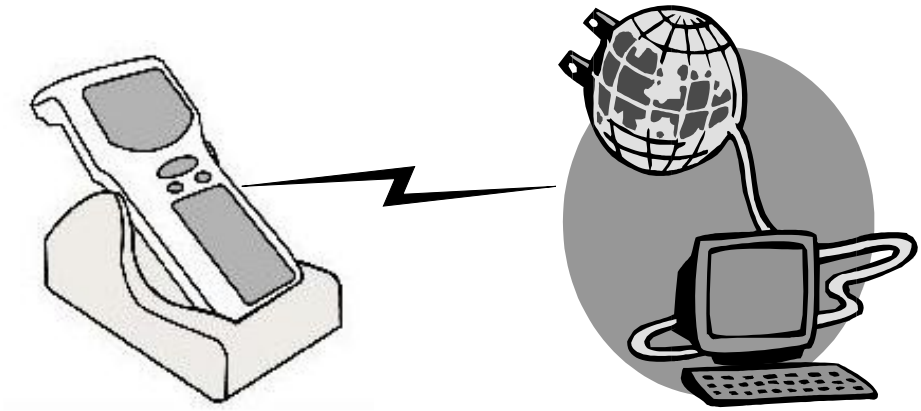
TCP-IP Library

for PHL Hand-held Terminals



Application Development Manual
Library versions:
CMWV0306 and CQWV0306

Table of Contents

1. Introduction

The TCP/IP Library for PHL Hand-held Terminals provides Opticon's PHL range of hand-held data collection terminals with the possibility to exchange data with computers on the Internet. This is a much more flexible and cost-effective solution than using a dedicated dial-in system.

Salient features:

- Dial-in using an Analog modem and GSM or GPRS modem/telephone
- Dial-in using an IrDA connection with a mobile phone that supports IrDA (Only supported by version LNAV0306, not by version LMAV0306)
- Connect to a LAN/Ethernet (with or without Internet access) using the Opticon Ethernet converter box.
- The FTP protocol is supported for file transfer (both active and passive FTP is supported).
- The POP and SMTP protocols are supported to receive and send email.
- The DNS protocol is supported to get the IP-addresses of domain names and vice versa.
- The IPCP protocol is supported to get your local IP address and the IP addresses op the ISP server and the (primary and secondary) DNS servers. (The IP addresses of the DNS servers can also be set manually in case the ISP server didn't provide the IP addresses of the primary and secondary DNS server.)
- The NTP protocol is supported to get the current time from a NTP time-server.

This manual describes the use of the TCP/IP library for PHL Hand-held Terminals in a C language application for the following data collection terminals:

Library version CMWV0306: PHL1300, PHL1700-10, PHL1700-20 and PHL2700.

Library version CQWV0306 (compatible with IrDA library): PHL1300 and PHL2700.

1.1. This manual

This manual assumes that the reader has sufficient understanding of application programming in the C language (ANSI C), and of common Internet terminology.
It describes how to use the functions of the TCP/IP Library for PHL Hand-held Terminals in application software. It does not describe in general how to develop applications for Opticon's barcode terminals. For this, we refer to the application development manual that is included in Opticon's "C-development kit for handheld barcode terminals".

2. Using the TCP/IP Library for PHL Hand-held terminals

This chapter gives general information on the use of the library in your application.

2.1. Selection and setup of the modem

Please observe the following:

- Due to the use of infrared communication between the PHL cradle and the terminal, only half-duplex operation is permitted. Some modems implement command echo in such a way that a command is echoed back before it is completely received. Even though the TCP/IP Library automatically disables the echo responses of most modems it's possible that the modem will still echo back. In this case the modem must be configured in the application so that it doesn't echo commands (consult its manual for instructions how to do this).
- The PHL cradles do not support hardware handshaking. Configure the modem so that it doesn't use hardware handshaking. If you use a PHL-1700 with the direct RS232 cable (instead of the cradle), you may be able to use a modem with hardware handshaking.
- Always use the fastest baudrate possible. Normally that's 115200bps or, when using the PHL1700 with the cradle, use 38400bps. This increases the connection speed and the reliability of the data connection.
- If you're using a wireless IrDA connection (only possible with the PHL2700 and PHL1300 with library version CQWV0306) set the baudrate to 9600bps.

2.2. Connecting the modem

The RS232 connector on the PHL cradle is a 9-pin, female, sub D connector wired as a DTE (Data Terminal Equipment) device. The RS232 connector on standard modems is wired as a DCE (Data Communication Equipment) device. For a DTE-DCE connection, you should simply connect the pins having the same signal names to each other (appendix B describes the pin assignment of the connectors).
The PHL direct RS232 cable is wired as a DCE. For a DCE-DCE connection, you have to use a "null modem" cable or adapter.

Some combinations of commonly available cables and adapters are suggested below. You can alternatively use a cable that is equivalently wired.

- To connect a modem that has a 9-pin female (9F) RS232 connector, you can use a straight (one-to-one connected) RS-232 cable with 9-pin male (9M) connectors at both ends.
- If you use the 9M-9F (crossed) PHL to PC cable that is supplied with the cradle, you need a 9M-9F null-modem adapter plus a 9M-9M gender changer.
- A 9M-9F null-modem adapter and 9M-9M gender changer is also necessary if you use the direct RS232 cable instead of a cradle.
- If the modem has only a 25F connector, you can use the above combinations together with a 9F to 25M adapter.

2.3. Using a wireless IrDA connection with a mobile phone

If you choose for an IrDA connection you can use a mobile phone that supports the IrDA protocol instead of using a modem to connect to an ISP server you'll have to use library version CQWV0306. At this moment it is only possible to make an IrDA connection with the PHL1300 and the PHL2700 and not with the PHL1700. In Appendix F all servers that have been successfully tested using an IrDA connection are listed.
More information about the use of IrDA can be found in the manual of the IrDA software library, which can be found in the C development kit of the PHL2700 and the PHL1300.

2.4. Making a GPRS connection with a mobile phone

A wireless connection with a mobile phone normally uses the standard GSM network of the mobile phone to dial an ISP server (using a telephone number and an Internet account). Many new types of mobile phones can also have a GPRS subsciption for faster mobile Internet. With this TCP/IP library it's now also possible to use these GPRS capabilities of mobile phones to make a faster wireless Internet connection.
How this can be done is described in detail in Appendix D.

2.5. Making an Ethernet connection with the Opticon Ethernet box

The Opticon Ethernet box allows you to connect an Opticon terminal to the Ethernet/LAN and has been designed to simulate the behavior of any modem and ISP server. Because of this, applications that work with a modem will also work with the Opticon Ethernet box without having to change the application software.

2.6. How to include the library in your application software

To build an application, see the programming manual of your type of PHL terminal. The functions of the TCP/IP library can be linked by adding one of the following lines to the link file, depending on if you want to use a modem or a wireless IrDA connection. It's best to place the line(s) in front of the line "LOAD c:\mccm77\CM77ISLC.LIB"

```
LOAD CMWV0306.LIB       ; Load TCP/IP library without IrDA support

        OR

LOAD CQMV0306.LIB       ; Load TCP/IP library without IrDA support
LOAD COWV0105.LIB       ; Load IrDA Library (for PHL2700 and PHL1300)
```

Note that if you're using TCP/IP library version CQWV0306, you'll also have to load the IrDA library (COWV010x). This will increase the size of your application with about 50kB. For this reason it is strongly recommended that you only use TCP/IP library version CQWV0306 when you're really planning to use a wireless IrDA connect to connect to a ISP server.

Below is an example of a link file that includes the TCP/IP library in an application for the PHL2700 with library version CMWV0306. (On the right side you can see how library version CQMV0306 can be linked.)

```
LISTMAP PUBLICS
LISTABS

FORMAT S2

ORDER startup, far_code, strings, const, literals
SECT  startup = $240000

ORDER ram_start,application,far_data,far_zerovars,far_initvars,heap
SECT  ram_start = $800000
SECT  application = $804000
SECT  heap =      $81D000
```

```
;PHL2700 without IrDA

LOAD STARTCBW
LOAD MAIN
LOAD MENU
LOAD DBASE
LOAD SETUP
LOAD CMWV0306.lib

LOAD c:\mccm77\CM77ISLC.LIB
END
```

```
;PHL2700 with IrDA

LOAD STARTCBW
LOAD MAIN
LOAD MENU
LOAD DBASE
LOAD SETUP
LOAD CQWV0306.lib     ;load TCP/IP library
LOAD COWV0105.lib     ;load IrDA library
LOAD c:\mccm77\CM77ISLC.LIB
```

The TCP-IP library for PHL Hand-held Terminals contains a header file "internet.h". This file contains function prototype declarations and constant definitions, and has to be included in all the C source files that use the functions of the TCP/IP library.

2.7. Parameters used by the library functions

The TCP/IP Library for PHL Hand-held Terminals provides functions to:

- Dial-in and connect to an ISP server or connect to Ethernet using the Opticon Ethernet Box
- Send and receive files using the FTP protocol (both active and passive FTP are supported)
- Receive email using the POP protocol
- Send email using the SMTP protocol (with and without authentication)
- Get IP-addresses of domain names and vice versa using the DNS protocol
- Get your local IP address, the IP addresses of the ISP server or the primary and secondary DNS servers using the IPCP protocol. (The IP addresses of the DNS servers can also be set manually in case the ISP server didn't provide the IP addresses of the primary and secondary DNS server.)
- Get the current time of an NTP time-server using the NTP protocol

The parameters used by these functions are contained in the structures of the type isp_config, ftp_config, pop_config, smtp_config, dns_config and ntp_config. The values in these structures have to be defined before calling the corresponding functions.

The TCP/IP Library exports a variable called '**internet_status'**, which determines if diagnostic messages of progress bars are output to the display of the terminal.

- If this variable is set to **0**, no messages will be output
- If this variable is set to **1**, messages will be output to the display that indicate the progress of operations ("connected to ISP", "file received", etc.).
- If this variable is set to **2**, the same messages will be output to the display that indicate the progress of operations. Also a progress indicator bar will be displayed during the transferring of files using POP, SMTP and FTP.
- If this variable is set to **3**, only a progress indicator bars will be displayed during the transferring of files using POP, SMTP and FTP but also during the initialisation of the connection (POP_INIT, FTP_INIT)

The library also exports another variable called '**debug_status'**, which determines if debug messages are output to the display and stored on the terminal.

- If this variable is set to **0**, no debug messages will be output or stored.
- If this variable is set to **1**, debug messages will be output to the display
- If this variable is set to **2**, messages will be output to the display and log files will be generated
- If this variable is set to **3**, a log file will be generated that can be analysed using the program **Ethereal**

Be aware that the use of this option can have negative effects on the performance of the application, especially when setting the variable to **0, 1** or **2**. When logging is needed it is recommended to use **3**, which is sufficient in almost all circumstances.

This variable should only be set to a different value then zero, if communication problems occur when using the TCP/IP library. This option can then be used to locate and solve the problem with or without the help of a programmer of Opticon Sensors Europe BV.

The file names of the generated log files are defined in the header file: internet.h, being:

- PPPLOG: Capture file containing all raw communication data between the handheld terminal and the ISP server. The log file is **compatible with the protocol analyzer Ethereal / Wireshark (free to download)** for an easy evaluation of a whole communication session.

- UIPLOG: Log file that is mainly intended for Opticon programmers to locate problems in the IP layer of this library

Both variables: 'debug_status' and 'internet_status' need to be initialized in your application before other functions of the TCP/IP library are called.

Initializing these variables can be done in the following two ways:

- Setting the values direct by adding the following lines to your source code:

```
internet_status = 2; // Show progress messages and a progress bar
debug_status = 0;    // Don't show or store any debug messages
```

- Setting the values indirectly by using the library function: TCPIP_init( ), this can be done by added the following line to you source code. (For more information see chapter 3.1).

```
TCPIP_init(2, 0);    // Show progress messages and a progress bar
```

2.8. Calling the library functions

For correct operation regular calls to the functions of the TCP-IP library must be made with only short intervals in between as long as the dial-up connection is up. For instance, if the application should wait for the user to press a key, depending on the server there may only be a limited time between two calls of the TCP-IP library, otherwise the connection can be lost.

3. TCP/IP Library Functions

3.1. TCPIP_init

Description Initializes the library variables 'internet_status' and 'debug_status'. The variable called **'internet_status'** determines whether diagnostic messages and/or progress bars are output to the display of the terminal or not. The variable called **'debug_status'** determines whether encrypted debug messages are output to the display of the terminal or if all received and sent data is stored on the terminal

Syntax void TCPIP_init(int internet_status, int debug_status);

Arguments

int internet_status	0	No status messages will be output .
	1	Messages will be output to the display that indicate the progress of operations ("connected to ISP", "file received", etc.).
	2	Messages will be output to the display that indicate the progress of operations. Also a progress indicator bar will be displayed during the transferring of files using POP, SMTP and FTP.
	3	Only progress indicator bars will be displayed during the transferring of files using POP, SMTP and FTP and also during the initialisation of the connection when using the FTP_INIT and POP_INIT commands
int debug_status	0	No debug messages or log file will be output.
	1	Encrypted messages will be output to the display. These messages can be used to determine which parts of the software are called.
	2	Besides displaying encrypted messages the application will also store all received and sent data and other information in 3 files on the terminal.
	3	A log file will be generated of all received and sent data that can be analysed using the program **Ethereal.** (No debug information is shown on the display)

Returns The file names of the generated log files are defined in internet.h:

- PPPLOG: Capture file containing all raw communication data between the handheld terminal and the ISP server. The log file is **compatible with the TCP/IP analyzing tool Ethereal** for an easy evaluation of a whole communication session.
- UIPLOG: Log file that is mainly intended for Opticon programmers to locate problems in the IP layer of this library
- NTPLOG: Hexadecimal encoded log file of NTP requests and answers
- DNSLOG: Hexadecimal encoded log file of DNS requests and answers

Remarks Both variables: 'debug_status' and 'internet_status' need to be initialized in your application before other functions of the TCP/IP library are called, the best way to do this is by calling this function with the desired values. **See also chapter 2.5.**
If 'internet_status' is set to 2 or 3 the progress indicator will only be displayed if 'debug_status' is set to 0 or 3.

3.2. connect

Description Uses the modem to dial-up to an Internet Service Provider or the Opticon Ethernet Box, and starts a PPP connection. After successful completion of this function, the terminal is connected to the Internet, and the other library functions can be used for using the FTP, SMTP, POP, DNS and NTP protocol.

Syntax int connect (struct isp_config *isp);

Arguments struct isp_config **isp*

```
struct isp_config{
        char phonenumber[20];
        char user[40];
        char pass[40];
        char com;
        char abort_key;
};
```

phonenumber
Null-terminated string containing the phone number of the Internet Service provider. In normal cases the dialing command of the modem will always start with ATDT, so the dialing command will be: “ATDT<isp. phonenumber>”.
If, in a rare case, the dialing command shouldn’t start with “ATDT”, this command can be overruled by specifying the complete dialing command in this variable.
(i.e isp.phonenumber -> “ATD*99***1#”)

user
Null-terminated string containing the user name that corresponds to the ISP account

pass
Null-terminated string containing the password for authenticating the user to the ISP

com
Specifies the serial port of the PHL that the modem is connected on:
0 Cable
1 Infrared Port
2 Cradle

abort_key
Specifies the key that can be used to abort the function.

Returns This function returns an error code (see appendix A)

Remarks If this function returns any other value than DIALUP_SUCCESS, the PPP connection was not established.

More information about the modem (AT-)commands that are being send to the modem or mobile phone when calling this connect( )-function, can be found in Appendix C.

When using the Opticon Ethernet Box, the variables 'phonenumber', 'user' and 'pass' are irrelevant, because the Ethernet Box will accept all.

3.3. connect_additional_ATcommand

Description See 3.2 connect, the only difference with the connect( )-function is that this function allows the user to specify an additional modem (AT-)command to set up the modem or mobile phone to the desired configuration.

Syntax int connect_additional_ATcommand(struct isp_config *isp, char *AT_Command)

Arguments struct isp_config **isp,* see 3.2 connect.

*char * AT_Command*
Additional modem (AT-)command string that will be send to the modem or mobile phone between the sending of the normal modem configuration commands and dialing command.

Returns This function returns an error code (see appendix A)

Remarks If this function returns any other value than DIALUP_SUCCESS, the PPP connection was not established.

More information about the modem (AT-)commands that are being send to the modem or mobile phone when calling this connect_additional_ATcommand( )-function, can be found in Appendix C.

3.4. connect_user_specified_ATcommand

Description See 3.2 connect, the only difference with the connect( )-function is that this function allows the user to specify its own modem (AT-)command to set up the modem or mobile phone to the desired configuration.

Syntax int connect_user_specified_ATcommand(struct isp_config *isp, char *AT_Command)

Arguments struct isp_config **isp,* see 3.2 connect.

*char * AT_Command*
The modem (AT-)command string that will be send to the modem or mobile phone between the sending of the modem reset command string (ATZ) and the dialing command.

Returns This function returns an error code (see appendix A)

Remarks If this function returns any other value than DIALUP_SUCCESS, the PPP connection was not established.

The modem reset string (ATZ) and the dialing command (i.e. ATDT<phonenumber>) will always be send and can't be overruled.

More information about the modem (AT-)commands that are normally send to the modem or mobile phone when calling the connect( )-function or the connect_additional_ATcommand( )-function, can be found in Appendix C. These commands can be used as reference when choosing the correct AT-command for this function.

3.5. register_manual_connection

Description This function allows the user to manually connect the PHL to an ISP server without using the connect( )-function of this library. After the connection has been established it must be registered to the TCP/IP library by using this function.

Syntax int register_manual_connection(struct isp_config *isp)

Arguments struct isp_config **isp*

```
struct isp_config{
        char phonenumber[20];
        char user[40];
        char pass[40];
        char com;
        char abort_key;
};
```

phonenumber
<not used by library>

user
Null-terminated string containing the user name that corresponds to the ISP account

pass
Null-terminated string containing the password for authenticating the user to the ISP.

com
Specifies the serial port of the PHL that the modem is connected on:
0 Cable
1 Infrared Port
2 Cradle

abort_key
Specifies the key that can be used to abort the function.

Returns This function returns an error code (see appendix A, same as connect( )-function)

Remarks If this function returns any other value than DIALUP_SUCCESS, the connection was not successfully registered.

More information about the modem (AT-)commands that are normally send to the modem or mobile phone when calling the connect( )-function or the connect_additional_ATcommand( )-function, can be found in Appendix C. These commands can be used as reference when choosing the correct AT-command for this function.

See Appendix E, for detailed information on how this function can be used.

3.6. disconnect

Description Terminates the PPP connection, hangs up the modem, mobile phone or Ethernet box and closes the COM port.

Syntax int disconnect (struct isp_config *isp)

Arguments struct isp_config *isp See 3.2. connect()

Returns This function returns an error code (see appendix A)

3.7. ftp_command

Description Function that executes different FTP commands. During an FTP session, files can be transmitted and received using the File Transfer Protocol. To start an FTP session, the function ftp_command must be called with its *command* argument set to FTP_INIT.
The function executes the specified command and then returns. It can be repeatedly called until the FTP session is closed (using the FTP_QUIT command).

Syntax int ftp_command (struct ftp_config *ftp, unsigned int command)

Arguments struct ftp_config **ftp*
A pointer to a ftp_config structure that contains the necessary parameters.

unsigned int *command*
A command code indicating the next FTP operation to be executed. The supported commands are the following:

FTP_INIT	Initialize connection with FTP server.
FTP_PUT	Upload a file.
FTP_PUT_PASSIVE	Upload a file using passive FTP.
FTP_GET	Download a file.
FTP_GET_PASSIVE	Download a file using passive FTP
FTP_CWD	Change working directory.
FTP_LIST	Get a directory listing.
FTP_LIST_PASSIVE	Get a directory listing using passive FTP
FTP_DEL	Delete file on FTP server.
FTP_REN	Rename file on FTP server.
FTP_QUIT	Disconnect from FTP server.

```
struct ftp_config{
        unsigned short IP1, IP2, IP3, IP4;
        char user[40];
        char pass[40];
        char local_file[13];
        char remote_file[40];
        unsigned char append;
        char abort_key;
};
```

IP1 … IP4	The 4 bytes specifying the IP address of the FTP-server
user	Contains the user name of the FTP server account
pass	Contains the password to be used for authentication to the FTP server.
local_file	**FTP_PUT_PASSIVE :** specifies the file that has to be uploaded to the FTP server. **FTP_GET_PASSIVE:** specifies what the downloaded file has to be named. **FTP_LIST_PASSIVE:** specifies the filename to which the directory listing is written. **FTP_REN**, this specifies the new name for the file. **FTP_INIT, FTP_CWD, FTP_DEL, FTP_QUIT**: this parameter is ignored.

remote_file	**FTP_PUT_PASSIVE:** specifies the file name of the uploaded file on the FTP server (if the string is empty, the FTP server is instructed to select a new, unique file name). **FTP_GET_PASSIVE:** specifies the name of the downloaded file. **FTP_LIST_PASSIVE:** specifies a file name mask (e.g. *.txt). If the string is empty, all files in the directory are listed. **FTP_CWD:** specifies the path of the new work directory (this may also be the parent directory by using '**..**') **FTP_REN:** specifies the file to be renamed. **FTP_DEL:** specifies the file to be deleted. **FTP_INIT, FTP_QUIT:** this parameter is ignored.
append	**FTP_GET_PASSIVE:** specifies if received data is to be appended to the specified local file or not. **FTP_PUT_PASSIVE:** specifies if the FTP server is to append the transmitted data to the specified remote file. (1 = Append, 0 = Overwrite)
abort_key	Specifies the key to abort the operation.

Returns This function returns an error code (see appendix A)

Remarks After the *ftp_command*-function returns, the application can evaluate what next command to issue. For example, it may examine the file downloaded by an **FTP_LIST** command, to determine if a certain file actually exists on the FTP-server before attempting to download it. However, note that the application should call *ftp_command* with only short intervals, until the FTP session has been closed, otherwise the FTP connection may time-out.

When using **FTP_PUT** to upload larger files, then the connection speed is likely to drop significantly after a few seconds. This is caused by the implementation of 'Delayed acknowledgements' on most FTP-servers and the fact that the IP-stack of the terminal only transmits one data packet at a time. Because the FTP-server will expect more frames, it will only send an acknowledgement after a certain delay, causing the connection speed to drop.

Due to the use of **Firewalls** that filter incoming data port connections from the server to the terminal it is possible that transferring files using (active) FTP will not work. For this reason this library supports the use of **Passive FTP (PASV)** by using the commands **FTP_PUT_PASSIVE**, **FTP_GET_PASSIVE** and **FTP_LIST_PASSIVE**. By using passive FTP the terminal opens the data connections instead of the server, so the firewalls will not block the data.

Example: FTP

```
#include "lib.h"
#include "internet.h"  // header file of the TCP/IP library
#include <stdio.h>
#include <string.h>

struct ftp_config ftp; // Declares structure which contains SMTP configurations
struct isp_config isp; // Declares structure which contains ISP configurations

void main (void)
{
        int error, key=0;

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud

        /* Put on the internet print status function with progress indicator bar*/
        TCPIP_init(2, 0); /* Put off the debug status function*/

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;

        // FTP IP-address = 62.58.50.14
        ftp.IP1 = 62; ftp.IP2 = 58; ftp.IP3 = 50; ftp.IP4 = 14; // change these values!
        strcpy (ftp.user, "phl1600");
        strcpy (ftp.pass, "test");
        ftp.append     = FALSE;                      // prices.txt will be overwritten
        ftp.abort_key  = CLR_KEY;                    // The CLR key will be used to abort

        while (1)
        {
                resetkey();                          // Clear keyboard buffer
                printf("\fFTP: choose\n");
                printf("1) Put file\n");
                printf("2) Get file\n");
                printf("3) Put&get files\n");

                do {key=getchar;}
                while ((key <'1')||(key>'3'));       // get option

                if ((error=connect(&isp))!= DIALUP_SUCCESS)  // connect to ISP
                {
                        printf("\nConnect: error #%i", error);
                        while(!kbhit());
                        disconnect(&isp);
                        continue;
                }

                // connect to FTP server
                if ((error=ftp_command(&ftp,FTP_INIT))!= FTP_SUCCESS)
                {
                        printf("\nftp_init: \nerror #%i", error);
                        while(!kbhit());
                        disconnect(&isp);
                        continue;
                }
```

```
                if ((key=='1')||(key=='3'))
                {
                        strcpy (ftp.local_file,"data.txt");
                        strcpy (ftp.remote_file,"barcodes.txt");
                        // upload file to FTP
                        if ((error= ftp_command(&ftp, FTP_PUT))!=FTP_SUCCESS)
                        {                                               // server
                                printf("\nFTP put:\nerror #%i", error);
                                if(ftp_connected()) // disconnect from FTP server
                                        ftp_command(&ftp,FTP_QUIT);
                                disconnect(&isp);                       // disconnect from ISP
                                while(!kbhit());
                                continue;
                        }
                }

                if ((key=='2'||key=='3'))
                {
                        strcpy (ftp.remote_file, "prices.txt");
                        strcpy (ftp.local_file, "prices.txt");
                        if ((error= ftp_command(&ftp,FTP_GET))!=FTP_SUCCESS)
                        {
                                printf("\nFTP get:\nerror #%i", error);

                                if(ftp_connected()) // disconnect from FTP server
                                        ftp_command(&ftp,FTP_QUIT);

                                disconnect(&isp);                       // disconnect from ISP
                                printf("\nHit a key");
                                while(!kbhit());
                                continue;
                        }
                }
                // disconnect from FTP server

                if(ftp_connected()) {
                        if ((error= ftp_command(&ftp,FTP_QUIT)) != FTP_SUCCESS)
                        {
                                printf("\nFTP quit:\nerror #%i", error);
                                disconnect(&isp);                               // disconnect from
                        ISP
                                printf("\nHit a key");
                                while(!kbhit());
                                continue;
                        }
                }
                disconnect(&isp);                       // disconnect from ISP
        }
}
```

3.8. ftp_connected

Description Function for checking the state of the connection with the FTP server

Syntax int ftp_connected(void)

Arguments none

Returns

0	Not connected
1	Connected

3.9. getmail

Description Retrieves email using the Post Office Protocol (POP3)

Syntax int getmail(struct pop_config *pop, int pop_command)

Arguments *int pop_command*
Specifies the type of command to be executed
POP_INIT : sets up a connection to the POP3 server.
POP_GET : retrieves a specified message

*struct pop_config *pop*

```
struct pop_config {
        unsigned short IP1, IP2, IP3, IP4;
        unsigned int msg_num;
        char local_file[13];
        char user[40];
        char pass[40];
        char nr_of_msgs;
        char incl_hdrs;
        char delete_msg;
        char msg_number;
        char abort_key;
}
```

IP1 .. IP4
The four bytes specifying the IP address of the POP server.

nr_of_msgs
After execution of the POP_INIT command, the number of messages on the server will be written into this variable.

incl_hdrs
To be used when executing the POP_GET command. This variable determines whether the header of the received message will be added to the file (local_file) or not.
(1 = “add header to file”, 0 = “only add body of message to file”)

delete_msg
To be used when executing the POP_GET command. This variable determines whether the received message will be deleted from the POP server or not, after the message was successfully received.
(1 = “delete message from server”, 0 = “leave message on server”)

msg_number
Before executing the POP_GET command, this variable has to contain the number of the message to be downloaded.

local_file
Specifies the file that a retrieved message will be written to.

user
Contains the username to identify the POP account.

pass
Contains the password for authentication to the POP-server.

abort_key
Contains the key that can be used to abort the function.

Returns This function returns an error code (see appendix A)

Remarks This function receives e-mail messages as plain text. Attachments are not decoded.

Example: POP (getmail)

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the internet header file, which declares the functions
#include <stdio.h>
#include <string.h>

struct pop_config pop; // Declares structure which contains POP configurations
struct isp_config isp; // Declares structure which contains ISP configurations

void main (void){
        int error=0;

        /* Put on the internet print status function with progress indicator bar*/
        TCPIP_init(2, 0); /* Put off the debug status function*/

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud
        resetkey();                   // Clears keyboard buffer

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;

        // POP IP-address = 62.58.50.10              // change these values!
        pop.IP1 = 62; pop.IP2 = 58; pop.IP3 = 50; pop.IP4 = 10;
        strcpy (pop.user, "phl1600");
        strcpy (pop.pass, "test");
        strcpy (pop.local_file, "database.txt");

        pop.incl_hdrs = FALSE;                //don't include hearder to file
        pop.delete_msg = TRUE;                //delete message from server

        pop.abort_key  = CLR_KEY;             // The CLR key will be used to abort

        if (error=connect(&isp)){
                switch(error){
                        case -1:
                                printf("connection error");
                                break;
                        case -2:
                                printf("modem not connected or not responding");
                                break;
                        case -3:
                                printf("ISP not responding");
                                break;
                }
        }
        else if ((error=getmail(&pop,POP_INIT))){
                switch(error){         // Only two errors are evaluated…
                        case -1:
                                printf("User aborted by pressing key");
                                break;
                        case -3:
                                printf("Application polled too many times");
                                break;
                        default:
                                printf("\nError #%d",error);
                }
        }
        else {
                printf("\n\n%d MESSAGES FOUND",pop.nr_of_msgs);

                if(pop.nr_of_msgs!=0) {
                        //get last message from server
                        pop.msg_number = pop.nr_of_msgs;
                        error=getmail(&pop,POP_GET);
                }

                if(!error)
                        error=getmail(&pop,POP_QUIT);
```

```
		if(error) {
			switch(error){		// Only two errors are evaluated…
				case -1:
					printf("User aborted by pressing key");
					break;
				case -3:
					printf("Application polled too many times");
					break;
				default:
					printf("\nError #%d",error);
			}
		}

	}
	disconnect(&isp);

	printf("\nHit a key");
	while(!kbhit()) idle();
}
```

3.10.sendmail

Description Sends email using the Simple Mail Transfer Protocol (SMTP).

Syntax int sendmail(struct mail_config *mail)

Arguments struct mail_config *mail

```
struct mail_config{
        unsigned short IP1, IP2, IP3, IP4;
        char sender[40];
        char recipient[40];
        char subject[40];
        char reply[40];
        char *smtp_msg.
        char local_file;
        char attach;
        char base64;
        char abort_key;
};
```

IP1.. IP4
The four bytes specifying the IP address of the SMTP Server

sender
Contains the email address of the sender, as will appear in the FROM: header of the transmitted email.
Note that some SMTP servers will only send the email if the domain of the senders email address matches the domain of the SMTP server

recipient
Contains the name/email address of the recipient. It will appear in the TO: header of the transmitted email.

subject
Contains the subject line, as will appear in the SUBJECT: header of the transmitted email.

reply
Contains the email address as will appear in the REPLY-TO: header of the transmitted email.

smtp_msg
This pointer to a string specifies an additional message, that is included in the email when the file is transmitted as an attachment.

local_file
Contains the name of the file that is either send in the message body or that is to be sent as an attachment.

attach
Flag that specifies whether the local file is to be sent in the message body or as an attachment to the email.
(1='send the file as an attachment'; 0='include the file contents in the message body').
If the file is sent as an attachment, it will be transmitted as MIME type application: octet_stream.

base64
Specifies whether the attachment uses base64 encoding or 7-bit ASCII encoding. Base64 encoding is appropriate for binary files. Base64 encoding should also be selected for ASCII text files if they contain single <CR> or <LF> characters rather than <CR><LF> sequences, or if they contain lines that are longer than 1000 characters.
(1 = "base 64 encoding"; 0 = "7-bit ASCII encoding")

abort_key
Specifies the key for aborting the operation.

Returns
The function returns an error code (see appendix A)

Remarks
- In case the file is transmitted in the body of the email or as a 7-bit ASCII encoded attachment, any single <CR> or <LF> characters will be expanded to <CR><LF>. In addition, <CR><LF> will be inserted after every sequence of 1000 characters that does not include <CR>, <LF> or <CR><LF>. The 8th bit of each character will be set to 0. These modifications are necessary to conform to the SMTP protocol.
- To be sure that a text or binary file is transferred without any modification, send the file as a binary attachment (base64 encoded).
- When using an **Opticon Ethernet box or a GPRS connection** you don't login with a user name and password to an ISP server. Therefor most SMTP will not allow you to send emails using that connection, unless you use **'sendmail_authenticated'** to authenticate yourself first.
- It is possible that an SMTP server doesn't use the default port 25 for receiving emails,but uses a different port number. In this case use the function: '**change_SMTP_remote_port_nr**'

3.11.sendmail_authenticated

Description Sends email using the Simple Mail Transfer Protocol (SMTP) with (plain) authentication, so mail servers can be used that aren't linked to the used ISP-server

Syntax int sendmail_authenticated(mail_config *mstruct, unsigned char *username, unsigned char *password)

Arguments

struct mail_config *mail	(see 3.10 Sendmail)
username	Pointer to SMTP Username (if NULL, no authentication is performed)
password	Pointer to SMTP Password (if NULL, no authentication is performed)

Returns
The function returns an error code (see appendix A)

Remarks:
- Not every SMTP server will support (PLAIN) authentication or allow connections from other ISP's
- See also the remarks of 3.10 sendmail
- sendmail_authenticated(mstruct, NULL, NULL) is the equivalent of sendmail(mstruct)

Example: SMTP (sendmail)

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the internet header file, which declares the functions
#include <stdio.h>
#include <string.h>

struct mail_config mail; // Declares structure which contains SMTP configurations
struct isp_config isp; // Declares structure which contains ISP configurations

void main (void){
        int error=0;
        char* textMessage = "ATTACHMENT INCLUDED";

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud

        /* Put on the internet print status function with progress indicator bar*/
        TCPIP_init(2, 0); /* Put off the debug status function*/

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;

        // SMTP IP-address = 62.58.50.46             //change these values
        mail.IP1 = 62; mail.IP2 = 58; mail.IP3 = 50; mail.IP4 = 46;

        strcpy(mail.sender,"phl1600@zonnet.nl");            //change this value
        strcpy(mail.recipient, "database@opticon.nl");      //change this value
        strcpy(mail.reply," database@opticon.nl");          //change this value

        strcpy(mail.subject,"testing PHL1600 email");
        strcpy(mail.local_file,"database.txt");
        mail.smtp_msg = textMessage;
        mail.attach = TRUE;           // attach as file to email
        mail.base64 = TRUE;           // use base64 code to send file as binary data
        mail.abort_key= CLR_KEY;      // The CLR key will be used to abort

        if (error=connect(&isp)){
                switch(error){
                        case -1:
                                printf("\nerror opening COM port");
                                break;
                        case -2:
                                printf("\nmodem not connected or not responding");
                                break;
                        case -3:
                                printf("\nISP not responding");
                                break;
                }
        }
        else if (error=sendmail(&mail)){
                switch(error){         // Only first three errors are evaluated…
                        case -1:
                                printf("User aborted by pressing key");
                                break;
                        case -2:
                                printf("File does not exist");
                                break;
                        case -3:
                                printf("Application polled too many times");
                                break;
                        default:
                                printf("\nOther Error");
                }
        }
        disconnect(&isp);
        while(!kbhit()) idle();
}
```

3.12.get_host_by_name

Description Returns the IP address of the specified domain name using the Domain Name System (DNS) protocol.

Syntax int get_host_by_name(struct dns_config *dns)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

domain_address
Contains the domain name of which the IP address should be retrieved

IP1.. IP4
The four bytes in which the IP address of the specified domain name is returned

abort_key
Specifies the key for aborting the operation.

Returns This function returns an error code (see appendix A)

Remarks If the function returns the error: -37 (ERR_DNS_NO_SERVERS_FOUND), use the functions: set_primary(/secondary)_DNS_server_IP to set the IP addresses of the DNS servers manually. (Note that most DNS servers will only respond to their own ISP server)

3.13.get_host_by_addr

Description Returns the domain name of the specified IP address using the Domain Name System (DNS) protocol.

Syntax int get_host_by_addr(struct dns_config *dns)

Arguments struct dns_config *dns

```
struct dns_config {
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

domain_address
String in which the domain name of specified IP address is returned

IP1.. IP4
Contain the IP address of the domain name that should be retrieved.

abort_key
Specifies the key for aborting the operation.

Returns This function returns an error code (see appendix A)

Remarks See 3.12 get_host_by_name

Example: DNS (get_host_by_name/address)

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the internet header file, which declares the functions
#include <stdio.h>
#include <string.h>

struct dns_config dns; // Declares structure which contains DNS configurations
struct isp_config isp; // Declares structure which contains ISP configurations

void main (void){
        int error=0;

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud
        resetkey();                   // Clears keyboard buffer

        /* Put on the internet print status function */
        TCPIP_init(1, 0); /* Put off the internet debug status function */

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;
        dns.abort_key = CLR_KEY;    //The CLR key will be used to abort

        strcpy (dns.domain_address,"ftp.aol.com");   // change this value!

        if (error=connect(&isp)){
                switch(error){
                        case -1:
                                printf("\nerror opening COM port");
                                break;
                        case -2:
                                printf("\nmodem not connected or not responding");
                                break;
                        case -3:
                                printf("\nISP not responding");
                                break;
                }
        }
        else if (error=get_host_by_name(&dns)) {
                printf("error %d occurred", error);
        }
        else {
                printf("\fDomain name:\n%s\n IP ->\n%03d.%03d.%03d.%03d",
                          dns.domain_address, dns.IP1, dns.IP2, dns.IP3, dns.IP4);

                // IP-address = 205.188.212.118              //change these values
                dns.IP1 = 205; dns.IP2 = 188; dns.IP3 = 212; dns.IP4 = 118;

                if (error=get_host_by_addr(&dns))
                        printf("error %d occurred", error);
                else {
                        printf("\fIP-address:\n%03d.%03d.%03d.%03d\nDomain name:\n%s\n",
                        dns.IP1, dns.IP2, dns.IP3, dns.IP4, dns.domain_address);
                }
        }
        disconnect(&isp);
        while(!kbhit()) idle();
}
```

3.14.get_local_IP_address

Description Returns the local IP address that is retrieved from the ISP server using the IPCP protocol.

Syntax int get_local_ip_address(struct dns_config *dns, int timeout)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

IP1.. IP4
The four bytes in which the local IP address is returned

abort_key
Specifies the key for aborting the operation.

timeout
Time out in seconds

Returns This function returns an error code (see appendix A)

3.15.get_ISP_server_IP

Description Returns the IP address of the ISP server that is retrieved using the IPCP protocol.

Syntax int get_ISP_server_IP(struct dns_config *dns, int timeout)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

IP1.. IP4
The four bytes in which the IP address of the ISP server is returned

abort_key
Specifies the key for aborting the operation.

timeout
Time out in seconds

Returns This function returns an error code (see appendix A)

3.16.get_primary_DNS_server_IP

Description Returns the IP address of the primary DNS server that is retrieved from the ISP

Syntax int get_primary_DNS_server_IP(struct dns_config *dns, int timeout)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

IP1.. IP4
The four bytes in which the IP address of the primary DNS server is returned

abort_key
Specifies the key for aborting the operation.

timeout
Time out in seconds

Returns This function returns an error code (see appendix A)

Remarks If the function returns the error: -37 (ERR_DNS_NO_SERVERS_FOUND), use the functions: set_primary_DNS_server_IP to set the IP address of the DNS server manually. (Note that most DNS servers will only respond to their own ISP server)

3.17.get_secondary_DNS_server_IP

Description Returns the IP address of the secondary DNS server that is retrieved from the ISP

Syntax int get_secondary_DNS_server_IP(struct dns_config *dns, int timeout)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

IP1.. IP4
The four bytes in which the IP address of the secondary DNS server is returned

abort_key
Specifies the key for aborting the operation.

timeout
Time out in seconds

Returns This function returns an error code (see appendix A)

Remarks If the function returns the error: -37 (ERR_DNS_NO_SERVERS_FOUND), use the functions: set_secondary_DNS_server_IP to set the IP address of the DNS server manually. (Note that most DNS servers will only respond to their own ISP server)

Example: get IP addresses

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the internet header file, which declares the functions
#include <stdio.h>
#include <string.h>

struct dns_config dns; // Declares structure which contains DNS configurations
struct isp_config isp; // Declares structure which contains ISP configurations

void main (void){
        int error=0;
        int dns_choice;

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud
        resetkey();                   // Clears keyboard buffer

        /* Put on the internet print status function */
        TCPIP_init(1, 0); /* Put off the internet debug status function */

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;
        dns.abort_key = CLR_KEY;    //The CLR key will be used to abort

        dns_choice = 1;                      //change this value if necessary

        printf("Starting...");

        if (error=connect(&isp)) {
                switch(error) {
                        case -1:
                                printf("\nerror opening COM port");
                                break;
                        case -2:
                                printf("\nmodem not connected or not responding");
                                break;
                        case -3:
                                printf("\nISP not responding");
                                break;
                }
        }
        else {
                switch(dns_choice) {
                        case 1:
                                error=get_local_IP_address(&dns, 30); // 30 second time out
                                break;
                        case 2:
                                error=get_ISP_server_IP(&dns, 30);
                                break;
                        case 3:
                                error=get_primary_DNS_server_IP(&dns, 30);
                                break;
                        case 4:
                                error=get_secondary_DNS_server_IP(&dns, 30);
                                break;
                }

                if(!error) {
                        printf("\nRequested IP:\n%03d.%03d.%03d.%03d\n",
                                        dns.IP1, dns.IP2, dns.IP3, dns.IP4);
                }
                else
                        printf("error %d occurred", error);
        }
```

```
        disconnect(&isp);
        while(!kbhit()) idle();
}
```

3.18.set_primary_DNS_server_IP / set_secondary_DNS_server_IP

Description Sets the IP addresses of respectively the primary and secondary DNS server.

Syntax void set_primary_DNS_server_IP(struct dns_config *dns)
void set_secondary_DNS_server_IP(struct dns_config *dns)

Arguments struct dns_config *dns

```
struct dns_config{
        char domain_address[80];
        unsigned short IP1, IP2, IP3, IP4;
        char abort_key;
};
```

IP1.. IP4
The four bytes in which the IP addresses of the primary or secondary DNS server should be set.

abort_key and domain_address
Aren't used in this function

Returns *None*

Remarks Use this function only if the functions get_primary(/secondary)_DNS_server_IP return the error: -37 (ERR_DNS_NO_SERVERS_FOUND). This indicates the ISP server didn't provide any IP addresses of DNS servers during the IPCP negotiation.

If the ISP server does provide any IP addresses of DNS servers, the IP addresses that are set using this function will be overwritten during the IPCP negotiation.

Note that most DNS servers will only respond to their own ISP server, so only specify DNS servers that work with the chosen ISP server.

3.19.get_NTP_time

Description Returns the current time from an NTP server using the Network Time Protocol (NTP)

Syntax int get_NTP_time(struct ntp_config *ntp, int time_zone)

Arguments

```
struct ntp_config
{
        unsigned short IP1, IP2, IP3, IP4;
        struct date *date;
        struct time *time;
        u8_t abort_key;
};
```

IP1.. IP4
The four bytes specifying the IP address of the NTP Server

date and time
Date and time structure as can be found in the file 'lib.h'., see below:

```
struct date {
        unsigned int  da_year;
        unsigned char da_day, da_mon;
}

struct time {
        unsigned char ti_hour, ti_min, ti_sec;
}
```

Before calling the get_NTP_function the structures 'date' and 'time' should be configured with the current local time and date of the PHL. (Offcourse, this local time and date doesn't have to be the correct time)

After the get_NTP_function is succesfully called these time and date structures will contain the returned local time from the NTP server.

abort_key
Specifies the key for aborting the operation.

time_zone
Local time zone in comparison with Greenwich Mean Time (GMT), also called Universal Coordinated Time (UTC).
i.e. Central European Time CET is +1 (or +2 at summer time)

Returns This function returns an error code (see appendix A)

Remarks It is strongly recommended to use DNS to get the IP addresses from the NTP servers (by using the function get_host_by_name), because the IP addresses of NTP servers will often change from time to time.

In Appendix G a list of almost 200 NTP servers can be found.

Example: NTP (get_NTP_time)

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the internet header file, which declares the functions
#include <stdio.h>
#include <string.h>

struct ntp_config ntp; // Declares structure which contains NTP configurations
struct isp_config isp; // Declares structure which contains ISP configurations
struct date dates;
struct time times;

void main (void)
{
        int error=0;

        setfont(SMALL_FONT, NULL);    // small font
        systemsetting ("SZ");         // COM speed set to 115200 baud
        resetkey();                   // Clears keyboard buffer

        /* Put on the internet print status function */
        TCPIP_init(1, 0); /* Put off the internet debug status function */

        // Internet Service Provider Settings
        strcpy (isp.phonenumber, "0676075030");      // change this value!
        strcpy (isp.user , "phl1600");               // change this value!
        strcpy (isp.pass, "test");                   // change this value!
        isp.com = COM0;               // Modem is connected with the cable
        isp.abort_key = CLR_KEY;
        ntp.abort_key = CLR_KEY;    //The CLR key will be used to abort

        ntp.IP1 = 62;         // IP from ntp1.theinternetone.net (62.4.94.211)
        ntp.IP2 = 4;          // IP-addresses can be changed without prior notice
        ntp.IP3 = 94;         // for this reason always use DNS (getHostbyName) to
        ntp.IP4 = 211;        // retrieve the IP-addresses of the used NTP servers

        if (error=connect(&isp))
                printf("\fError occurred:%d",error);
        else {
                TCPIP_init(0, 0);
                printf("\f\n Please wait  ");

                while(1) {
                        gettime(&times);       //get current local time
                        getdate(&dates);       //get current date

                        ntp.date = &dates;     //set date and time parameters
                        ntp.time = &times;

                        if(error=get_NTP_time(&ntp, +2)) { // get time from NTP
                                printf("\fError occurred:%d",error);
                                break;
                        }
                        else {
                                gotoxy(0,1);
                                printf("   %02d:%02d:%02d ",ntp.time->ti_hour,       \
                                        ntp.time->ti_min,ntp.time->ti_sec);
                                gotoxy(0,2);
                                printf("  %02d-%02d-%04d",ntp.date->da_day,          \
                                        ntp.date->da_mon,ntp.date->da_year);

                                setdate(ntp.date);     settime(ntp.time);
                        }
                }
        }
        disconnect(&isp);      //hang up

        while(!kbhit())
                idle();
}
```

3.20.change_FTP_remote_port_nr

Description Changes the port number of the remote FTP server to the specified port number

Syntax int change_FTP_remote_port_nr(unsigned short port_nr)

Arguments *unsigned short port_nr*
Specifies the port number of the remote FTP server.
Port numbers 25(SMTP), 53(DNS), 110(POP), 123(NTP) are reserved port numbers and may therefor not be used.

Returns

OK	On success
ERR_INVALID_PORT_NUMBER	If invalid port number is specified

Remarks This function should be used if the remote FTP server doesn't operate on port 21. If the port number needs to be changed, call this function after establishing a connection and before using the 'ftp_command' function. The FTP port number is reset to default (port 21) when the 'connect' function is called.

3.21.change_SMTP_remote_port_nr

Description Changes the port number of the remote SMTP server to the specified port number

Syntax int change_SMTP_remote_port_nr(unsigned short port_nr)

Arguments *unsigned short port_nr*
Specifies the port number of the remote SMTP server.
Port numbers 21(FTP), 53(DNS), 110(POP), 123(NTP) are reserved port numbers and may therefor not be used.

Returns *See 3.20 'change_FTP_remote_port_nr'*

Remarks This function should only be used if the remote SMTP server does not operate on port 25. If the port number needs to be changed, call this function after establishing a connection and before using the 'sendmail' function. The SMTP port number is reset to default (port 25) when the 'connect' function is called.

3.22. change_POP_remote_port_nr

Description Changes the port number of the remote POP server, to the specified port number

Syntax int change_POP_remote_port_nr(unsigned short port_nr)

Arguments *unsigned short port_nr*
Specifies the port number of the remote POP server.
Port numbers 21(FTP), 25(SMTP), 53(DNS), 123(NTP) are reserved port numbers and may therefor not be used.

Returns *See 3.20 'change_FTP_remote_port_nr'*

Remarks This function should be used if the remote POP server doesn't operate on port 110. If the port number needs to be changed, call this function after establishing a connection and before using the 'getmail' function. The POP port number is reset to default (port 110) when the 'connect' function is called.

3.23.tcpip_lib_version

Description Returns a pointer to the version string of the TCP/IP library.

Syntax char *tcpip_lib_version(void)

Syntax char *tcpip_lib_version(void)

Arguments.....**-**

Remarks It's recommended that an application includes the possibility to display this version string for diagnostic purposes.

Example: Library version

```
#include "lib.h"
#include "internet.h"
#include <stdio.h>

void main(void)
{
        printf("\fLib. version:\n\n  %s", tcpip_lib_version());
        while(!kbhit());
}
```

APPENDIX A: Error codes

Can occur with function *								Name	Description
C	D	F	G	S	DNS	IP	NTP		
*								DIALUP_SUCCESS	(No error): PPP connection established.
	*							DISCONNECTED	(No Error): PPP connection terminated, modem hung up
		*						FTP_DONE	(No Error): FTP command was successfully executed
			*					EMAIL_RETRIEVED	(No Error): Email was successfully retrieved
				*				EMAIL_SENT	(No Error): Email was successfully sent
					*			DNS_OK	(No Error): DNS information was succesfully retreived
						*		IP_OK	(No Error): IP address information was succesfully retreived
							*	NTP_OK	(No Error): Time was succesfully retreived from NTP server
*	*	*	*	*	*	*	*	ERR_USER_ABORTED	User aborted the operation using the specified key
*								ERR_NODIALTONE	No dial tone
*								ERR_BUSY	Telephone line busy
*								ERR_NOANSWER	Remote system did not pick up the phone
*								ERR_NOCARRIER	No carrier detected
*	*							ERR_MODEM	The modem has encountered an general error
*								ERR_DELAYED	Modem currently not allowed to dial
*								ERR_BLACKLISTED	Number is on modem's blacklist: not allowed to dial
*	*							ERR_NORESPONSE	No response from modem
*	*	*	*	*				ERR_BUFFER	A serial interface buffer overrun occurred.
*								ERR_DIALUP_COM_PORT	Could not open the specified COM port
	*							ERR_CLOSING_COM_PORT	An error occurred while closing COM port
	*							ERR_OFF_HOOK	No modem OK reply to the ATH0(off hook) command
*								ERR_LOGIN_FAIL	Failed to login with ISP
		*	*	*	*	*	*	ERR_POLLED	TCP connection timeout
		*	*	*				ERR_ILLEGAL_COMMAND	protocol command not accepted by the server
		*	*	*				ERR_SERVER_REJECTED	TCP acknowledgement packet rejected by server
		*	*	*				ERR_CONNECTION_ABORTED	TCP connection reset by peer
		*	*	*				ERR_TCP_RECEIVED_RESET	TCP connection reset by peer
		*						ERR_FTP_NO_CONNECTION	Not connected to FTP server (FTP_INIT command!)
		*						ERR_FTP_TEMP_ERROR	FTP server temporary error (try again later)
		*						ERR_FTP_NOT_LOGGED_IN	The user is not logged in on the FTP server
		*						ERR_FTP_SERVER_DOWN	No response from FTP server
		*						ERR_FTP_SYNTAX	Incorrect FTP protocol command syntax
		*						ERR_FTP_NO_ACCOUNT	Invalid FTP account specified
		*	*	*				ERR_NO_SUCH_FILE	Specified file does not exist
			*					ERR_POP_USER_REJECTED	Invalid POP3 account specified
			*					ERR_POP_PASS_REJECTED	POP3 password not accepted
			*					ERR_POP_SERVER_DOWN	No response from POP server
				*				ERR_SMTP_SENDER_REJECTED	SMTP server rejected "sender" value
				*				ERR_SMTP_RECIPIENT_REJECTED	SMTP server rejected "recipient" value
				*				ERR_SMTP_DATA_REJECTED	SMTP server rejected data
				*				ERR_SMTP_SERVER_DOWN	No response from SMTP server
				*				ERR_SMTP_AUTH_FAILED	SMTP server rejected username/pass or doesn't support auth.
					*			ERR_DNS_NO_RESPONSE	DNS servers didn't respond
					*			ERR_DNS_DATA_INVALID	The response of the DNS servers wasn't in the correct format
					*			ERR_DNS_NO_SERVERS_FOUND	The ISP-server didn't give the IP-addresses of a DNS servers
					*			ERR_DNS_NAME_ERROR	The DNS servers don't know the given IP or domain address
							*	ERR_NTP_NO_RESPONSE	NTP server didn't respond
							*	ERR_NTP_DATA_INVALID	The response of the NTP server wasn't in the correct format

*****

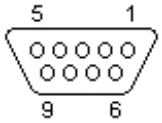
Code	Function(s)
C	**Connect()**
D	**Disconnect()**
F	**ftp_command()**
G	**getmain()**
S	**sendmail() / sendmail_authenticated()**
DNS	**get_host_by_name(), get_host_by_addr()**
IP	**get_local_IP_address(), get_(ISP/primaryDNS/secondaryDNS)_server_IP()**

The header file "internet.h", that is used by the TCP-IP library, contains all the definitions of error codes that are listed above.

APPENDIX B: Serial connection from terminal to modem

9 pin sub-D female connector
The table describes the connection of the pins on the PHL cradle, the PHL-1700 RS-232 cable, and commonly available modems that have a 9-pin sub-D RS232 connector.
The PHL cradle should be connected to the modem via a one-to-one connected cable.
The PHL terminal with RS232 cable should be connected via a null-modem, which has crossed connections swapping pins 2 with 3 and pins 7 with 8.

pin #	signal name	DTE (PHL cradle)	DCE (Modem, PHL-1700 cable)	Description
1	[CD]			Carrier Detect
2	[RxD]	in	out	Receive Data
3	[TxD]	out	in	Transmit Data
4	[DTR]			Data Terminal Ready
5	[GND]	gnd	gnd	Signal Ground
6	[DSR]			Data Set Ready
7	[RTS]	out	in	Request To Send
8	[CTS]	in	out	Clear To Send
9	[RI]			Ring Indicator

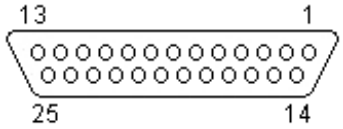


Sub-D 25 pin female connector

This table describes the connection of the pins on a 25 pin sub-D connector as found on commonly available modems.

pin #	signal name	DCE (Modem)	Description
1	[shield]		Shield ground
2	[TxD]	in	Transmit Data
3	[RxD]	out	Receive Data
4	[RTS]	in	Request To Send
5	[CTS]	out	Clear To Send
6	[DSR]		Data Set Ready
7	[GND]	gnd	Signal Ground
8	[CD]		Carrier Detect
20	[DTR]		Data Terminal Ready
22	[RI]		Ring Indicator

Note: 9 – 19 / 21 / 23 – 25 are not used

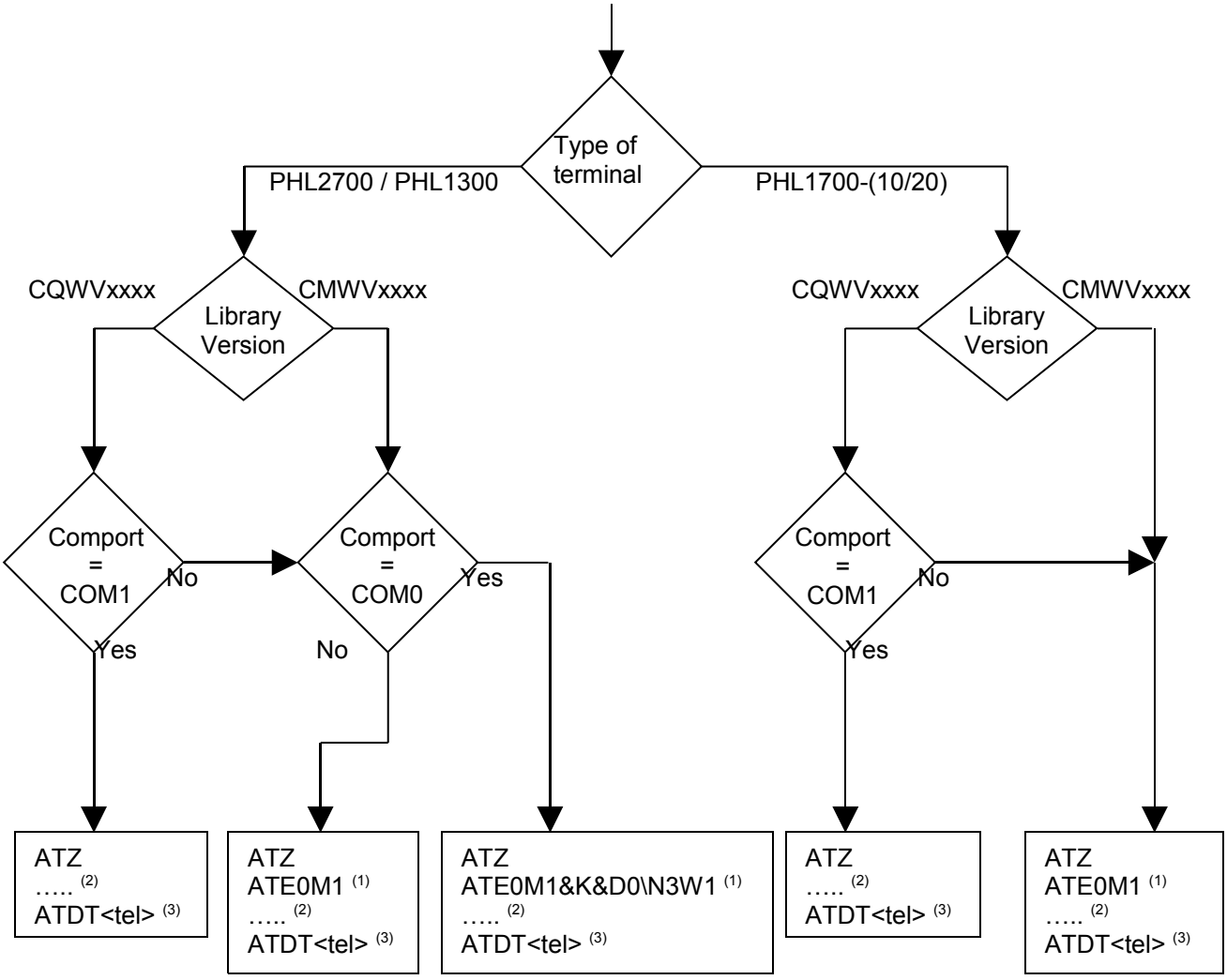


APPENDIX C: AT-command diagram of the connect-functions

The diagram below shows which modem commands are sent to the modem of mobile phone when calling the connect( )-function or the connect_additional_Atcommand( )-function *.

As can be seen below, the AT commands that are being sent under which circumstances depend on the:

- Type of terminal PHL2700/1300 or the PHL1700
- Library version CMWVxxxx (normal) or CQMVxxxx (IrDA compatible)
- Selected comport Direct cable (COM0), Infrared(COM1), Cradle(COM2)



(1) The standard configuration commands (1) won't be sent when using the connect_user_specified_ATcommand( )-function.

(2) The positions of the lines of dots (…..) in the diagram above, show the place where the additional AT-commands will be sent when using the connect_additional_ATcommand( )-function or the connect_user_specified_ATcommand( )-function.

(3) In normal cases the dialing command will always start with ATDT, so the dialing command will be: "ATDT<isp. phonenumber>", if in a rare case the dialing command shouldn't start with ATDT, this command can be overruled by specifying the complete dialing command in the variable: isp. phonenumber.

(i.e isp.phonenumber -> “ATD*99***1#”)

APPENDIX D: Making a GPRS connection with a mobile phone.

With this TCP/IP library it’s also possible to use the GPRS capabilities of mobile phones with GPRS subscription.

The advantage of using the GPRS network of a mobile phone is that establishing a GPRS connection will be much faster than establishing a connection with an ISP server using the GSM network.

Note: Before trying to make a wireless GPRS connection first make sure that the mobile phone that’s being used has a working GPRS subscription.

Making a GPRS connection can’t be done by using the standard connect( )-function of this TCP/IP library. Instead the function: connect_additional_ATcommand( ) should be used, because an additional modem (AT-)commands needs to be send to the mobile phone. This is the following AT-command:

AT+CGDCONT=1,”IP”,”INTERNET” (some GPRS providers may demand a specified IP-address or domain name at the places: “IP” and “INTERNET”. If this is the case, contact your provider or the Internet for more information.)

The make a GPRS connection one of the following commands should be used as ISP phone number:

`ATD*99#` or `ATD*99***1#`

Which of these commands should be used depends on the type of mobile phone you’re using.
The table below shows which AT-commands should be used to establish a GPRS connection on with different types of mobile phones. (Note that GPRS hasn’t been tested on most types of the mobile phones that are listed below, see Appendix F)

Sony-Ericsson T68i AT+CGDCONT=1, ”IP”,”INTERNET” (AT+CGQREQ=1,0,0,0,0,0)* ATD*99***1#	***Motorola Timeport 280*** (AT&F&K5)* AT+CGDCONT=1,”IP”,”INTERNET” ATD*99#
Motorola TP260 (ATQ0V1E1S0=0&C1&D2)* AT+CGDCONT=1, ”IP”,”INTERNET” ATDT*99***1#	***Nokia 6310*** (AT&F&K4)* AT+CGDCONT=1,”IP”,”INTERNET” ATD*99# or ATD*99***1#
Ericsson T65, T68, R520m AT+CGDCONT=1,”IP”,”INTERNET” ATD*99***1#	***Siemens (S45):*** (AT&F)* AT+CGDCONT=1,”IP”,”INTERNET” ATD*99***1#
Motorola P280 (AT&FE1Q0) AT+CGDCONT=1, ”IP”,”INTERNET” (AT+CGQMIN=1,0,0,3,0,0 OK)* (AT+CGQREQ=1,0,0,3,0,0 OK)* ATD*99#	***Nokia 7650, 8910*** AT+CGDCONT=1,”IP”,”INTERNET” ATD*99***1#

* The modem commands in the table above that are place between brackets may possibly be left out.

In the following example demonstrates how a GPRS connection can be make with a Nokia mobile phone:

```
#include "lib.h"       // Declares some functions and addresses of the terminal
#include "internet.h"  // This is the header file of the TCP/IP library CQMVxxxx
#include <stdio.h>
#include <string.h>

struct isp_config isp;
struct dns_config dns;

void main(void)
{
        int error;
        systemsetting("K8");      //set baudrate to 38400bps

        strcpy (isp.phonenumber, "ATD*99***1#");
        strcpy (isp.user ,"");              //most providers do not demand the use of a ISP user
        strcpy (isp.pass, "");              //name of password, but this depends on the provider.

        isp.com = 1;              //use IrDA, so use library version CQMVxxxx
        isp.abort_key = dns.abort_key = CLR_KEY;

        // connect to ISP
        TCPIP_init(1,0);
        setfont(SMALL_FONT, NULL);

        error=connect_additional_ATcommand(&isp,(char *)("AT+CGDCONT=1,\"IP\",\"INTERNET\""));

        if (error != DIALUP_SUCCESS)
```

```
            printf("\fError: %d",error);
        else if(error = get_local_IP_address(&dns, 30))
            printf("\fError: %d",error);
        else
        {
            printf("\fGPRS connect:  OK");
            printf("\n\nAssigned IP: \n %d.%d.%d.%d", dns.IP1, dns.IP2, dns.IP3, dns.IP4);
        }

        disconnect(&isp)        ;

        while(!kbhit())    idle();
}
```

Sending email to a remote SMTP server using GPRS (also applies to the Ethernet box!)

After establishing the GPRS connection it might happen that it's not possible to connect to a remote SMTP server. This is because the SMTP server demands that users authenticate theirself. Authentication normally happens during the establishing of the PPP connection with the ISP server of this email account. Because a username and password aren't used when establishing a GPRS connection, this results in an authenticating problem.
On many SMTP servers this problem can be solved as follows. Use **'sendmail_authenticated'** or before connecting to the SMTP server, a connection should first be made with the POP-server of the used email account to let the PHL terminal authenticate itself at the mail-server. To do this use the getmail(&pop,POP_INIT)-function with the IP-address of the POP-server and a valid username and password of the email account.

APPENDIX E: Manually establish a TCP/IP connection

Normally a TCP/IP connection is established using one the following three functions of this library.

- connect (See chapter 3.2)
- connect_additional_Atcommand (See chapter 3.3)
- connect_ user_specified_Atcommand (See chapter 3.4)

However in some special situations these functions aren't sufficient to establish a working TCP/IP connection. If this is the case the TCP/IP connection can also be established manually.

For instance, when you need to establish a GPRS connection with a GPRS/GSM modem, additional initializations sometimes need to be executed to register the modem to the mobile network. Because these initializations consist of multiple steps they can't be executed with the standard connect-functions and therefor the establishing of the TCP/IP connection will need to be performed manually.

When establishing a TCP/IP connection manually the following 5 steps need to be followed:

1. Open the serial port of the PHLxxxx that you want to connect with, using the function 'comopen'

COM-port	Description
COM0	Direct cable
COM1	Cradle / IrDA dongle
COM2	Cradle

2. Initialize the modem with (AT-)commands and check for the desired response.

The following two functions of this library can be usefull to do this:
- transmit AT-commands to the modem (See chapter '3.x.x. sendstring)
- wait for a certain response of the modem (See chapter '3.x.x. checkstring)

`void sendstring(const char *string)`	Transmit as string to the modem.
`int checkstring(const char *string, int timeout,unsigned char abort_key)`	Waits till the desired response is received from the modem or the time out has passed. (Also automatically checks for standard modem error messages like: NO CARRIER and NO DIALTONE)

Example:

```
sendstring("ATZ\r");                          // Send Reset-command

if( checkstring("OK", 100, Q1_KEY) != OK )    // Wait for "OK" for 2 seconds.
       return ERROR;
```

3. Establish the connection with the ISP or GPRS provider

Transmit the dial or connect command to the modem, which also can be done using the sendstring( )-function.

To connect to an ISP server most modems use the AT-command:

```
ATDT<telephone number>
```

To connect to a GPRS server most mobile modems use the AT-command:

```
ATD*99***1#       or          ATD*99#
```

4. Wait for the result of the ATD-command

Wait for the desired response of the modem, which can be done using the checkstring( )-function. The normal response of a modem after its established a connection is: 'CONNECT'.

Example:

```
//Wait for 'CONNECT'
if (error=checkstring("CONNECT", 3000, Q1_KEY) != OK)
      return ERROR;
```

5. Register the connection to the TCP/IP library

After the connection has been successfully established it must be registered to the TCP/IP library by using the function 'register_manual_connection' (See chapter 3.5).

Example:

```
struct isp_config isp;
int result;

isp.phonenumber[0] = '\0';       //Not necessary because your already connected
isp.user[0] = '\0';              //Most GPRS providers do not demand the
isp.pass[0] = '\0';              //use of a ISP username & password

isp.com = COM4;                  //Specifies COM-port that the current
                                 //connection is using

isp.abort_key = Q1_KEY           //Specifies the abort-key

register_manual_connection(&isp); //Register the manually made connection
                                 //to the TCP/IP library
```

```
#include <stdio.h>
#include <string.h>
#include "lib.h"          // Declares some functions and addresses of the terminal
#include "internet.h"     // This is the header file of the TCP/IP library

struct isp_config isp;
struct dns_config dns;

int check_network_registration(int timeout, unsigned char abort_key);

void main(void)
{
    int error, i, connected;

    cursor(NOWRAP);           // Don't use auto-wrap
    TCPIP_init(1,0);          // Initialize TCP/IP library
    systemsetting("SZ");      // Baudrate: 115200bps

    for(;;)
    {   setfont(LARGE_FONT, NULL);
        printf("\f\nConnecting...");
        connected = 0;

        if(comopen(COM2)==OK) //Open comport of cradle
        {   sendstring("ATZ\r");                  //Reset Modem

            if( checkstring("OK", 100, Q1_KEY) == OK )     //Wait for "OK"
            {
                sendstring("ATE0\r");          //Set Echo off

                if( checkstring("OK", 100, Q1_KEY) == OK ) //Wait for "OK"
                {   //Check if modem is registered to mobile network
                    error=check_network_registration(50, Q1_KEY);

                    if(error > 0 && error != 1)
                    {   //Sending the WRONG PIN-code can cause your SIM to be blocked!
                        //Replace '0000' by the correct PIN-code!
                        sendstring("AT+CPIN=0000\r");

                        if((error=checkstring("OK", 100, Q1_KEY)) == OK)//Wait for "OK"
                        {
                            sendstring("AT+CREG=1\r");   //Register to network attempt
                            error=checkstring("OK", 100, Q1_KEY); //Wait for "OK"
                        }
                        if(error == OK)
                        {   i = 0;

                            //Wait till registered to network
                            while((error=check_network_registration(50, Q1_KEY)) != 1)
                            {
                                if(error < 0) break;     //Error

                                while(!endtimer()) idle(); //Wait till time out passed

                                if(++i >= 30) break;       //Time out after 30 seconds
                            }
                        }
                    }
                    if(error == 1)        //Registered to mobile network
                    {   //Initialize GPRS settings
                        sendstring("AT+CGDCONT=1,\"IP\",\"internet\"\r");

                        if( checkstring("OK", 100, Q1_KEY) == OK ) //Wait for "OK"
                        {
                            sendstring("ATD*99***1#\r");         //Connect to GPRS

                            //Wait for CONNECT
                            if (error=checkstring("CONNECT", 1500, Q1_KEY) == OK)
                                connected = 1;           //Connected to GPRS!
                        }
                    }
                }
            }
        }
```

```
        if(connected)
        {
            isp.phonenumber[0] = '\0'; // Not necessary because your already connected
            isp.user[0] = '\0';        // Most GPRS providers do not demand
            isp.pass[0] = '\0';        // the use of a ISP username & password

            isp.com = COM2;            //Specify the COM-port of the connection

            isp.abort_key = dns.abort_key = Q1_KEY;        //Specify the abort-key

            setfont(SMALL_FONT, NULL);
            printf("\f");

            //Register manually made GPRS connection to TCP/IP library
            error=connected_manually(&isp);

            if (error != DIALUP_SUCCESS)
                printf("\fError: %d",error);
            else if(error = get_local_IP_address(&dns, 30))
                printf("\fError: %d",error);
            else
            {
                printf("\fConnect: OK");
                printf("\nAssigned IP:\n %d.%d.%d.%d", dns.IP1,dns.IP2,dns.IP3,dns.IP4);
            }

            disconnect(&isp); //Close the connection and the COM-port
        }
        else
        {
            printf("\nFailed");
            comclose(COM4);
        }

        while(!kbhit()) idle();
    }
}

int check_network_registration(int timeout, unsigned char abort_key)
{
    int c;

    sendstring("AT+CREG?\r");

    if( (c=checkstring("CREG", timeout, abort_key)) != OK )
        return c;

    starttimer(timeout);  // After Time Out Return negative...

    while(1)
    {
        if(endtimer())
            return(ERR_NORESPONSE);

        if( (c=getcom(1)) != EOF )
        {
            if(c == ',')          //   CREG: <mode>,c
            {
                while(1)
                {
                    if(endtimer())
                        return(ERR_NORESPONSE);

                    //1 = connected, 0 and 2 not connected
                    if( (c=getcom(1)) >= '0' && c<='2')
                        return (c - '0');
                }
            }
        }
    }
}
```

APPENDIX F: Test Results (TCP/IP with an extern modem)

This appendix includes all the successfully tested ISP, FTP, POP, SMTP and DNS servers, using a serial connection with an extern modem

Dialup servers / ISP's (including their primary and secondary DNS servers)

☑ ZONNET	:	06760 - 75030
☑ XS4ALL	:	020 –5350535
☑ RAKETNET	:	06760 - 50300
☑ 12MOVE	:	06760 - 77777
☑ WANADOO	:	06760 - 22205
☑ FREE ACCESS	:	023 - 7430000
☑ DIRECT internet	:	06760 - 12321
☑ French Telecom	:	0033 - 860000000

FTP servers

☑ ZONNET	:	ftp.home.zonnet.nl	[062.058.050.014]
☑ 12MOVE	:	home.12move.nl	[062.035.014.015]
☑ WANADOO	:	home.wanadoo.nl	[194.134.035.012]
☑ IFRANCE	:	ftp.ifrance.com	[062.039.122.012]
☑ RAKETNET	:	ftp.raketnet.nl	[213.197.030.200]
☑ XS4ALL	:	reflectix.xs4all.nl	[194.109.006.026]
☑ OVfiets.nl	:	ftp.OVfiets.nl	[212.204.242.076]
☑ FREE.fr	:	ftpperso1-2.free.fr	[213.228.000.170]
☑ NMPP.fr.ft	:	paramdiffuseurs.nmpp.fr	[172.017.000.004]

POP servers

☑ ZONNET	:	pop3.zonnet.nl	[062.058.050.010]
☑ 12MOVE	:	pop3.12move.nl	[195.241.076.041]
☑ WANADOO	:	pop.wanadoo.nl	[194.134.035.136]
☑ IFRANCE	:	pop3.ifrance.com	[062.039.122.017]
☑ RAKETNET	:	pop3.raketnet.nl	[213.197.030.201]
☑ XS4ALL	:	pop.xs4all.nl	[194.109.006.055]

SMTP servers

☑ ZONNET	:	smtp.zonnet.nl	[062.058.050.046]
☑ 12MOVE	:	smtp.12move.nl	[195.241.076.179]
☑ WANADOO	:	smtp.wanadoo.nl	[194.134.035.138]
☑ IFRANCE	:	smtp.ifrance.com	[062.039.122.20]
☑ RAKETNET	:	smtp.raketnet.nl	[213.197.030.201]
☑ XS4ALL	:	smtp.xs4all.nl	[194.109.006.051]
☑ OPTICON	:	mail.opticon.nl	[080.073.128.038]

Equipment used

Terminal	:	PHL1300, PHL1700, PHL2700 with cable and cradle
Modem	:	Dynalink 1456E-R2 external 56k modem
		Tornado V90 external 56k modem
		Siemens MC35I GSM/GPRS modem
		Opticon Ethernet box
Adapter	:	25-pins male gender changer
		25-pins NULL modem
		2x '9-pins male to 25-pins female' connector

TCP/IP Library : CMMV0306 and CQMV0306

APPENDIX G: Test Results (TCP/IP with an IrDA connection)

This appendix includes all the successfully tested ISP, FTP, POP, SMTP and DNS servers, using a wireless IrDA connection with a mobile phone.

Dialup servers / ISP's (including their primary and secondary DNS servers)

☑ XS4ALL	:	020 –5350535
☑ RAKETNET	:	06760 - 50300
☑ 12MOVE	:	06760 - 77777
☑ WANADOO	:	06760 - 22205
☑ FREE ACCESS	:	023 - 7430000
☑ DIRECT internet	:	06760 – 12321
☑ ZONNET	:	06760 - 75030

FTP servers

☑ 12MOVE	:	home.12move.nl	[062.035.014.015]
☑ WANADOO	:	home.wanadoo.nl	[194.134.035.012]
☑ RAKETNET	:	ftp.raketnet.nl	[213.197.030.200]
☑ OVfiets.nl	:	ftp.OVfiets.nl	[212.204.242.076]
☑ ZONNET	:	ftp.home.zonnet.nl	[062.058.050.014]

POP servers

☑ 12MOVE	:	pop3.12move.nl	[195.241.076.041]
☑ WANADOO	:	pop.wanadoo.nl	[194.134.035.136]
☑ RAKETNET	:	pop3.raketnet.nl	[213.197.030.201]
☑ XS4ALL	:	pop.xs4all.nl	[194.109.006.055]
☑ ZONNET	:	pop3.zonnet.nl	[062.058.050.010]

SMTP servers

☑ 12MOVE	:	smtp.12move.nl	[195.241.076.179]
☑ WANADOO	:	smtp.wanadoo.nl	[194.134.035.138]
☑ RAKETNET	:	smtp.raketnet.nl	[213.197.030.201]
☑ XS4ALL	:	smtp.xs4all.nl	[194.109.006.051]
☑ ZONNET	:	smtp.zonnet.nl	[062.058.050.046]

Equipment used

Terminal	:	PHL1300 and PHL2700 with an IrDA connection
Mobile phone	:	Nokia 6210
		Ericsson T68
		Nokia 7650
		Nokia 8910
TCP/IP Library	:	CQMV0306
IrDA library	:	COWV0105

Notes:

- **The French servers of Free.fr and French Telecom haven't been tested with a wireless connection**

APPENDIX H: NTP Time servers

The NTP implementation in this TCP/IP library has been successfully tested with over 250 NTP servers around the world.

A list of NTP servers can be found on the following URL (Note that not all the servers on this website will wortk with this library!). http://ntp.isc.org/bin/view/Servers/WebHome