BIS C-870 handheld driver for Windows CE 6.0

User's Manual

Table of content

1 Introduction

BIS_C_WinCE_DLL.dll is a driver, written for BIS C-870, which covers the user's command set of *Balluff Dialog* protocol and also provides some extra functionality to the user under *Windows CE* environment.
The driver is written in native C++ code and designed for Windows CE 6.0. The functions are exported in C format, ignoring name-mangling.
It is prepared for use of 2 virtual communication ports (VCP), which has the driver from *Silabs*. (https://www.silabs.com)

The Read/Write head module connects to the handheld via USB port, thus the functions of the USB Host (like *open, close, read, write* etc.) is reached through function-calls of the provided USB driver interface (SIUSBXP_LIB.dll).

Implemented command set of Balluff Dialog:

- ReadData – read data from data carrier (normal and 60R type)
- WriteData – write data to data carrier (normal and 60R type)
- WritePattern – write a constant character to the data carrier
- InitDataCarrier – init data carrier for CRC16
- InitReader – set configuration of the processor
- ResetReader – reset read/write processor
- StopCommand – stop actual running command
- MemoryLock – locks the desired area of the code tag

There are other functions which are not part of the Balluff Dialog protocol, but can be called via the driver:

- *USBPortState* – gets the state of the USB port (open, closed, exist etc.)
- *OpenUSBPort* – opens the USB port
- *CloseUSBPort* - closes the USB port
- *OpenReader* – initializes and tests the read/write head
- *CloseReader* – closes the read/write head
- *GetDLLVersion* – gets the version info of the driver (version number, release date, BIS system type)
- *GetLastError* – gets the error number of the last error within the driver
- *PowerOffRWHead* – turns off the power of the USB port, thus the RW head
- *UnLoadUSB* – close the USB port, but keeps the connection handle to opened BIS
- *ReLoadUSB* – re-opens the USB port to use the original connection handle

2 Additional files to the project

As it was mentioned before, the device uses USB driver from Silabs. Thus the following files are needed to be added to the project:

- *SIUSBXP.dll* USB driver of Silabs; has to be added in *System32* folder of the op. system
- *SIUSBXP_LIB.h* header file to the driver function declarations

- *SIUSBXP_LIB.lib* links the driver and header file
- *SIUSBXP_LIB.dll* interface library to the driver

This driver is written specially for pocket PC from *Zebra Technologies* (*PSION Teklogix)*. When user turns on the PC, the USB port is *not powered*, thus the driver has to power on the port. There is a function `EnablePower` in the PSION's Workabout API collection. Because of this the following files has to be added to the project as well:

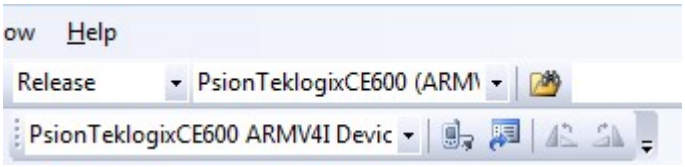
- *ExpansionSlotLibrary*.lib — this is a static library, containing API functions of the pocket PC
- *PsionTeklogixWORKABOUTPROHDK*.hpp — the header of the static library with the function declarations
- *PtxSdkCommon.dll* — this is a library, containing common API functions regarding to PSION WAP

At runtime the following 4 files are needed: SIUSBXP_LIB.dll, which is an API collection to SIUSBXP.dll driver. (SIUSBXP.dll has to be installed in Windows folder), PtxSdkCommon.dll and BIS_C_WinCE_DLL.dll.
An additional file is needed (SIUSBXP.reg) which will create the entries for the USB driver in the Registry.

We recommend using Microsoft Visual Studio 2005 or higher to develop RFID applications for mobile devices. *Visual Studio maintains mobile solutions from Professional version!*

Before starting of development, Psion Teklogix Mobile Devices SDK for Windows® CE .NET CE 6.0 has to be installed (Find in this SDK or download from the homepage of Zebra Technologies: *PlatformCE600.msi*).

Then in Visual Studio the *PsionTeklogixCE600 (ARMV4I)* has to be selected as solution platform. Target platform is *PsionTeklogixCE600 ARMV4I Device*. These options allow the developer to develop against the appropriate hardware and environment running on PSION WorkaboutPro4. (See picture below.)



3 Header files

Only one header file has to be added to the project; *BISReader.h*.

3.1 BISReader.h

This header contains the function declarations to use BIS C-870 reader and all the BIS C-870 related types, structures, enumerations and macros (definitions). All Balluff Dialog error codes and driver-related error codes are defined here.
User also has to add *BIS_C_WinCE_DLL.lib* static library to the project. This library file links the declared functions to the driver.

Definitions and enumerations

```
#define ERR_NOERROR                 0    //<Ack>'0' - wait for data sending
#define ERR_NO_DC_PRESENT           1    //No data carrier present
#define ERR_READ                    2    // . . .
#define ERR_READ_DC_REMOVED         3
#define ERR_WRITE                   4
#define ERR_WRITE_DC_REMOVED        5
#define ERR_INTERFACE               6
#define ERR_TELEGRAM_FORMAT         7
#define ERR_BCC                     8
#define ERR_CABLE                   9
#define ERR_NEW_CMD_READ            10
#define ERR_NEW_CMD_WRITE           11
#define ERR_NEW_CMD_HEADSELECT      12
#define ERR_HEAD_COMMUNICATION      13
#define ERR_DC_CRC                  14
#define ERR_DC_ADDR                 15
#define ERR_DC_FUNC_NOT_SUPPORTED   16
#define ERR_EEPROM                  18
#define ERR_LOCK                    19



#define ERR_UNKNOWN                 -1   //unknown error

#define ERR_COMPORT_USED            49   //COM port is busy
#define ERR_RWHEAD_USED             50   //RW head is busy
#define ERR_COMPORT_OPENED          51   //COM port is opened
#define ERR_COMPORT_CLOSED          52   //COM port is not opened
#define ERR_COMPORT_EXIST           53   //COM port exists
#define ERR_COMPORT_NOT_EXIST       54   //COM port doesn't exist
#define ERR_READER_EXIST            55   //The R/W head is found
#define ERR_READER_NOT_EXIST        56   //The R/W head doesn't exist
#define ERR_READER_OPENED           57   //R/W head is opened successfully
#define ERR_READER_OPENING          58   //R/W head couldn't be opened
#define ERR_READER_CLOSED           59   //R/W head closed successfully
#define ERR_READER_CLOSING          60   //R/W head couldn't be closed
#define ERR_DLL_VERSION_EXIST       61   //DLL version information exists
#define ERR_OTHER_PORTS_USED        62   //This process uses other USB ports
#define ERR_FREE_PORT_OBJECT        63   //At least one free port object
                                           exists
#define ERR_NO_FREE_PORT_OBJECT     64   //No free Serial port object exists
```

```
#define ERR_NO_ANSWER                65   //No answer from the BIS R/W head
#define ERR_INVALID_COMM_HANDLE      66   //The communication handle passed is
                                            invalid
#define ERR_READER_INIT_FAILED       67   //Reader initialization process
                                            failed
#define ERR_DATACARRIER_INIT_FAILED 68   //Data carrier initialization process
                                            failed
#define ERR_DATA_OVER_1023_BYTES     69   //Datalength is over 1024 bytes
#define ERR_TOO_SMALL_DATA           70   //Datalength is 0
#define ERR_TEST_READER              71   //Reader test is failed
#define ERR_UNKNOWN_CODETAG          72   //Unknown Code Tag type
#define ERR_TURN_ON_ANT              73   //Turn On the antenna failed
#define ERR_TURN_OFF_ANT             74   //Turn Off the antenna failed
#define ERR_CHANGE_READER_KEY        75   //Error changing READER key
#define ERR_CHANGE_TAG_KEY           76   //Error changing TAG key
#define ERR_WRONG_KEY_LENGTH         77   //Too short or too long key
#define ERR_WRONG_CODETAG_TYPE       78   //The codetag doesn't maintain this
                                            function
#define ERR_UNLOAD_USB               79   //USB port unload is failed
#define ERR_RELOAD_USB               80   //USB port reload is failed
#define ERR_USBPORT_INIT             81   //USB port initialization failed
#define ERR_POWEROFF_RWHEAD          82   //Power Off RW head failed
#define ERR_READ_COMPORT             83   //Read COM port failed
#define ERR_WRITE_COMPORT            84   //Write COM port failed
#define ERR_INVALID_PARAMETER        85   //Invalid parameter found
#define ERR_MEMORY_LOCK              86   //Error during Code Tag memory
                                            optimization



#define CRC16_NO        false
#define CRC16_YES       true
```

//Specifies command ending of Balluff 007 protocol

```
#define BIS_BCC         0x00
#define BIS_CR          0x01
#define BIS_CRE         0x02
#define BIS_LFCR        0x03

typedef enum
{
      BIS_C_Unknown     = 0x00,
      BIS_C_1xx_04_05,
      BIS_C_1xx_11_32,

} CodeTags;


typedef enum
{
      _4_Bytes          = 0x01,
      _8_Bytes,
      _16_Bytes

} LockArea;
```

Structures

```
typedef struct BIS_DLL_Version
{
      TCHAR Version[5];                  //4 digits for versioning
      TCHAR DateOfRelease[11];           //Date of release of the BIS DLL
      TCHAR SystemType[2];               //C

} BISVERSION, *lpBISVERSION;



typedef struct ConnectionHandle
{
      UINT  COMPort;                //Number of the used COM port
      UINT  LOCK;                   //Indicates the running command (not used)
      DWORD Process;                //Process handle
      CodeTags CodeTagType;         //Type of the Codetag
      bool  CRCCheck;               //Use CRC check

} CONNHND, *lpCONNHND;

typedef enum
{
      _4_Bytes          = 0x01,
      _8_Bytes,
      _16_Bytes

} LockArea;
```

4 Driver functions in details

The user communicates with the BIS module via the function calls of the driver. The following chapter describes these functions.
In order to the user can use the drivers functionalities, he has to open a port with *BIS_OpenUSBPort* and open the BIS module with *BIS_OpenReader*. When functions return without error, user has a valid connection handler with which he can access the functions.
User should never change this connection handler manually, because he can't call the other functions after that.

The following section contains the list of the exported functions.

4.1 BIS_USBPortState

Checks the status of the given port.

Prototype: `int BIS_USBPortState(UINT PortNumber)`

Parameters: 1. *PortNumber* – the number of the USB port

Return value: `ERR_COMPORT_NOT_EXIST`
`ERR_COMPORT_USED`
`ERR_COMPORT_CLOSED`
`ERR_UNKNOWN`

4.2 BIS_OpenUSBPort

Opens the given port and gives back a Connection Handle.

Prototype: `int BIS_OpenUSBPort(UINT PortNumber, CONNHND& ConnHnd)`

Parameters: 1. *PortNumber* – the number of the USB port
2. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_COMPORT_CLOSED`
`ERR_COMPORT_OPENED`

4.3 BIS_OpenReader

Opens the BIS module on the previously opened port and sets the Connection Handle.

Prototype: `int BIS_OpenReader(CONNHND& ConnHnd, bool bCRC16, CodeTags DataCarrierType)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable
2. *bCRC16* – true if the BIS module is set to use CRC16 check
3. *DataCarrierType* – defines which data carrier type can be used to the reader

Return value: `ERR_READER_OPENED`
`ERR_READER_CLOSED`

4.4 BIS_CloseReader

Closes the opened BIS module, accordingly the Connection Handle.

Prototype: `int BIS_CloseReader(CONNHND& ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_READER_CLOSED`
`ERR_READER_OPENED`

4.5 BIS_CloseUSBPort

Closes the opened port, accordingly the Connection Handle.

Prototype: `int BIS_CloseUSBPort(CONNHND& ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_COMPORT_CLOSED`
`ERR_COMPORT_OPENED`

4.6 BIS_InitReader

Sets some parameters of the given opened BIS module, accordingly the Connection Handle.

Prototype: `int BIS_InitReader(CONNHND& ConnHnd, bool bCRC16, CodeTags DataCarrierType)`

Parameters:
1. *ConnHnd* – pointer to a connection handle variable
2. *bCRC16* – set the module to use CRC16 check
3. *DataCarrierType* – set the module to use the chosen data carrier type

Return value: `ERR_NOERROR`
`ERR_READER_INIT_FAILED`

4.7 BIS_ReadData

Reads data from the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_ReadData(CONNHND ConnHnd, UINT StartAddr, UINT Datalength, TCHAR* DataBuffer)`

Parameters:
1. *ConnHnd* – pointer to a connection handle variable
2. *StartAddress* – start address of the data carrier to read from
3. *Datalength* – number of bytes to read (max. size of the DataBuffer)
4. *DataBuffer* – pointer to the data buffer where the data arrives back; Unicode characters

Return value: `ERR_NOERROR`
`ERR_READ`

4.8 BIS_WriteData

Writes data to the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_WriteData(CONNHND ConnHnd, UINT StartAddr, TCHAR* DataToWrite, UINT Datalength)`

Parameters:
1. *ConnHnd* – pointer to a connection handle variable
2. *StartAddress* – start address of the data carrier to write from
3. *DataToWrite* – user data to write onto the data carrier; Unicode characters

4. *Datalength* – the number of the desired bytes to write (max. size of DataToWrite)

Return value: `ERR_NOERROR`
`ERR_WRITE`

4.9 BIS_WritePattern

Writes a constant character to the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_WritePattern(CONNHND ConnHnd, UINT StartAddr, UINT Patternlength, TCHAR Pattern)`

Parameters:
1. *ConnHnd* – pointer to a connection handle variable
2. *StartAddress* – start address of the data carrier to write from
3. *Patternlength* – the number of the bytes to write
4. *Pattern* –contains the pattern character; Unicode

Return value: `ERR_NOERROR`
`ERR_WRITE`

4.10 BIS_InitDataCarrier

Initializes a data carrier with all 0 hex values, preparing it for use of CRC16. Starts from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_InitDataCarrier(CONNHND ConnHnd, unsigned short DataLength)`

Parameters:
1. *ConnHnd* – pointer to a connection handle variable
2. *uiDataLength* – lengh of the memory area to initialize

Return value: `ERR_NOERROR`
`ERR_DATACARRIER_INIT_FAILED`

4.11 BIS_ResetReader

Resets the given BIS module, accordingly the Connection Handle.

Prototype: `int BIS_ResetReader(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_NOERROR`
`ERR_UNKNOWN`
`ERR_HEAD_COMMUNICATION`
`ERR_NO_ANSWER`
`ERR_READ`

4.12 BIS_StopCommand

Stops the actual running command, accordingly the Connection Handle.

Prototype: `int BIS_StopCommand(CONNHND ConnHnd, bool state)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable
2. *state* – true if command should be stopped

Return value: `ERR_NOERROR`
`ERR_READ`

4.13 BIS_MemoryLock

Locks certain areas of the code tag thus that becomes Read-Only. User can choose to lock bytes at 0 to 3, 0 to 7 or 0 to 15 address of the code tag. User also can unlock these areas.

Prototype: `int BIS_MemoryLock(CONNHND ConnHnd, bool Lock, LockArea LockSize)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable
2. *Lock* – specifies whether lock or unlock the code tag
4. *LockSize* – specifies the area size to lock

Return value: `ERR_NOERROR`
`ERR_MEMORY_LOCK`

4.14 BIS_GetDllVersion

Retrieves information about the driver, such as date of release, version, BIS system type.

Prototype: `int BIS_GetDLLVersion(BISVERSION& Version)`

Parameters: 1. *Version* – pointer to the structure where the information will be stored

Return value: `ERR_DLL_VERSION_EXIST` (not an error)

4.15 BIS_GetLastError

Gives back the last error number after a function fails.

Prototype: `int BIS_GetLastError()`

Parameters: -

Return value: `The appropriate error number.`

4.16 BIS_GetLastErrorText

Gives back the last error as a character array after a function fails.

Prototype: `TCHAR* BIS_GetLastErrorText()`

Parameters: -

Return value: `The appropriate error with text.`

4.17 BIS_UnLoadUSB

Closes the USB port, but keeps the Connection Handle for later use. This function has to be called before the Pocket PC goes to Suspended mode, because it powers off the USB port. If the port stays opened, Windows stores its handle, and after the user turns on the WAP4, he can't access the previously opened port. During a USB device enumeration user may find 2 devices, although only 1 exists in the Windows system.

Prototype: `int BIS_UnLoadUSB(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – the connection handle variable

Return value: `ERR_NOERROR`
`ERR_UNLOAD_USB`

4.18 BIS_ReLoadUSB

Re-opens the USB port accordingly the Connection Handle. This function has to be called after the Pocket PC comes back from Suspended mode. This function re-opens the previously opened USB port, thus user can communicate with the same BIS C-870 module, without going through again the opening procedure.

Prototype: `int BIS_ReLoadUSB(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – the connection handle variable

Return value: `ERR_NOERROR`
`ERR_RELOAD_USB`

4.19 BIS_PowerOffRWHead

Turns off the power of the USB port.

Prototype: `int BIS_PowerOffRWHead()`

Parameters: -

Return value: `ERR_NOERROR`

5 Usage of the driver (examples)

In the following chapter the reader can find some code snippets, how to use the driver properly.

5.1 Preparation

Include the main header (BISReader.h), which contains the driver related definitions and the declaration of the exported driver functions.

```
#include "BISReader.h"
```

Before user can use the driver, he has to declare a variable type of CONNHND. This is a connection handler, which contains information about the opened device on the port.

```
CONNHND MyConnection;
```

5.2 Open a BIS module on port

Before user can use a BIS module, he has to open the port and the given device.

```
BIS_OpenUSBPort(1, MyConnection);
BIS_OpenReader(MyConnection, CRC16_NO, CodeTags::BIS_C_1xx_04_05);
```

Important!
Every function call returns with 0 on success (ERR_NOERROR) with the exception of OpenCOMPort and OpenReader functions, of which return codes are ERR_PORT_OPENED and ERR_READER_OPENED respectively.

5.3 Read from the code tag

Create a *DataBuffer*, type of TCHAR to store the retrieved data, and then call the appropriate function.

```
TCHAR DataBuffer[1024];

BIS_ReadData(MyConnection, 0, 1023, ReceivedData);
```

For parameter details, see chapter 4.

5.4 Write to the code tag

```
BIS_WriteData(MyConnection, 0, L"Text to write", 13);
```

The data is waited in Unicode coding! The last parameter indicates how many characters are in the buffer; less can be given, but more may cause runtime errors!

5.5 Close the opened BIS module

In order to user can close an opened connection to a USB port, he has to provide the connection handle in the parameter. Call the functions in the following order!

```
BIS_CloseReader(MyConnection);
BIS_CloseUSBPort(MyConnection);
```