

# TCP-IP Library for the OPH100x and H13



Application Development Manual  
Library version:  
XMAV0306 / XPAV0306  
©2008, Opticon Sensors Europe BV

## Table of Contents

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| 1. Introduction.....                                                  | 3  |
| 1.1. This manual.....                                                 | 4  |
| 2. Using the TCP/IP Library for the OPH100x and H13.....              | 5  |
| 2.1. Selection and setup of the modem.....                            | 5  |
| 2.2. Connecting the modem.....                                        | 5  |
| 2.3. Making a GSM / GPRS connection with a mobile phone.....          | 5  |
| 2.4. Making an Ethernet connection with the Opticon Ethernet box..... | 5  |
| 2.5. How to include the library in your application software.....     | 6  |
| 2.6. Parameters used by the library functions.....                    | 6  |
| 2.7. Calling the library functions.....                               | 8  |
| 3. TCP/IP Library Functions.....                                      | 9  |
| 3.1. TCPIP_init.....                                                  | 9  |
| 3.2. connect.....                                                     | 10 |
| 3.3. connect_additional_ATcommand.....                                | 11 |
| 3.4. connect_user_specified_ATcommand.....                            | 11 |
| 3.5. register_manual_connection.....                                  | 12 |
| 3.6. disconnect.....                                                  | 13 |
| 3.7. ftp_command.....                                                 | 14 |
| 3.8. ftp_connected.....                                               | 18 |
| 3.9. getmail.....                                                     | 19 |
| 3.10. sendmail.....                                                   | 23 |
| 3.11. sendmail_authenticated.....                                     | 24 |
| 3.12. get_host_by_name.....                                           | 26 |
| 3.13. get_host_by_addr.....                                           | 26 |
| 3.14. get_local_IP_address.....                                       | 28 |
| 3.15. get_ISP_server_IP.....                                          | 28 |
| 3.16. get_primary_DNS_server_IP.....                                  | 29 |
| 3.17. get_secondary_DNS_server_IP.....                                | 29 |
| 3.18. set_primary_DNS_server_IP / set_secondary_DNS_server_IP.....    | 31 |
| 3.19. get_NTP_time.....                                               | 32 |
| 3.20. change_FTP_remote_port_nr.....                                  | 34 |
| 3.21. change_SMTP_remote_port_nr.....                                 | 34 |
| 3.22. change_POP_remote_port_nr.....                                  | 34 |
| 3.23. tcpip_lib_version.....                                          | 35 |
| APPENDIX A: Error codes.....                                          | 36 |
| APPENDIX B: Serial connection from terminal to modem.....             | 37 |
| APPENDIX C: AT-command diagram of the connect-functions.....          | 38 |
| APPENDIX D: Making a GPRS connection with a mobile phone.....         | 39 |
| APPENDIX E: Manually establish a TCP/IP connection.....               | 41 |
| APPENDIX F: Test Results.....                                         | 45 |
| APPENDIX G: NTP Time servers.....                                     | 46 |

## 1. Introduction

The TCP/IP Library for Opticons Handheld Terminals OPH100x and H13 provide the possibility to exchange data over the Internet.

Salient features:

- Dial-in using an Analog modem and GSM or GPRS modem/telephone
- Connect to an Ethernet network (with or without Internet access like DSL) using the Opticon Ethernet converter box.
- A standard PPP dial-up account can be used to access the Internet.
- 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 of 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 the OPH100x and H13 in a C language application for the following data collection terminals:

Library versions:           **XMAV0306 (OPH-1003) and XPAV0306 (OPH-1004 & H13)**

## **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 the OPH100x and H13 in application software. It does not describe in general how to develop applications for Opticon barcode terminals. For this, we refer to the Opticon Programming manual that is included in Opticon's "C-development kit for handheld barcode terminals".

The TCP/IP Library for the OPH100x and H13 is copyrighted © 2008 Opticon Sensors Europe BV. It is partly based on the uIP TCP/IP protocol stack developed by Adam Dunkels. The appropriate copyright notice is shown below (note: the inclusion in this library is considered to be redistribution in binary form).

Copyright (c) 2001, Adam Dunkels.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:  
This product includes software developed by Adam Dunkels.
4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## **2. Using the TCP/IP Library for the OPH100x and H13**

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 cradle and the terminal, only half-duplex operation is permitted. Some modems implement the 'echo' command 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 cradles of the OPH100x and H13 do not support hardware handshaking. Configure the modem so that it doesn't use hardware handshaking.
- Always use the fastest baud rate possible. Normally that's 115200bps. This increases the connection speed and the reliability of the data connection.

### ***2.2. Connecting the modem***

The RS232 connector on the cradles of the OPH100x and H13 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).

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) RS232 cable with 9-pin male (9M) connectors at both ends.
- If the modem has only a 25F connector, you can use the above combinations in combination with a 9F-to-25M adapter.

### ***2.3. Making a GSM / 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 subscription 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.

### ***2.4. 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.

## 2.5. How to include the library in your application software

To build an application, see the Opticon programming manual for handheld terminals.

The functions of the TCP/IP library can be linked by adding the library to the linker flags (look for 'LDFLAGS' in your makefile).

For the **OPH1003** add **-IXMAV0306** to the linker flags to add the library file **libXMAV0306.a'**

For the **OPH1004** and **H14** add **-IXPAV0306** to the linker flags to add the library file **libXPAV0306.a'**

### Example for the OPH1003:

```
LDFLAGS = -nostartfiles -nostdlib --entry 0x00000000 -Wl,  
          -Map=$(TARGET).map,--cref -T$(SUBMDL).ld  
          -L. -loph100x -lXMAV0306 -lc -lm -lgcc
```

### Example for the OPH1004 & H13:

```
LDFLAGS = -nostartfiles -nostdlib --entry 0x00000000 -Wl,  
          -Map=$(TARGET).map,--cref -T$(SUBMDL).ld  
          -L. -lXPAV0305 -lc -lm -lgcc
```

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

## 2.6. Parameters used by the library functions

The TCP/IP Library for the OPH100x and H13 provides functions to:

- Dial-in using an Analog modem and GSM or GPRS modem/telephone
- Connect to an Ethernet network (with or without Internet access like DSL) using the Opticon Ethernet converter 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`
- `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 or a progress bar 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 to 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 initialization 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 **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
```

## ***2.7. 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 time-out.



## 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 and if all received and sent data is stored on the terminal. See also chapter 2.5.

**Syntax**                void TCPIP\_init(int internet\_status, int debug\_status);

|                  |                            |   |                                                                                                                                                                                                            |
|------------------|----------------------------|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Arguments</b> | <i>int internet_status</i> | 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 |
|                  | <i>int debug_status</i>    | 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 2 files on the terminal.                                                         |
|                  |                            | 3 | A log file will be generated of all received and sent data that can be analysed using the program <b>Ethereal / Wireshark</b> . (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 analysing 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

**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, and starts a PPP connection. After a 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 that contains the password to authenticate the user to the ISP.

*com*

Specifies the serial port of the terminal that the modem is connected on:

|      |               |
|------|---------------|
| COM1 | Infrared Port |
| COM2 | 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 (= 0), 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*

|                    |                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | See 3.2 <i>connect</i> , the only difference with the <i>connect( )</i> -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.                                                                                      |
| <b>Syntax</b>      | <code>int connect_additional_ATcommand(struct isp_config *isp, char *AT_Command)</code>                                                                                                                                                                                                                                    |
| <b>Arguments</b>   | <code>struct isp_config *isp</code> , see 3.2 <i>connect</i> .<br><br><i>char * AT_Command</i><br>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.                                                  |
| <b>Returns</b>     | This function returns an error code (see appendix A)                                                                                                                                                                                                                                                                       |
| <b>Remarks</b>     | If this function returns any other value than <code>DIALUP_SUCCESS</code> , the PPP connection was not established.<br><br>More information about the modem (AT-)commands that are being send to the modem or mobile phone when calling this <i>connect_additional_ATcommand( )</i> -function, can be found in Appendix C. |

### 3.4. *connect\_user\_specified\_ATcommand*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | See 3.2 <i>connect</i> , the only difference with the <i>connect( )</i> -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.                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Syntax</b>      | <code>int connect_user_specified_ATcommand(struct isp_config *isp, char *AT_Command)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>   | <code>struct isp_config *isp</code> , see 3.2 <i>connect</i> .<br><br><i>char * AT_Command</i><br>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.                                                                                                                                                                                                                                                                                                                                                  |
| <b>Returns</b>     | This function returns an error code (see appendix A)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Remarks</b>     | If this function returns any other value than <code>DIALUP_SUCCESS</code> , the PPP connection was not established.<br><br>The modem reset string (ATZ) and the dialing command (i.e. <code>ATDT&lt;phonenumber&gt;</code> ) will always be send and can't be overruled.<br><br>More information about the modem (AT-)commands that are normally send to the modem or mobile phone when calling the <i>connect( )</i> -function or the <i>connect_additional_ATcommand( )</i> -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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |      |               |      |        |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|---------------|------|--------|
| <b>Description</b> | This function allows the user to manually connect the OPH100x or H13 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |      |               |      |        |
| <b>Syntax</b>      | int register_manual_connection(struct isp_config *isp)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |      |               |      |        |
| <b>Arguments</b>   | <p>struct isp_config *<i>isp</i></p> <pre>struct isp_config{     char phonenumber[20];     char user[40];     char pass[40];     char com;     char abort_key; };</pre> <p><i>phonenumber</i><br/>&lt;not used by library&gt;</p> <p><i>user</i><br/>Null-terminated string containing the user name that corresponds to the ISP account</p> <p><i>pass</i><br/>Null-terminated string that contains the password to authenticate the user to the ISP.</p> <p><i>com</i><br/>Specifies the serial port of the terminal that the modem is connected on:</p> <table><tr><td>COM1</td><td>Infrared Port</td></tr><tr><td>COM2</td><td>Cradle</td></tr></table> <p><i>abort_key</i><br/>Specifies the key that can be used to abort the function.</p> | COM1 | Infrared Port | COM2 | Cradle |
| COM1               | Infrared Port                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |      |               |      |        |
| COM2               | Cradle                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |      |               |      |        |
| <b>Returns</b>     | This function returns an error code (see appendix A, same as 'connect( )' )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |      |               |      |        |
| <b>Remarks</b>     | <p>If this function returns any other value than DIALUP_SUCCESS (=0), the connection was not successfully registered.</p> <p>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.</p> <p>See Appendix E, for detailed information on how this function can be used.</p>                                                                                                                                                                                                           |      |               |      |        |

### **3.6. *disconnect***

|                    |                                                                                                          |                    |
|--------------------|----------------------------------------------------------------------------------------------------------|--------------------|
| <b>Description</b> | Terminates the PPP connection, hangs up the modem, mobile phone or Ethernet box and closes the COM port. |                    |
| <b>Syntax</b>      | int disconnect (struct isp_config *isp)                                                                  |                    |
| <b>Arguments</b>   | struct isp_config *isp                                                                                   | See 3.2. connect() |
| <b>Returns</b>     | 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*

Command the specifies 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.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>remote_file</i> | <p><b>FTP_PUT_PASSIVE</b>: 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).</p> <p><b>FTP_GET_PASSIVE</b>: specifies the name of the downloaded file.</p> <p><b>FTP_LIST_PASSIVE</b>: specifies a file name mask (e.g. *.txt). If the string is empty, all files in the directory are listed.</p> <p><b>FTP_CWD</b>: specifies the path of the new work directory (this may also be the parent directory by using '..')</p> <p><b>FTP_REN</b>: specifies the file to be renamed.</p> <p><b>FTP_DEL</b>: specifies the file to be deleted.</p> <p><b>FTP_INIT, FTP_QUIT</b>: this parameter is ignored.</p> |
| <i>append</i>      | <p><b>FTP_GET_PASSIVE</b>: specifies if received data is to be appended to the specified local file or not.</p> <p><b>FTP_PUT_PASSIVE</b>: specifies if the FTP server is to append the transmitted data to the specified remote file. (1 = Append, 0 = Overwrite)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>abort_key</i>   | 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 <stdio.h>
#include <string.h>
#include "lib.h"
#include "internet.h" // header file of the TCP/IP library

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCP_IP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030"); // change this value!
    strcpy(isp.user, "oph100x"); // change this value!
    strcpy(isp.pass, "test"); // change this value!
    isp.com = COM2; // Modem is connected with the cradle
    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, "user");
    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')      // upload file to FTP
{
    strcpy (ftp.local_file,"data.txt");
    strcpy (ftp.remote_file,"barcodes.txt");

    if ((error= ftp_command(&ftp, FTP_PUT))!=FTP_SUCCESS)
    {
        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')      // Download file from FTP
{
    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 4 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 server after the message was successfully received. (1 = delete message, 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 <stdio.h>
#include <string.h>
#include "lib.h"
#include "internet.h"    // header file of the TCP/IP library

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;

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030");    // change this value!
    strcpy(isp.user, "oph100x");              // change this value!
    strcpy(isp.pass, "test");                 // change this value!
    isp.com = COM2;                           // Modem is connected with the cradle
    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, "user123");
    strcpy (pop.pass, "test");
    strcpy (pop.local_file, "database.txt");

    pop.incl_hdrs = FALSE;                  //don't include header to file
    pop.delete_msg = TRUE;                  //delete message from server after receiving
    pop.abort_key = CLR_KEY;                // The CLR key will be used to abort

    if ( (error=connect(&isp)) < 0 )
    {
        switch(error)
        {
            case ERR_USER_ABORTED:
                printf("connection error");
                break;
            case ERR_POLLED:
                printf("ISP not responding");
                break;
            default:
                printf("\nError #%d",error);
        }
    }
    else if ( (error=getmail(&pop,POP_INIT)) < 0 )
    {
        switch(error) // Only two errors are evaluated in this example
        {
            case ERR_USER_ABORTED:
                printf("User aborted by pressing key");
                break;
            case ERR_POLLED:
                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 in this example
        {
            case ERR_USER_ABORTED:
                printf("User aborted by pressing key");
                break;
            case ERR_POLLED:
                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 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 8<sup>th</sup> 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. Therefore 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)

|                  |                          |                                                                       |
|------------------|--------------------------|-----------------------------------------------------------------------|
| <b>Arguments</b> | struct mail_config *mail | (see 3.10 Sendmail)                                                   |
|                  | username                 | Pointer to SMTP Username<br>(if NULL, no authentication is performed) |
|                  | password                 | Pointer to SMTP Password<br>(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 <stdio.h>
#include <string.h>
#include "lib.h"
#include "internet.h"          // header file of the TCP/IP library

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030");    // change this value!
    strcpy(isp.user, "oph100x");              // change this value!
    strcpy(isp.pass, "test");                 // change this value!
    isp.com = COM2;                           // Modem is connected with the cradle
    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, "oph100x@zonnet.nl");    //change this value
    strcpy(mail.recipient, "database@opticon.com"); //change this value
    strcpy(mail.reply, " database@opticon.com"); //change this value

    strcpy(mail.subject, "testing OPH100x 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)) < 0 )
    {
        switch(error){
            case ERR_USER_ABORTED:
                printf("\nerror opening COM port");
                break;
            default:
                printf("\nError #%d", error);
        }
    }
    else if ( (error=sendmail(&mail) < 0 ) )
    {
        switch(error)          // This example only handles 3 errors
        {
            case ERR_USER_ABORTED:
                printf("User aborted by pressing key");
                break;
            case ERR_NO_SUCH_FILE:
                printf("File does not exist");
                break;
            case ERR_POLLED:
                printf("Application polled too many times");
                break;
            default:
                printf("\nOther Error");
        }
    }
    disconnect(&isp);
    while(!kbhit())
        idle();
}
```

### 3.12.get\_host\_by\_name

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns the IP address of the specified domain name using the Domain Name System (DNS) protocol.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Syntax</b>      | <code>int get_host_by_name(struct dns_config *dns)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Arguments</b>   | <p><code>struct dns_config *dns</code></p> <p><code>struct dns_config{</code><br/>    <code>char domain_address[80];</code><br/>    <code>unsigned short IP1, IP2, IP3, IP4;</code><br/>    <code>char abort_key;</code><br/><code>};</code></p> <p><i>domain_address</i><br/>Contains the domain name of which the IP address should be retrieved</p> <p><i>IP1.. IP4</i><br/>The four bytes in which the IP address of the specified domain name is returned</p> <p><i>abort_key</i><br/>Specifies the key for aborting the operation.</p> |
| <b>Returns</b>     | The function returns an error code (see appendix A)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Remarks</b>     | If the function returns the error: -37 (ERR_DNS_NO_SERVERS_FOUND), use the functions: <code>set_primary(/secondary)_DNS_server_IP</code> 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns the domain name of the specified IP address using the Domain Name System (DNS) protocol.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Syntax</b>      | <code>int get_host_by_addr(struct dns_config *dns)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>   | <p><code>struct dns_config *dns</code></p> <p><code>struct dns_config {</code><br/>    <code>char domain_address[80];</code><br/>    <code>unsigned short IP1, IP2, IP3, IP4;</code><br/>    <code>char abort_key;</code><br/><code>};</code></p> <p><i>domain_address</i><br/>String in which the domain name of specified IP address is returned</p> <p><i>IP1.. IP4</i><br/>Contain the IP address of the domain name that should be retrieved.</p> <p><i>abort_key</i><br/>Specifies the key for aborting the operation.</p> |
| <b>Returns</b>     | The function returns an error code (see appendix A)                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Remarks</b>     | See 3.12 <code>get_host_by_name</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Example: DNS (get\_host\_by\_name/address)

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

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030"); // change this value!
    strcpy(isp.user, "oph100x");           // change this value!
    strcpy(isp.pass, "test");              // change this value!
    isp.com = COM2;                        // Modem is connected with the cradle
    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 ERR_USER_ABORTED:
                printf("\nerror opening COM port");
                break;
            case ERR_NORESPONSE:
                printf("\nmodem not connected or not responding");
                break;
            default:
                printf("\nError #%d",error);
        }
    }
    else if ((error=get_host_by_name(&dns))< 0) // Retrieve IP-address from DNS-server
    {
        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;

        // Retrieve domain name from DNS-server
        if ((error=get_host_by_addr(&dns)) < 0)
            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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns the IP address of the primary DNS server that is retrieved from the ISP                                                                                                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>      | int get_primary_DNS_server_IP(struct dns_config *dns, int timeout)                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <div>struct dns_config *dns</div> <div>struct dns_config{<br/>    char domain_address[80];<br/>    unsigned short IP1, IP2, IP3, IP4;<br/>    char abort_key;<br/>};</div> <div><i>IP1.. IP4</i>        The four bytes in which the IP address of the primary DNS server is returned</div> <div><i>abort_key</i>        Specifies the key for aborting the operation.</div> <div><i>timeout</i>          Time out in seconds</div> |
| <b>Returns</b>     | This function returns an error code (see appendix A)                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>     | 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns the IP address of the secondary DNS server that is retrieved from the ISP                                                                                                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>      | int get_secondary_DNS_server_IP(struct dns_config *dns, int timeout)                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <div>struct dns_config *dns</div> <div>struct dns_config{<br/>    char domain_address[80];<br/>    unsigned short IP1, IP2, IP3, IP4;<br/>    char abort_key;<br/>};</div> <div><i>IP1.. IP4</i>        The four bytes in which the IP address of the secondary DNS server is returned</div> <div><i>abort_key</i>        Specifies the key for aborting the operation.</div> <div><i>timeout</i>          Time out in seconds</div> |
| <b>Returns</b>     | This function returns an error code (see appendix A)                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Remarks</b>     | 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 <stdio.h>
#include <string.h>
#include "lib.h"
#include "internet.h"          // header file of the TCP/IP library

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030"); // change this value!
    strcpy(isp.user, "oph100x");           // change this value!
    strcpy(isp.pass, "test");              // change this value!
    isp.com = COM2;                        // Modem is connected with the cradle
    dns.abort_key = CLR_KEY;               //The CLR key will be used to abort

    dns_choice = 1;                       //1: local IP, 2: ISP IP, 3: Prim. DNS, 4: Secnd. DNS

    printf("Starting...");

    if ( (error=connect(&isp)) < 0 )
    {
        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 sec 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 4 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 terminal. (Off course, 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** The 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 tested NTP servers can be found. (Note that this list will start to contain many non functioning servers in the near future.)*



### Example: NTP (get\_NTP\_time)

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

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

void main (void)
{
    int error=0;

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

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    // Internet Service Provider Settings
    strcpy(isp.phonenumber, "0676075030");    // change this value!
    strcpy(isp.user, "oph100x");              // change this value!
    strcpy(isp.pass, "test");                 // change this value!
    isp.com = COM2;                           // Modem is connected with the cradle
    ntp.abort_key = CLR_KEY;                  //The CLR key will be used to abort

    ntp.IP1 = 83;                             // IP from nl.pool.ntp.org (83.83.82.105)
    ntp.IP2 = 83;                             // IP-addresses can be changed without prior notice
    ntp.IP3 = 82;                             // for this reason always use DNS (getHostbyName) to
    ntp.IP4 = 105;                            // retrieve the IP-addresses of the used NTP servers

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

        while(1)
        {
            gettime(&t);    //get current local time
            getdate(&d);    //get current date

            ntp.date = &d; //set date and time parameters
            ntp.time = &t;

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

                setdate(&d);
                settime(&t);
            }
        }
    }
    disconnect(&isp);    //hang up

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

### **3.20.change\_FTP\_remote\_port\_nr**

|                    |                                                                                                                                                                                                                                                                                                                          |                                                   |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| <b>Description</b> | Changes the port number of the remote FTP server to the specified port number                                                                                                                                                                                                                                            |                                                   |
| <b>Syntax</b>      | int change_FTP_remote_port_nr(unsigned short port_nr)                                                                                                                                                                                                                                                                    |                                                   |
| <b>Arguments</b>   | <i>unsigned short port_nr</i><br>Specifies the port number of the remote FTP server.<br>Port numbers 25(SMTP), 53(DNS), 110(POP), 123(NTP) are reserved port numbers and may therefor not be used.                                                                                                                       |                                                   |
| <b>Returns</b>     | OK<br>ERR_INVALID_PORT_NUMBER                                                                                                                                                                                                                                                                                            | On success<br>If invalid port number is specified |
| <b>Remarks</b>     | This function should only be used if the remote FTP server does not 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**

|                    |                                                                                                                                                                                                                                                                                                                         |  |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Description</b> | Changes the port number of the remote SMTP server to the specified port number                                                                                                                                                                                                                                          |  |
| <b>Syntax</b>      | int change_SMTP_remote_port_nr(unsigned short port_nr)                                                                                                                                                                                                                                                                  |  |
| <b>Arguments</b>   | <i>unsigned short port_nr</i><br>Specifies the port number of the remote SMTP server.<br>Port numbers 21(FTP), 53(DNS), 110(POP), 123(NTP) are reserved port numbers and may therefor not be used.                                                                                                                      |  |
| <b>Returns</b>     | See 3.20 'change_FTP_remote_port_nr'                                                                                                                                                                                                                                                                                    |  |
| <b>Remarks</b>     | 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**

|                    |                                                                                                                                                                                                                                                                                                                        |  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Description</b> | Changes the port number of the remote POP server, to the specified port number                                                                                                                                                                                                                                         |  |
| <b>Syntax</b>      | int change_POP_remote_port_nr(unsigned short port_nr)                                                                                                                                                                                                                                                                  |  |
| <b>Arguments</b>   | <i>unsigned short port_nr</i><br>Specifies the port number of the remote POP server.<br>Port numbers 21(FTP), 53(DNS), 25(SMTP), 123(NTP) are reserved port numbers and may therefor not be used.                                                                                                                      |  |
| <b>Returns</b>     | See 3.20 'change_FTP_remote_port_nr'                                                                                                                                                                                                                                                                                   |  |
| <b>Remarks</b>     | This function should only be used if the remote POP server does not 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

|                    |                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns a pointer to the version string of the TCP/IP library.                                                        |
| <b>Syntax</b>      | char *tcpip_lib_version(void)                                                                                         |
| <b>Arguments</b>   | -                                                                                                                     |
| <b>Returns</b>     | Returns a pointer to the version string of the TCP/IP library, i.e. XMAV0306 or XPAV0306                              |
| <b>Remarks</b>     | 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())
        idle();
}
```

## 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 successfully retrieved        |
|                          |   |   |   |   |     | *  |     | IP_OK                       | (No Error): IP address information was successfully retrieved |
|                          |   |   |   |   |     |    | *   | NTP_OK                      | (No Error): Time was successfully retrieved 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 a 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.

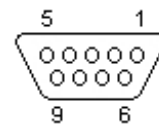
## APPENDIX B: Serial connection from terminal to modem

### 9 pin sub-D female connector

This table describes the connection of the pins on the CRD1001 cradle and commonly available modems that have a 9-pin sub-D RS232 connector.

The OPH100x and H13 cradle should be connected to the modem via a one-to-one connected cable.

| pin # | signal name | DTE<br>(OPH / H13 cradle) | Description         |
|-------|-------------|---------------------------|---------------------|
| 1     | [CD]        |                           | Carrier Detect      |
| 2     | [RxD]       | in                        | Receive Data        |
| 3     | [TxD]       | out                       | Transmit Data       |
| 4     | [DTR]       |                           | Data Terminal Ready |
| 5     | [GND]       | gnd                       | Signal Ground       |
| 6     | [DSR]       |                           | Data Set Ready      |
| 7     | [RTS]       | out                       | Request To Send     |
| 8     | [CTS]       | in                        | Clear To Send       |
| 9     | [RI]        |                           | Ring Indicator      |

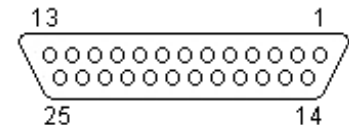


### Sub-D 25 pin female connector

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<br>(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 \*.

Since the OPH100x and H13 only support 2 comports that are both infrared ports, the same AT command is sent in both cases:

|      |                     |
|------|---------------------|
| COM1 | IrDA                |
| COM2 | Cradle (=also IrDA) |

ATZ  
ATE0M1&K&D0\N3W1 <sup>(1)</sup>  
..... <sup>(2)</sup>  
ATDT<tel> <sup>(3)</sup>

<sup>(1)</sup> The standard configuration commands <sup>(1)</sup> won't be sent when using the connect\_user\_specified\_ATcommand( )-function.

<sup>(2)</sup> 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.

<sup>(3)</sup> 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 use: *isp.phonenumber* -> "ATD\*99\*\*\*1#" when choosing for a GPRS connection, see Appendix D)

## 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 or modems with GPRS subscription.

The advantage of using the GPRS network of a mobile phone/modem 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 / modem 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.)

To 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 provider or modem 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)

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

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

The following example demonstrates how a GPRS connection can be made:

```
#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;

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 an ISP user
    strcpy (isp.pass, ""); //name of password, but this depends on the provider.

    isp.com = 1; // use IrDA (or cradle)
    isp.abort_key = dns.abort_key = CLR_KEY;

    // 2: Display progress messages and a progress bar
    // 0: Turn off the debug status messages
    TCPIP_init(2, 0);

    setfont(SMALL_FONT, NULL);

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

    if (error < 0)
        printf("\fError: %d",error);
    else if( (error = get_local_IP_address(&dns, 30)) < 0)
        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();
}
```

**Note:** If the function **connect\_additional\_ATcommand** is not sufficient to establish a GPRS connection using a GRPS modem (i.e. because a PIN-code must be negotiated) you must send the required AT-commands yourself without using any library functions and use the function: **'register\_manual\_connection'** as soon your connected to the ISP.

### ***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 their selves. 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 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. 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 correct serial port of the OPH100x or the H13 using the function 'comopen'

| COM-port | Description          |
|----------|----------------------|
| 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: **`sendstring( )`**
- wait for a certain response of the modem: **`checkstring( )`**

|                                                                                           |                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>void sendstring(const char *string)</code>                                          | Transmit as string to the modem.                                                                                                                                                         |
| <code>int checkstring(const char *string,<br/>int timeout,unsigned char abort_key)</code> | Waits till the desired response is received from the modem or the time out has passed.<br>(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;
```



```

#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 < 0)
            printf("\fError: %d",error);
        else if( (error = get_local_IP_address(&dns, 30)) < 0 )
            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

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

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

|                                     |                 |   |                  |
|-------------------------------------|-----------------|---|------------------|
| <input checked="" type="checkbox"/> | ZONNET          | : | 06760 - 75030    |
| <input checked="" type="checkbox"/> | XS4ALL          | : | 020 -5350535     |
| <input checked="" type="checkbox"/> | RAKETNET        | : | 06760 - 50300    |
| <input checked="" type="checkbox"/> | 12MOVE          | : | 06760 - 77777    |
| <input checked="" type="checkbox"/> | WANADOO         | : | 06760 - 22205    |
| <input checked="" type="checkbox"/> | FREE ACCESS     | : | 023 - 7430000    |
| <input checked="" type="checkbox"/> | DIRECT internet | : | 06760 - 12321    |
| <input checked="" type="checkbox"/> | FREE.FR         | : | 0033 - 173922000 |

### *FTP servers*

|                                     |            |   |                         |                   |
|-------------------------------------|------------|---|-------------------------|-------------------|
| <input checked="" type="checkbox"/> | ZONNET     | : | ftp.home.zonnet.nl      | [062.058.050.014] |
| <input checked="" type="checkbox"/> | 12MOVE     | : | home.12move.nl          | [062.035.014.015] |
| <input checked="" type="checkbox"/> | WANADOO    | : | home.wanadoo.nl         | [194.134.035.012] |
| <input checked="" type="checkbox"/> | IFRANCE    | : | ftp.ifrance.com         | [062.039.122.012] |
| <input checked="" type="checkbox"/> | RAKETNET   | : | ftp.raketnet.nl         | [213.197.030.200] |
| <input checked="" type="checkbox"/> | XS4ALL     | : | reflectix.xs4all.nl     | [194.109.006.026] |
| <input checked="" type="checkbox"/> | OVfiets.nl | : | ftp.OVfiets.nl          | [212.204.242.076] |
| <input checked="" type="checkbox"/> | FREE.fr    | : | ftpperso1-2.free.fr     | [213.228.000.170] |
| <input checked="" type="checkbox"/> | NMPP.fr.ft | : | paramdiffuseurs.nmpp.fr | [172.017.000.004] |

### *POP servers*

|                                     |          |   |                  |                   |
|-------------------------------------|----------|---|------------------|-------------------|
| <input checked="" type="checkbox"/> | ZONNET   | : | pop3.zonnet.nl   | [062.058.050.010] |
| <input checked="" type="checkbox"/> | 12MOVE   | : | pop3.12move.nl   | [195.241.076.041] |
| <input checked="" type="checkbox"/> | WANADOO  | : | pop.wanadoo.nl   | [194.134.035.136] |
| <input checked="" type="checkbox"/> | IFRANCE  | : | pop3.ifrance.com | [062.039.122.017] |
| <input checked="" type="checkbox"/> | RAKETNET | : | pop3.raketnet.nl | [213.197.030.201] |
| <input checked="" type="checkbox"/> | XS4ALL   | : | pop.xs4all.nl    | [194.109.006.055] |

### *SMTP servers*

|                                     |          |   |                  |                   |
|-------------------------------------|----------|---|------------------|-------------------|
| <input checked="" type="checkbox"/> | ZONNET   | : | smtp.zonnet.nl   | [062.058.050.046] |
| <input checked="" type="checkbox"/> | 12MOVE   | : | smtp.12move.nl   | [195.241.076.179] |
| <input checked="" type="checkbox"/> | WANADOO  | : | smtp.wanadoo.nl  | [194.134.035.138] |
| <input checked="" type="checkbox"/> | IFRANCE  | : | smtp.ifrance.com | [062.039.122.20]  |
| <input checked="" type="checkbox"/> | RAKETNET | : | smtp.raketnet.nl | [213.197.030.201] |
| <input checked="" type="checkbox"/> | XS4ALL   | : | smtp.xs4all.nl   | [194.109.006.051] |
| <input checked="" type="checkbox"/> | OPTICON  | : | mail.opticon.nl  | [080.073.128.038] |

### *Equipment used*

|          |   |                                                                                                                                |
|----------|---|--------------------------------------------------------------------------------------------------------------------------------|
| Terminal | : | OPH1003, OPH1004 and H13 with their cradles                                                                                    |
| Modem    | : | Dynalink 1456E-R2 external 56k modem<br>Tornado V90 external 56k modem<br>Siemens MC35I GSM/GPRS modem<br>Opticon Ethernet box |
| Adapter  | : | 25-pins male gender changer<br>25-pins NULL modem<br>2x '9-pins male to 25-pins female' connector                              |

## **APPENDIX G: 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 work with this library! ). <http://ntp.isc.org/bin/view/Servers/WebHome>