sent to a callback function. The function initializes all necessary data structures and initializes the connection without having to wait for successful completion. The status of the connection is determined over the callback function, which must be created in the application layer. It reports the current connection status and is always called from the DLL if the connection status changes. It is therefore possible to react accordingly to any breaks in the connection within the application.

Parameter:

***pEthCanHandle_p**:* Address of the instance handle of the CAN-Ethernet Gateway

This variable is a pointer of type ***tEthCanHandle***. Upon successful initialization, this address includes a valid hardware handle that also serves as an instance handle. This instance handle should be secured and identified as a return parameter when all other functions of this instance are called.

***pEthCanHwParam_p**:* Address to the hardware parameter structure

This variable is an address to a ***tEthCanHwParam*** hardware parameter structure. It has the following structure:

```
typedef struct
{
  DWORD          m_dwIpAddress;         //IP address
  WORD           m_wPort;               //Port number
  tUsedProtocol  m_UsedProtocol;        //Protocol (UDP or TCP)
  DWORD          m_dwReconnectTimeout;  //Timeout for "Reconnect"
  DWORD          m_dwConnectTimeout;    //Timeout for "Connect"
  DWORD          m_dwDisconnectTimeout; //Timeout for "Disconnect"

}tEthCanHwParam;
```

Figure 14: Hardware Parameter Structure Overview

This structure has to be filled out accordingly prior to a return to the function. The IP address and port number correspond to the remote address (IP address of the CAN-Ethernet Gateway) to which a connection has to be established. The following format must be used:

```
#define IP_ADDR_DEFAULT ((192<<0)+(168<<8)+(10<<16)+(111<<24))
#define IP_PORT_DEFAULT (8234)
```

UDP and TCP are available for the transfer protocol to be used.

```
typedef enum
{
    kUseTCP = 0x00, // TCP Protocol
    kUseUDP = 0x01  // UDP Protocol

}tUsedProtocol;
```

Figure 15: CAN-Ethernet-Gateway Transfer Protocols

The member variable ***m_dwReconnectTime*** describes the duration following a break in the connection before a new connection is established automatically. If this time is 0, then no new connection is established. The member variable ***m_dwConnectTimeout*** is only relevant if the Init function is called in ***„Blocked Mode“***. It describes the time, after which the Init function returns if a successful connection could not be established. If this time is **0**, then a default timeout of **5s** is set.
The same applies for the member variable ***m_dwDisconnectTimeout***, which sets the timeout for the Deinit function, after which an active connection must be established.

fpEthCanCbConnectFct_p*:*

Address of the Callback function for the connection status of the CAN-Ethernet Gateway.

This value can be NULL when it is passed to the Init function, which means that there is no Callback function.
If a Callback function is to be used to react to changes in the connection status, the following declaration has to be made:

void PUBLIC EthCanConnectControlFct(
tEthCanHandle EthCanHandle_p,
DWORD dwConnectionState_p);

It is therefore possible for a single Callback function to be assigned for the initialization of multiple instances. The Instance handle (***EthCanHadle_p***), which is returned to the Callback function, is used

to determine at which instance the connection status changed. However, there is the possibility of assigning a separate Callback function for each initialized instance.

Note:
If a separate Callback function is created for each instance, it is important to be sure that different function names are assigned to prevent Compiler and Linker errors!

The parameter ***dwConnectionState_p*** describes the current connection status and can adapt the following values, described by the type ***tConnectionState***:

```
typedef enum
{
  kConnecting  = 0,  // Connection in process
  kEstablished = 1,  // Connection established
  kClosing     = 2,  // Disconnecting
  kClosed      = 3,  // Disconnected
}tConnectionState;
```

Figure 16: CAN-Ethernet Gateway Connection Status

pArg_p: Address of argument for the Callback Function

At this location an argument can be given, which will be sent back from the DLL upon a call of the Callback function.
This could be the address of an instance of the CAN-Ethernet Gateway, if multiple connections are managed within an instance table are to be addressed. If only one Callback function was declared for multiple instances, then the argument pointer and its access to the elements in the instance table can be used to determine which instance the Callback function was called from. Since the return parameter ***pArg_p*** is a ***void**** type, then parameters of any kind can be returned at this location, depending on the application.

Return Values: *(see Section 7.3.3)*

ETHCAN_SUCCESSFULL
ETHCAN_ERR_RESOURCE
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_HWINUSE
ETHCAN_ERR_HWCONNECT_FAILD
ETHCAN_ERR_MAXMODULES
ETHCAN_ERR_SAL
ETHCAN_ERR_IFBTP

Application Example:

```
#define IP_ADDR_DEFAULT ((192 << 0)+(168 << 8)+ (10 << 16)+(111 << 24))
#define IP_PORT_DEFAULT (8234)

DWORD          dwRetcode;
tEthCanHandle  EthCanHandle;
tEthCanHwParam EthCanHwParam;

EthCanHwParam.m_dwReconnectTimeout  = 120000;//120s
EthCanHwParam.m_dwIpAddress         = IP_ADDR_DEFAULT;
EthCanHwParam.m_wPort               = IP_PORT_DEFAULT;
EthCanHwParam.m_dwConnectTimeout    = 5000;//5s
EthCanHwParam.m_dwDisConnectTimeout = 5000;//5s
```

without Callback Function:

```
// initialize a CAN-Ethernet Gateway without a Callback function
dwRetcode = EthCanInitHardware (&EthCanHandle,&EthCanHwParam,NULL,NULL);
```

with Callback Function:

```
void PUBLIC EthCanConnectControlFct (tEthCanHandle EthCanHandle_p,
                                     DWORD dwConnectionState_p,
                                     void* pArg_p)
{
    switch(dwConnectionState_p)
    {
        //Connection in process
        case kConnecting:……
            break;

        //Connection Established
        case kEstablished:……
            break;

        //Disconnecting
        case kClosing:……
            break;

        //Disconnected
```

```
        case kClosed:……
            break;
    }
}

//initializes a CAN-Ethernet Gateway with Callback function
dwRetcode = EthCanInitHardware (&EthCanHandle, &EthCanHwParam,
                                EthCanConnectControlFct,NULL);
```

7.3.2.3 EthCanDeinitHardware

Syntax:

DWORD STDCALL EthCanDeinitHardware (
tEthCanHandle EthCanHandle_p);

Usage: HW_INIT

Description:

This function is the complement to the initialization function ***EthCanInitHardware()***. This means that this function operates in „***Blocked Mode***“ as well as in „***Non-blocked Mode***“. The call mode is determined by the call mode of the initialization function, so that there is always an equivalent.
The task of this function is to break an existing connection in a controlled manner and then de-initialize the data structures of the instance. The return parameter ***EthCanHandle_p*** describes the instance that is to be disconnected.

1. In „***Blocked Mode***“ the connection process gets started and waits for the completion of the connection process. The function returns when the connection has been disconnected or if an error or timeout occurs.

2. In „***Non-blocked Mode***“ the disconnection process is started without waiting for completion. The function returns immediately. If the connection status is changed the Callback function is called from the DLL, which returns the current connection status.

Parameter:

***EthCanHandle_p**:* Instance Handle of the CAN-Ethernet Gateway

Return Values: *(see Section 7.3.3)*

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_HWNOINIT
ETHCAN_ERR_HWDISCONNECT_FAILD
ETHCAN_ERR_SAL
ETHCAN_ERR_IFBTP
ETHCAN_ERR_RESOURCE

Note:
The function ***EthCanDeinitHardware()*** needs to be called as many times, as the function ***EthCanInitHardware()*** was called previously and returned without error. If the functions were called in ***„Non-blocked Mode“*** it is important to be sure in any case that before ending the application the disconnect has been signaled via the Callback function.

Application Example:

Both application examples below show the use of the function with a blocking and non-blocking call.

Blocking call

```
#define IP_ADDR ((192 << 0)+(168 << 8)+ (10 << 16)+(111 << 24))
#define IP_PORT (8234)

void main (void)
{
  DWORD dwRetcode;
  tEthCanHandle  EthCanHandle;
  tEthCanHwParam EthCanHwParam;

  EthCanHwParam.m_IpAdress = IP_ADDR;
  EthCanHwParam.m_wPort = IP_PORT;
  EthCanHwParam.m_UsedProtocol = kUseTCP;

  dwRetcode = EthCanInitHardware(&EthCanHandle,&EthCanHwParam,NULL,NULL);
  if(dwRetcode == ETHCAN_SUCCESSFUL)
  {
       printf("\n*** Successfully initialized! ***\n");
  }
  else
  {
    goto Exit;
  }
              .
              .
  dwRetcode = EthCanDeinitHardware(EthCanHandle);
  if(dwRetcode == ETHCAN_SUCCESSFUL)
  {
       printf("\n*** Successfully closed! ***\n",
  }

Exit:
  return(dwRetcode);
}
```

Non-blocking call

```
//Callback Function for the Connection Status
void PUBLIC EthCanConnectControlFct (tEthCanHandle EthCanHandle_p,
                                     DWORD dwConnectionState_p,
                                     void* pArg_p)
{
       switch(dwConnectionState_p)
       {
              case kEstablished:
                     EthCanInst_g[EthCanHandle_p].fConnected = TRUE;
                     break;

              case kConnecting:
              case kClosing:
              case kClosed:
                     EthCanInst_g[EthCanHandle_p].m_fConnected = FALSE;
                     break;
       }
}


void main (void)
{
  DWORD dwRetcode;
  tEthCanHandle EthCanHandle;

  //initialize a CAN-Ethernet Gateway with callback function
  dwRetcode = EthCanInitHardware (&EthCanHandle, &EthCanHwParam,
                                  EthCanConnectControlFct,NULL);

  if(dwRetcode == ETHCAN_SUCCESSFUL)
  {
     printf("\n*** Successfully initialised! ***\n",
  }
                           .
                           .
                           .
                           .
  dwRetcode = EthCanDeinitHardware(EthCanHandle);
  if(dwRetcode == ETHCAN_SUCCESSFUL)
  {
     printf("\n*** Successfully closed! ***\n");
  }

  //Wait for the disconnect, signalized by the Callback function
  do
  {
    Sleep(10);

  }while(EthCanInst_g[EthCanHandle].m_fConnected);

  //Exit application
  return(dwRetcode);
}
```

7.3.2.4 EthCanReadCanMsg

Syntax:

BYTE STDCALL EthCanReadCanMsg(
tEthCanHandle EthCanHandle_p,
tCANMsg pRcvCanMsg_p*
tCANTimestamp pRcvTime_p);*

Usability: HW_INIT

Description:

This function reads a CAN message from the DLL's message buffer. The CAN message is then deleted from the message buffer. If there is no CAN message in the receive buffer, the function will return with value ***ETHCAN_CANERR_QRCVEMPTY***.

Parameter:

EthCanHandle_p: Instance Handle of the CAN-Ethernet Gateway

pRcvCanMsg_p: Address to a CAN message structure
This address must not be NULL!

The structure of the CAN message is as follows:

```
typedef struct
{
  DWORD  m_dwID;      // CAN-Identifier
  BYTE   m_bMsgType;  // CAN-Frame-Format
  BYTE   m_bLen;;     // CAN-data length
  BYTE   m_bData[8];  // CAN-Data (max. 8 Bytes)

}tCANMsg;
```

Figure 17: CAN Message Structure

When formatting the CAN message, there is differentiation between two types. CAN messages with 11-bit identifiers (Standard CAN Frame) and CAN messages with 29-bit identifiers (Extended CAN Frame) are supported. This also applies for so-called remote frames (RTR frames) in Standard or Extended-CAN frame format. The CAN message's message format corresponds to a bit combination that is defined as follows:

```
//Standard CAN-Frame
#define ETHCAN_MSGTYPE_STANDARD  0x00

//Remote Frame (11 Bit and 29 Bit CAN-ID)
#define ETHCAN_MSGTYPE_RTR       0x01

//Extended CAN Frame 2.0 B Frame (29 Bit CAN-ID)
#define ETHCAN_MSGTYPE_EXTENDED  0x02
```

For a RTR frame in extended format, the values have to be combined accordingly and entered as a message format into the CAN message structure.

pRcvTime_p: Address to a Timestamp structure of a CAN message. This address must not be NULL

The Timestamp structure is defined as follows:

```
typedef struct
{
  DWORD  m_dwMilliSec;          //Milliseconds
  WORD   m_wMilliSec_Overflow;  //Milliseconds-overflow
  WORD   m_wMicroSec;           //Microseconds

}tCANTimestamp;
```

Figure 18: The CAN timestamp structure

The member variable ***m_dwMilliSec*** contains the number of milliseconds that have passed since the system start of the CAN-Ethernet Gateway hardware. The time between two CAN messages is determined by the difference between the two values.

Note:
The Member variables ***m_wMilliSec_Overflow*** and ***m_wMicroSec*** of the Timestamp structure are not used at this time and always contain the value ***0***.

Return Value: (*see Section 7.3.3 and 7.3.4*)

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_HWNOINIT
ETHCAN_ERR_HWNOTCONNECTED
ETHCAN_CANERR_QRCVEMPTY
ETHCAN_CANERR_QOVERRUN

Application Example:

```
tEthCanHandle EthCanHandle;
tCANMsg       RecvCanMsg;
tCANTimestamp RecvTime;
DWORD         dwRetcode;

//Read CAN-message
dwRetcode = EthCanReadCanMsg(EthCanHandle,&RecvCanMsg,&RecvTime);
if(dwRetcode == ETHCAN_SUCCESSFUL)
{
      //CAN-message received
}
else
if(dwRetcode & ETHCAN_CANERR_ QRCVEMPTY)
{
      //no CAN-message in the message buffer
}
else
{
      //error during receipt of the CAN-message
}
```

7.3.2.5 EthCanWriteCanMsg

Syntax:

DWORD STDCALL EthCanWriteCanMsg(
tEthCanHandle EthCanHandle_p,
tCANMsg pSendCanMsg_p,*
tCANTimestamp pSendTime_p);*

Usability: HW_INIT

Description:

This function writes a CAN-message to the send buffer located in the ***EthCan.Dll***. If the CAN-message could not be stored in the send buffer (e.g. buffer overflow), the function returns with the error code ***ETHCAN_CANERR_QXMTFULL***, which means that a buffer overflow occurred.

Parameter:

EthCanHandle_p: Instance handle of the CAN-Ethernet Gateway

pSendCanMsg_p: Address to a CAN-message structure. This address must not be NULL!

pSendTime_p: Address to a Timestamp structure This address must not be NULL!

The parameter ***pSendCanMsg_p*** corresponds to an address pointing to the structure of a CAN-message, as explained for the function ***EthCanReadCanMsg()*** (*see Section 7.3.2.4*). The message format should be set (*see Section 7.3.2.4*) corresponding to the CAN-message (29-Bit, 11-Bit, RTR) to be sent.
The parameter ***pSendTime_p*** is an address pointing to a Timestamp structure. For this function, this parameter has no meaning, however

the returned address must not be NULL. The structure elements are to be initialized with ***0***.

Return Value: (*see Section 7.3.3 and 7.3.4*)

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_HWNOINIT
ETHCAN_ERR_HWNOTCONNECTED
ETHCAN_CANERR_QXMTFULL

Application Example:

```
tEthCanHandle EthCanHandle;
tCANMsg       CanMsg;
tCANTimestamp SendTime;
DWORD         dwRetcode;

//Initialization of a Standard-RTR-Frame
CanMsg.m_dwId     = 0x180; //CAN-ID
CanMsg.m_bMsgType = ETHCAN_MSGTYPE_STANDARD & ETHCAN_MSGTYPE_RTR;
CanMsg.m_bLen     = 8;//8 Byte requested Data length

SendTime.m_dwMillis = 0;

//Send CAN-message
dwRetcode = EthCanWriteCanMsg(EthCanHandle,&CanMsg,&SendTime);
if(dwRetcode == ETHCAN_SUCCESSFUL)
{
      //CAN-message sent
}
else
if(dwRetcode & ETHCAN_CANERR_ QXMTFULL)
{
      //Overflow of the send buffer
}
else
{
      //Error when sending the CAN-message
}
```

7.3.2.6 EthCanGetStatus

Syntax:

DWORD STDCALL EthCanGetStatus(
tEthCanHandle EthCanHandle_p,
tStatus pStatus_p);*

Usability: HW_INIT

Description:

The function returns the error status of the CAN-driver as well as the connection status of the CAN-Ethernet Gateway. If a CAN-error occurs on the CAN-Ethernet Gateway (e.g. send or receive buffer overflow), then this status will be returned via Ethernet and can be called up with the help of this function. In addition to the CAN-status the current connection status of the Ethernet connection between the PC and the CAN-Ethernet Gateway is also returned. This function should be called in specific time intervals in order to be able to react to potential CAN-errors.

Parameter:

***EthCanHandle_p**:* Instance Handle of the CAN-Ethernet Gateway

***pStatus_p**:* Address to a status structure. This address must not be NULL!

This status structure is defined as follows:

```
typedef struct
{
  WORD  m_wCanStatus;         // current CAN state
  WORD  m_wConnectionStatus;  // current connection state
}tStatus;
```

Figure 19: CAN-Status Structure

Return Values: *(see Section 7.3.3)*

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_HWNOINIT
ETHCAN_ERR_HWNOTCONNECTED

Application Example:

```
tEthCanHandle EthCanHandle;
tStatus       Status;
DWORD         dwRetcode;

//read CAN-Status
dwRetcode = EthCanGetStatus(EthCanHandle,&Status);
if(dwRetcode == ETHCAN_SUCCESSFUL)
{
      if(Status.m_wCanStatus & ETHCAN_CANERR_OVERRUN)
      {
            //Overrun error occurred
      }
}
else
{
      //Error while reading the CAN status
}
```

7.3.2.7 EthCanGetConnectionState

Syntax:

DWORD PUBLIC EthCanGetConnectionState(
tEthCanHandle EthCanHandle_p,
tConnectionState pState_p);*

Usability: HW_INIT

Parameter:

EthCanHandle_p: Instance handle of the CAN-Ethernet Gateway

pState_p: Address to the connection status variable. This address cannot be NULL!

Description:
This function returns the current connection status of the CAN-Ethernet Gateway. If there was no callback function declared upon initialization of the Gateway, then this function can be used to call up the connection status with a polling procedure. This function should be used if the initialization and de-initialization routines have been called in „***Blocked Mode***“.
The parameter ***pState_p*** returns the current connection status after the function has been called and can take on the values as previously described in *Figure 16*.

Return Values: *(see Section 7.3.3)*

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_HWNOINIT

Application Example:

```
tEthCanHandle    EthCanHandle;
tConnectionState ConnectionState;
DWORD            dwRetcode;

//Read Connection Status
dwRetcode = EthCanGetStatus(EthCanHandle,&ConnectionState);
if(dwRetcode == ETHCAN_SUCCESSFUL)
{
      if(ConnectionState == kConnecting)
      {
            //Code to be executed
      }
      if(ConnectionState == kEstablished)
      {
            //Code to be executed
      }
                  .
                  .
                  .
}
else
{
      //Error while reading the connection status
}
```

7.3.2.8 EthCanResetCan

Syntax:

DWORD PUBLIC EthCanResetCan(
tEthCanHandle,
dwResetCode_p)

Usability: HW_INIT

Parameter:

EthCanHandle_p: Instance handle of the CAN-Ethernet Gateway

dwResetCode_p: Reset-Code for CAN

Description:

This function is used for performing a specific reset of the CAN communication on the CAN-Ethernet Gateway and the DLL when CAN errors occur as the result of a buffer overflow or interference on the CAN-Bus. The parameter ***dwResetCode_p*** is used to determine whether the send and receive buffer in the DLL as well as the send and receive buffer of the CAN-Ethernet Gateway and its CAN interface (CAN-controller) should be reset.
The following parameter values can be given for the reset code:

```
//Erase the send buffer for CAN messages in the DLL
#define RESET_TRANSMIT_QUEUE                0x00

//Erase the receive buffer for CAN messages in the DLL
#define RESET_RECEIVE_QUEUE                 0x01

//Erase the send and receive buffer for CAN messages in the DLL
#define RESET_ALL_QUEUES                    0x02

//Erase the send and receive buffer for CAN messages on
```

```
//the Gateway and reset the CAN controller
#define RESET_CAN_CONTROLLER                0x04
```

These constants can be combined bit by bit so that, depending on the application, it is possible to reset specific or all CAN components.

Note:
No CAN messages can be sent or received while the CAN interface is being reset or if the buffer is being erased!

Return Values: (*see Section 7.3.3*)

ETHCAN_SUCCESSFUL
ETHCAN_ERR_ILLHANDLE
ETHCAN_ERR_ILLPARAM
ETHCAN_ERR_HWNOINIT
ETHCAN_ERR_HWNOTCONNECTED

Application Example:

```
tEthCanHandle EthCanHandle;
DWORD         dwResetCode;
DWORD         dwRetcode;

//Reset all buffers and reset the CAN interface
//of the CAN-Ethernet Gateway
dwResetCode = (RESET_ALL_QUEUES & RESET_CAN_CONTROLLER);

//Reset the CAN interface
dwRetcode = EthCanResetCan(EthCanHandle,dwResetCode);
if(dwRetcode == ETHCAN_SUCCESSFUL)
{
      //CAN interface was reset successfully
}
else
{
      //An error occurred while resetting the CAN interface
}
```

7.3.3 Error Code Description

The functions of the EthCan.Dll return an error code in the form of a DWORD. Every return value corresponds to an error. All error codes and their numerical values are listed in the following table.

Error Code	Numerical Value
ETHCAN_SUCCESSFUL	0x0
ETHCAN_ERR_ILLPARAM	0x1
ETHCAN_ERR_ILLPARAMVAL	0x2
ETHCAN_ERR_ILLHANDLE	0x3
ETHCAN_ERR_HWNOINIT	0x4
ETHCAN_ERR_HWINUSE	0x5
ETHCAN_ERR_HWNOTCONNECTED	0x6
ETHCAN_ERR_HWCONNECT_FAILED	0x7
ETHCAN_ERR_HWDISCONNECT_FAILED	0x8
ETHCAN_ERR_MAXMODULES	0x9
ETHCAN_ERR_SAL	0xA
ETHCAN_ERR_IFBTP	0xB
ETHCAN_ERR_RESOURCE	0xC

Table 11: EthCan.Dll Interface Function Error Codes

ETHCAN_SUCCESSFUL

The function was executed without errors.

ETHCAN_ERR_ILLPARAM

An illegal parameter was passed to the function. The most probable cause for this error is passing of a NULL pointer.

ETHCAN_ERR_ILLPARAMVAL

An invalid parameter value was passed to the function.

ETHCAN_ERR_ILLHANDLE

An invalid instance handle was passed to the function. One possible cause for this error is the return of an instance handle with an invalid instance number, 0 for example. In addition the driver only supports max. 5 instances so that the return of an instance handle greater than 5 will also result in this error.

ETHCAN_ERR_HWNOINIT

The function was called with an instance handle, for which the corresponding hardware initialization function was not yet called. Therefore the function ***EthCanInitHardware()*** has to be called, which will return a valid instance handle upon successful completion.

ETHCAN_ERR_HWINUSE

The function ***EthCanInitHardware()*** was called with an instance handle, for which the initialization routine was already called and returned successfully. In order to perform a new initialization with modified parameter values, the de-initialization function ***EthCanDeinitHardware()*** has to be called before a new initialization can occur.

ETHCAN_ERR_HWNOTCONNECTED

The function was called, but at the time of the call there was no connection of the CAN-Ethernet Gateway to the PC. A possible cause may be a break in the connection as a result of the loss of the physical link (Ethernet cable was disconnected).
If a callback function was assigned during initialization, the application is able to react to this event. For example, the send and receive functions for a CAN message should only be polled if the connection status is ***kEstablished***.

If no callback function is defined then the function ***EthCanGetConnectionState()*** should be used to poll the connection status. Following a successful "***Reconnect***" the function can be called again.

ETHCAN_ERR_HWCONNECT_FAILED

This error code is returned by the function ***EthCanInitHardware()*** only if it was called in blocking mode, i.e. without the declaration of a callback function. This error code indicates that no connection to the remote address could be established within the timeout time, which was given to the parameter structure upon initialization,. The default timeout is set to ***5s*** in the header file ***EthCan32.h***. If upon initialization an application specific timeout time was given, then it is up to the user to make sure the timeout is sufficient for a connection to be established successfully (depending on the network topology).
If this error code is returned in order to check whether the assigned parameters, such as IP address and port number, are correct and that the remote address really is accessible over the Ethernet connection.

ETHCAN_ERR_HWDISCONNECT_FAILED

This error code is returned by the function ***EthCanDeinitHardware()*** only if it was called in blocking mode. The cause of this is that the disconnection to the remote address could not be closed within the timeout time assigned to the parameter structure upon initialization. A default timeout of ***5s*** is defined in the header file ***EthCan32.h***. If upon initialization an application specific timeout time was given, then it is necessary to check whether the timeout is sufficient for a connection to be disconnected successfully.

ETHCAN_ERR_MAXMODULES

The maximum number of CAN-Ethernet Gateways that can be supported by the DLL has been reached. No other instances can be initialized. De-initialize any instances that are no longer required and then call the init function again.

ETHCAN_ERR_SAL

An error occurred during initialization or de-initialization of the SAL layer (Stack Abstraction Layer) for TCP or UDP. Possible causes for the error could be:

- The creation or closure of the Windows socket interface may not have occurred due to missing resources or an unsupported version of the Windows socket interface.
- The call of the WIN32 functions for the Windows socket interface, such as Connect(), Bind(), Accept(), Send() and Recv() returned an error due to invalid parameters.

ETHCAN_ERR_IFBTP

An error occurred during the initialization or de-initialization of the BTP interface (Block Transfer Protocol) for UDP or TCP.

ETHCAN_ERR_RESOURCE

A resource could not be created or is not available. Understood as resources are memory requirements, handles or threads, which are managed by Windows.

7.3.4 CAN Error Code Description

The CAN Error Code which is returned by the functions ***EthCanReadCanMsg***(), ***EthCanWriteCanMsg***() and ***EthCanGetStatus***(), corresponds to a bit combination from the error codes listed in the following table. The error code may indicate multiple errors.

CAN Error Code	Numerical Value
ETHCAN_CANERR_OK	0x0000
ETHCAN_CANERR_XMTFULL	0x0001
ETHCAN_CANERR_OVERRUN	0x0002
ETHCAN_CANERR_BUSLIGHT	0x0004
ETHCAN_CANERR_BUSHEAVY	0x0008
ETHCAN_CANERR_BUSOFF	0x0010
ETHCAN_CANERR_QRCVEMPTY	0x0020
ETHCAN_CANERR_QOVERRUN	0x0040
ETHCAN_CANERR_QXMTFULL	0x0080
ETHCAN_CANERR_REGTEST	0x0100
ETHCAN_CANERR_MEMTEST	0x0200

Table 12: CAN Error Codes

ETHCAN_CANERR_OK

No CAN Error has occurred

ETHCAN_CANERR_XMTFULL

The send buffer in the CAN Controller of the CAN-Ethernet Gateway has reached the maximum number of CAN messages.

ETHCAN_CANERR_OVERRUN

The receive buffer in the CAN controller of the CAN Ethernet Gateway has reached the maximum number of CAN messages.

ETHCAN_CANERR_BUSLIGHT

The error counter in the CAN controller has reached Warning Limit 1.

ETHCAN_CANERR_BUSHEAVY

The error counter in the CAN controller has reached Warning Limit 2.

ETHCAN_CANERR_BUSOFF

Because of the error counter the CAN controller has switched to the BUS_OFF state to prevent further interference on the CAN Bus.

ETHCAN_CANERR_QRCVEMPTY

The receiving queue for CAN messages within the DLL does not contain any CAN messages, which means that all CAN messages have been read or no new CAN messages have been received.

ETHCAN_CANERR_QOVERRUN

Overflow of the receive queue for CAN messages. It is possible that some CAN messages will have been lost.

ETHCAN_CANERR_QXMTFULL

Overflow of the send queue for CAN messages in the DLL. It is possible that some CAN messages will have been lost.

ETHCAN_CANERR_REGTEST

The register test of the SJA1000 failed. Refer to the manual for the CAN Controller SJA 1000.

ETHCAN_CANERR_MEMTEST

The memory test of the SJA1000 failed. Refer to the manual for the CAN Controller SJA 1000.

7.3.5 Using the DLL Functions

7.3.5.1 Demo project

As mentioned in Section 7.2, a demo project was created in the installation path upon installation. This project contains a „C“ source code file Demo.c as well as the corresponding header file Demo.h and shows how to use the DLL interface functions.
The ***EthCan.Dll*** is loaded dynamically during runtime with the function ***LoadLibrary***() and the function pointer is acquired with the function ***GetProcAddress***().
The demo program is based on a WIN32 console application and is limited to the support of a CAN-Ethernet Gateway instance. After the program has been started and the ***EthCan.Dll*** has been loaded, a connection to a CAN-Ethernet Gateway, addressed by IP address and Port number, will be established. If the connection was established successfully, then a CAN message will be sent cyclically with the CAN-ID 0x180.
To verify the receipt of the CAN message via Ethernet and its subsequently transmission on the CAN bus, an monitor tool (e.g. PCAN View or CAN-REport) should be used.
Using the same tools it is possible to send CAN messages, which are transmitted to the PC via Ethernet and are displayed on the console window with its CAN-ID, data length code and data field.

7.3.5.2 Starting the Demo Program

The demo program requires various command line parameters, such as the IP address and the port number of the Gateway, as well as the transfer protocol to be used (TCP or UDP). The demo program is started as follows:

Open a command shell and switch to the folder of the executable file ***EthCanDemo.exe*** (which was given during installation).
To start the demo program enter the name of the executable program when given the input prompt, followed by the IP address, the Port number and the desired transfer protocol. The following shows two examples:

Example 1:

```
...\Release\EthCanDemo.exe  192.168.010.111  8234  TCP
```

Example 2:

```
...\Release\EthCanDemo.exe  192.168.010.111  8234  UDP
```

8 Updating the device Firmware

The serial interface on the CAN-Ethernet Gateway is also used to update the device firmware.
For updating the firmware, the Infineon ***MemTool*** software is used. It is provided free of charge on the CD accompanying the device or can be downloaded on the Infineon homepage (http://www.infineon.com).

The following requirements must be met in order to proceed with the firmware update:

1. Windows-OS based PC with the MemTool software installed and at least one free serial port (COM port).
2. A Nullmodem cable to connect the CAN-Ethernet Gateway to the PC.

8.1 Preparations

Proceed with the following steps to prepare the firmware update:

1. Power-on the device.
2. Connect the serial interface on the CAN-Ethernet Gateway with a free COM port on the PC via the Nullmodem cable mentioned above.
3. Activate the „Bootstrap-mode“. Therfore switch the boot switch to position ***„On“*** and reset the device.

The device is now ready for downloading the firmware.

8.2 Firmware download

The firmware of the CAN-Ethernet Gateway can be downloaded using the so-called MemTool software (see *Figure 20*).

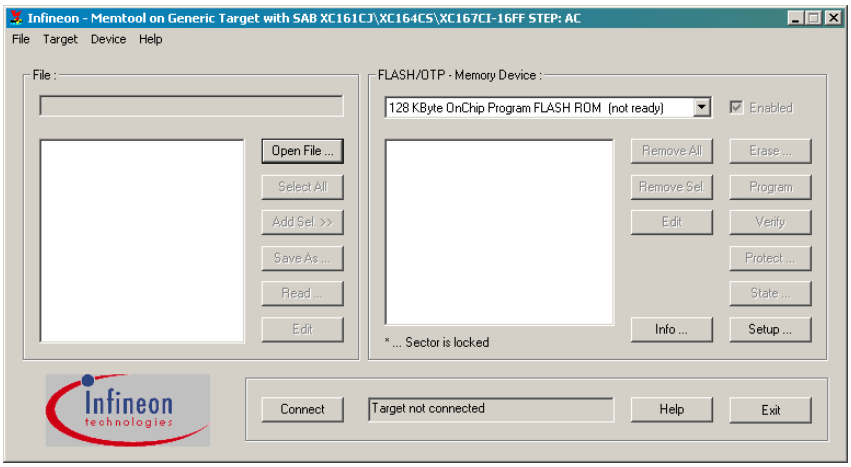

Figure 20: MemTool software

Proceed with the following steps to update the firmware to the CAN-Ethernet Gateway:

1. Go to menu ***„Target/Change“*** and select the controller: SAB XC161CJ\XC164CS\XC167CI-16FF STEP:AC
2. Next the serial communication interface needs to be configured. Therefore go to menu ***„Target/Setup Communication Port...“***. A configuration dialog window appears, where you can configure the serial interface used for firmware update.
3. Connect to the target using the ***„Connect“*** button. After the connection has been established successfully, the available memory sectors appear (see *Figure 21*).

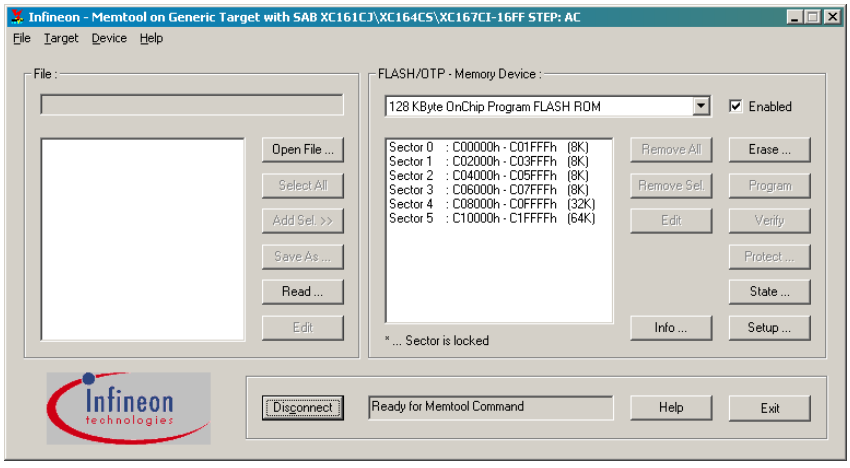

Figure 21: Memory sectors of the CAN-Ethernet Gateway

4. Erase the memory using the ***„Erase“*** button.
5. Open the new firmware file (e.g. E4050.h86) via menu **„File/Open File...“**.
6. Select all sectors from the right side list (see *Figure 22*) and assign them to the memory sectors on the left side of the application window by using the ***„Add Sel >>“*** button.

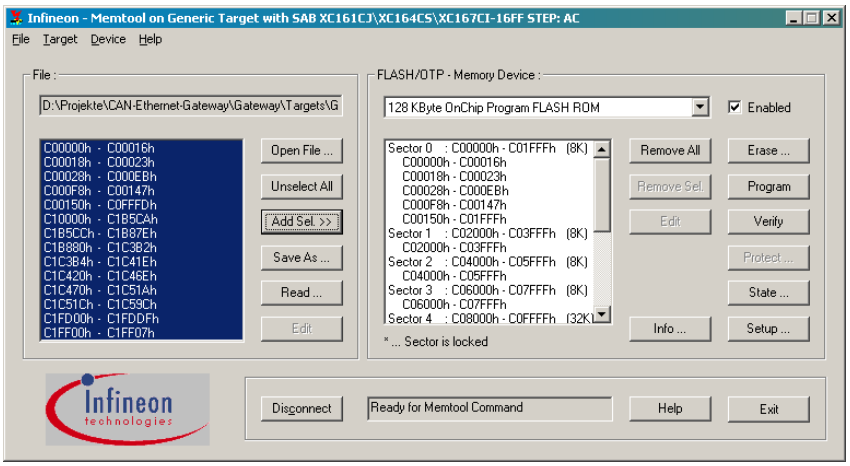

Figure 22: Memory areas and sector assignment

7. Click button ***„Program“*** to start the program download.
8. After successfully downloading the firmware set the boot switch to ***„Off“*** and reset the device by reset button or power cycle.

Note:
After downloading a firmware with a newer version number the resource files on the EEPROM will be discarded. With restarting the device after firmware update the standard configuration with default IP **192.168.10.111** and the settings described in the standard RC file is loaded.
The user specific configuration must be downloaded afterwards.

Index

Document:	CAN-Ethernet-Gateway
Document number:	L-1032e_10, Edition July 2010

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Return to: SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY

Fax : +49 (0) 36 61 / 62 79 99