IrDA lite Library

for PHL Hand-held Terminals

Application Development Manual
IrDA library versions:
COWV0106 and COGV0106

Introduction

The IrDA Library for PHL Hand-held Terminals provides Opticon's PHL range of hand-held data collection terminals with the possibility to exchange data over a wireless infrared IrDA connection with mobile phones, IrDA printers and other infrared devices that support the IrDA protocol.

The IrDA library can be used for the following purposes:

- Establish a wireless infrared connection with mobile phones that support IrDA
  - to use the mobile phone as modem
  - to send SMS messages
- Establish a wireless infrared connection with mobile IrDA printers to print data
- Transferring data over a wireless IrDA connection with other IrDA devices that support the IrCOMM or IRLPT protocol

The IrDA library can not be used for the following purposes:

- Setting up a connection between a PHL terminal and the infrared port of a laptop, PC or another PHL terminal.
- Setting up a connection with IrDA devices that do not support the IrCOMM or IRLPT protocol

This manual describes the use of the TCP/IP library for PHL Hand-held Terminals in a C language application for the data collection terminals: PHL1300, PHL1700-10, PHL1700-20 and PHL2700.

How to build an application using the library

To build an application, see the programming manual of your type of PHL terminal. The functions of the IrDA library can be linked by adding one of the following 2 lines to the link file, depending on the model of your PHL terminal, e.g. in front of the line "LOAD c:\mccm77\CM77ISLC.LIB"

```
LOAD COWV0106.LIB       ; Load IrDA library for the PHL-1300 or PHL-2700

        OR

LOAD COGV0106.LIB       ; Load IrDA library for the PHL-1700-10 or PHL-1700-20
```

In all the source files that make use of the functions of the IrDA library, include the file 'irda_lib.h".

All the functions in the IrDA library are described in the following pages.

In appendix 1 information is given on how the hardware must be set-up to establish an IrDA connection between the PHL terminal and a mobile phone that supports IrDA, for instance to transfer data to a remote PC.

In appendix 2 an example program is included to demonstrate how a connection can made between a PHL, a mobile phone and a PC with a modem.
Because almost all the functions of this library are used in this program, it can be used as a useful reference guide on how to use this library.

In Appendix 3 is a small trouble-shooting guide in case of communication problems

General functions

com_open

Description Opens the given COM-port of the PHL terminal. If the IrDA port (port = 1) is opened the IrDA stack is automatically initialised.

Syntax
```
int com_open(int port, char abort_key)
```

Arguments

`port`	The COM-port that must be opened 0 = cable 1 = IrDA port 2 = cradle
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- IrDA port opened, stack initialised	0
- COM-port, other then the IrDA port, successfully opened	1
- (syntax) error, i.e. no (compatible) IrDA device in range	-1
- Aborted by user	-2

Remarks This function must always be used (with port = 1) before the remaining functions of this library can be used.
The PHL can't connect with all hardware (mobile phones, laptops, printers etc.) with IrDA capabilities, see Appendix 1.

com_close

Description Closes the given COM-port of the PHL terminal. If the IrDA port was opened, the IrDA connection is closed as well

Syntax
```
int com_close(int port, char abort_key)
```

Arguments

`port`	Number of the COM-port that must be opened 0 = cable 1 = IrDA port 2 = cradle
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- COM-port closed	0
- Aborted by user	-2

Remarks This function can also be called if the IrDA stack wasn't initialised.

irda_lib_version

Description Returns the library version string of this IrDa library

Syntax
```
char *irda_lib_version(void)
```

Arguments -

Returns
- pointer to a buffer, into which the version string is put

Remarks
- the buffer size is 8 characters.
- the string is not zero-terminated.

irda_com_is_empty

Description Checks to see if there is still any data left in the send-buffer of the virtual comport, that hasn't been sent yet.

Syntax `int irda_com_is_empty(void)`

Arguments -

Returns
- send-buffer is not empty 0
- send-buffer is empty 1

Remarks This function is intended to be used to check if all data has been send that has been put in the send-buffer by the functions: 'putcomF' and 'transmit_string_ir'.

Important:
If you need to wait (by using a software loop) untill the irda comport buffer is empty, make sure the loop contains the function call Delay(<value>), otherwise to IrDA connection will be lost.

irda_com_is_full

Description Checks send-buffer of the virtual com-port to see if it is full or not.

Syntax `int irda_com_is_full(void)`

Arguments -

Returns
- send-buffer is not full 0
- send-buffer is full 1

Remarks This function is intended to be used to check if send-buffer of the virtual comport is full, so it can be determined whether the functions: 'putcomF' and 'transmit_string_ir' can be successfully called or not.

Important:
If you need to wait (by using a software loop) untill the irda comport buffer is no longer full, make sure the loop contains the function call Delay(<value>), otherwise to IrDA connection will be lost.

IrDA_SetFlowControl

Description Enables/Disables the Flow control initialisation when establishing a connection using com_open.

Syntax `void IrDA_SetFlowControl(int on_off)`

Arguments
Execute Flow control initialisation at com_open() ON (default)

Skip Flow control initialisation at com_open() OFF

Returns -

Remarks Some IrDA devices don't accept Flow-control commands during the initialisation of an IrDA connection. These commands can then be misinterpreted and can cause the IrDA device to misinterpret these commands as data or to respond in an unexpected way. This can result in unwanted (printed) data, a connection failure or even a crash of the OPL972x. If this occurs, try adding the following line at the start of the main() function of your application:

```
IrDA_SetFlowControl(OFF);       // Disable FlowContol initialization of IrDA stack
```

get_comF

Description Attempts to read a byte from the IrDA connection before the timeout occurs. The IrDA connection is maintained

Syntax
```
int get_comF(int timeout, char abort_key)
```

Arguments

`timeout`	time = timeout x 20ms
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- Succesfull, character value (00..FF) is returned	>=0
- Time out	-1
- Aborted by user	-2

Remarks Be sure that the IrDA connection was established by using the com-open( ) function, before this function is called

put_comF

Description Sends a byte over the IrDA connection. The IrDA connection is maintained

Syntax
```
Int get_comF(int timeout, char abort_key)
```

Arguments

`timeout`	= timeout x 20ms
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- Succesfull	0
- Time out	-1
- Aborted by user	-2

Remarks Be sure that the IrDA connection is established by using the com-open( ) function, before this function is called

Important:
Only use this function when a single character needs to be send to the secondary IrDA device, because each character that's being send using this function will be wrapped in a separate IrDA data packet of about 60 bytes.
Sending more characters after one each other using this function will therefor result in a very slow connection speed and the irda comport buffer will be full very quick. If you need to send more data always of the function: transmit_string_ir()

transmit_string_ir

Description Transmits a string of characters over the IrDA connection and maintains the connection

Syntax
```
int transmit_string_ir(char *string, int len,
                       char abort_key)
```

Arguments

`string`	String of characters
len	Length of the string
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- Successful 0
- Time out -1
- Aborted by user -2

Remarks Be sure that the IrDA connection is established by using the com-open( ) function, before this function is called

Important:
If you need to call transmit_string_ir multiple times in a row, use the functions irda_com_is_empty or irda_com_is_full to check if all data has been succesfully send or to check if more data can already be send.

receive_string

Description Receives data until the passed string (CheckStr) occurs in the received data or until a timeout has occurred. Also maintains the IrDA connection.

Syntax
```
int receive_string(char *CheckStr, int timeout,
                   char abort_key)
```

Arguments

`checkStr`	String of characters
`timeout`	Timeout value (x 20ms)
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- Successful 0
- Time out or no match -1
- Aborted by user -2

Remarks If more than 512 characters are received before the CheckStr is matched to function returns: '–1'.

Be sure that the IrDA connection is established by using the com-open( ) function, before this function is called

Delay

Description Maintains the IrDA-connection during the period of this delay routine

Syntax
```
void Delay(int time, char abort_key)
```

Arguments

`time`	delay time = time x 20ms
`abort_key`	Specifies the key that can be used to abort the function.

Returns

- Successful 0
- Aborted by user -2

Remarks Be sure that the IrDA connection is established by using the com-open( ) function, before this function is called

Appendix 1: Hardware set-up

The IrDA library can be used for the following purposes:

- Establish a wireless infrared connection with a mobile phone that support IrDA
  - to use the mobile phone as modem
  - to send SMS messages
- Establish a wireless infrared connection with a mobile IrDA printer to print data
- Transfer data over a wireless IrDA connection with other IrDA devices that support the IrCOMM protocol

The IrDA library can NOT be used for the following purposes:

- Setting up a connection between a PHL terminal and the infrared port of a laptop, PC or another PHL terminal.
- Setting up a connection with IrDA devices that do not support the IrCOMM protocol

The IrDA software has been tested on the following models of mobile phones that support IrDA:

- the Nokia 6210
- the Nokia 7110
- Ericson T68.

Also the IrDA software has been tested on the following models of IrDA printers:

- Custom S'Print-I
- Canon BJC-50
- EXTECH S1500T

Note:
It can not be guaranteed that the software will work on other models of mobile phones, printers and other devices that support IrDA.

The following hardware set-up can be used to establish an IrDA connection between a PHL terminal a mobile phone and a remote PC.

[PHL terminal] ←→ [Nokia 6210/7110/ T68] ←→ [Extern modem] ←→ [PC with pro-comm plus in host-mode]
(9600bps) (9600bps)

Before a connection can be established the mobile phone must be ready the receive IrDA-commands. This means that the IrDA port of the mobile phone must be activated.

Note:
It's not possible to establish a connection with certain types of mobile phones, like the Nokia 6210, the Nokia 7110 and the Ericson T68 within the first 15 to 30 seconds after the previous connection was abnormally cancelled. For instance if the secondary device was removed from the line of sight of the PHL terminal during the connection or after a failed connection attempt.

Appendix 2 : Example application

The source code on the following pages is an example program that can be used to transfer a file from the PHL terminal to a remote PC (with pro-comm plus in host-mode) by using an IrDA connection with a mobile phone. In appendix 1 the hardware set-up for this program was explained.

All the functions of the library that are used in this application are listed below.

- com_open( )
- com_close( )
- get_comF( )
- put_comF( )
- transmit_string_ir( )
- receive_string( )
- irda_com_is_empty( )
- Delay( )

Note:
This example application does not include the actual sendig of a file to pro-comm plus.
The complete version of this example application can be found in the software developers kit of this IrDA library.

The following other IrDA demonstration applications can be requested at Option Sensors Europe BV if desired:

CFx23770.S32 **:** Graphical demonstration application with wireless file transfer by using the IrDa library

CFx29450.S32 **:** Graphical demonstration application that sends plain barcode data to IrDA printers and other IrDA devices

CFx23830.S32 **:** Graphical demonstration application made specifically for the Custom S'Print-I printer

Example application

```
// ***********************************************************************
// testIrDa version 1.0
// ***********************************************************************

#define WINDOWS            0
#define DOS                1

void IrDA_procomm_example (int procomm_version) {

       char telnr[]      = "ATDT0230000000\r";
       char f_name[]     = "John\r";
       char fl_name[]    = "John Doe\r";
       char l_name[]     = "Doe\r";
       char p_pass[]     = "opticon\r";
       char filename[]     = "DEMO.TXT\r";
       char d_descript[] = "Demo test bestand\r";
       char r_reset[]    = "ATZ\r";

       char first_name[] = "rst";
       char full_name[]  = "ULL Name:";
       char last_name[]  = "ast name:";
       char file_name[]  = "ile name? ";
       char correct[]    = "correct";
       char password[]   = "ssword:";
       char choice[]     = "hoice? ";
       char descript[]   = "escription: ";
       char procedure[]  = "rocedure...";
       char complete[]   = "FER COMPLETE";
       char isOK[]       = "K\0";

       int num_length;

#ifdef PHL2700
       setfont(MEDIUM_FONT,NULL);
#else
       setfont(SMALL_FONT,NULL);
#endif

              printf("\fInit irDa...\n");
       if((error = com_open(1, CLR_KEY)))
              goto error_label;

       if((error = Delay(50, CLR_KEY)))
              goto error_label;

       transmit_string_ir(r_reset,4, CLR_KEY);

       if((error = receive_string(isOK, 250, CLR_KEY)))
              goto error_label;

       printf("Calling %s\n",telnr+4);
       transmit_string_ir(telnr,num_length, CLR_KEY);

       if((error = Delay(500, CLR_KEY)))
              goto error_label;\

       printf("Waiting for reply...\n");

       //  procomm for windows
       if(procomm_version==WINDOWS)
       {
              if((error = receive_string(full_name, 2000, CLR_KEY)))
                     goto error_label;

              printf("Full name: ");
              transmit_string_ir(fl_name,9, CLR_KEY);
              printf("John Doe\n");
       }
```

```
else
{
	if((error = receive_string(first_name, 1250, CLR_KEY)))
		goto error_label;

	printf("First name: ");
	transmit_string_ir(f_name,5, CLR_KEY);
	printf("John\n");

	if((error = receive_string(last_name,250, CLR_KEY)))
		goto error_label;

	printf("Last name: ");
	transmit_string_ir(l_name,4, CLR_KEY);
	printf("Doe\n");
}

if((error = receive_string(correct,500, CLR_KEY)))
	goto error_label;

printf("Is this correct: ");
put_comF('Y', CLR_KEY);
printf("Yes\n");

if((error = receive_string(password,500, CLR_KEY)))
	goto error_label;

printf("Password: ");
transmit_string_ir(p_pass,8, CLR_KEY);
printf("*******\n");

if((error = Delay(150, CLR_KEY)))
	goto error_label;

if((error = receive_string(choice,500, CLR_KEY)))
	goto error_label;

printf("Enter choice: ");
put_comF('U', CLR_KEY);
printf("Upload\n");

if((error = Delay(150, CLR_KEY)))
	goto error_label;

if(procomm_version!=WINDOWS && (error = receive_string(choice,500, CLR_KEY)))
	goto error_label;

printf("Enter choice: ");
put_comF('K', CLR_KEY);
printf("Kermit\n");

if((error = Delay(50, CLR_KEY)))
	goto error_label;

// procomm for windos
if(procomm_version==WINDOWS)
{
	transmit_string_ir(filename,10, CLR_KEY);
	printf("File name: 'DEMO.TXT'\n");

	if((error = Delay(100, CLR_KEY)))
		goto error_label;
}
else // procomm for dos
{
	if((error = receive_string(file_name,500, CLR_KEY)))
		goto error_label;

	transmit_string_ir(filename,10, CLR_KEY);
	printf("File name: 'DEMO.TXT'\n");
```

```
		if((error = receive_string(descript,500, CLR_KEY)))
			goto error_label;

		printf("Description:\n");
		transmit_string_ir(d_descript,18, CLR_KEY);
		printf("'Demo test bestand'\n");

		if((error = receive_string(procedure,1200, CLR_KEY)))
			goto error_label;

		printf("Init file transfer\n");

		if((error = Delay(50, CLR_KEY)))
			goto error_label;
	}

	filename[8] = '\0';
	printf("Transferring file...\n");

	/* This example application does not include the actual sendig of a file to procomm
	if((error = sendsw(filename, CLR_KEY)))
		goto error_label;
	*/
	printf("Sending of file has not be implemented in this application!!");

	if((error = receive_string(complete,500, CLR_KEY)))
		goto error_label;

	printf("Transfer complete\n");

	if(procomm_version!=WINDOWS && (error = receive_string(choice,500, CLR_KEY)))
		goto error_label;
	else if((error = Delay(200, CLR_KEY)))
		goto error_label;

	printf("Enter choice: ");

	put_comF('G', CLR_KEY);
	printf("Goodbye\n");

	if((error = Delay(250, CLR_KEY)))
		goto error_label;

	com_close(1, CLR_KEY);

	setfont(LARGE_FONT,NULL);

	return;

error_label:
	ErrorC(error);
	return;
}

void ErrorC(int error) {
	printf("\nError %d occured",error);
	Delay(50, CLR_KEY);
	com_close(1, CLR_KEY);
	setfont(LARGE_FONT,NULL);
}
```

Appendix 3: Trouble-shooting (Questions and Answers)

Q1: My IrDA printer prints unwanted characters after the initialisation of the connection
Q2: The communication fails directly after the initialisation of the connection
Q3: The PHL crashes directly after or during the initialisation of the connection

A1..3: Some IrDA devices don't accept certain additional commands during the initialisation of the IrDA connection. These commands can then be misinterpreted and can cause the IrDA device to misinterpret these commands as data or to respond in an unexpected way. If any of the 3 problems listed above occur, then try adding the following line at the start of the main() function of your application:

```
IrDA_SetFlowControl(OFF);  // Disable FlowContol initialization of IrDA lite stack
```

Q4: I can't compile/build my software after I added IrDA functions to my software

A4:
Make sure you've verified all the following steps:

- Check if you added the IrDA library to link-command in the link-file:
  `Example: tulink startup.rel LOAV0104.LIB testprog.rel etc.`
- Check if you included the file: 'irda_lib.h' in all the source files that make use of the IrDA library
- Check if you have OS version 1.36 (or higher) on your PHL

Q5: Making an IrDA connection always fails:

A5:

- Check if your IrDA device supports the IrCOMM or IRLPT protocol
- Check if the IrDA sensor of the remote device is in line of sight of the IrDA sensor of the PHL
- Check if IrDA is enabled on your remote IrDA device
- The IrDA lite library (normally) can't be used to set up a between an PHL terminal and the infrared port of a laptop, PC or another PHL terminal.

Q6: The IrDA communication is extremely slow

A6:

- If you use the function 'put_comF' a lot in your software, realise that each character that's being send using this function will be wrapped in a separate IrDA data packet of about 60 bytes. Sending more characters after one each other using this function will therefor result in a very slow connection speed and the IrDA comport buffer will be full very quick. If you need to send more data always of the function: `'transmit_string_ir()'`
