# PCLink

USB Communication Interface

(Rev. 10-Jan12)

Laserpoint srl – Via Burona, 51 – 20090 Vimodrone (Milano) – Italy
Phone +39 02 27 400 236 – Telefax +39 02 25 029 161
www.laserpoint.it

# *Table of Contents*

# *1 Installation*

Connect the PCLink electronics  through the USB 1.1 port to the host PC device by a USB cable A to B type.

Include in your Code the FTD2XX.DLL Dynamic Library (described in Annex 1) for Windows in order to write your application.

# *2 Input Commands and answer messages*

When the PCLink receives a valid input command, it confirms to the host device that the command has been received and return the answer as follows.

## 2.1 Input Commands

The format of a valid command is as follow:

***COMMANDNAME:**

where:

**"*" :** Start of command
**":" :** End of command
**"_" :** space character
**COMMANDNAME :** the instruction as described in the following table; it is an ASCII character sequence. The command name must be in capitals.

## 2.2 Answer messages

PCLink device send a message through USB interface only if it received an Input Command from the Host Device. Maximum response time from PCLink is ~100msec.

The format of an answer is as follow:

**ANSWER ;**

where:

**";" :** End of answer
**ANSWER :** there are three kind of answer

1. String: ASCII character sequence
2. Int: integer number, numerical sequence (in ASCII code)
3. Float: floating point number, numerical sequence plus decimal point (in ASCII code); ex. the NumericValue 23.45 is codified with the 5 ASCII characters "23.45".

## 2.3 Commands & Answers description Table

| Command Name | Description | Answer |
|---|---|---|
| HEADN | Displays the Head model | String |
| SERNU | Displays the Head serial number | Int |
| KEFUN | Displays SW measurement's type:<br>0: Power Meter (PM) mode<br>1: FIT Mode<br>2: Energy (E) Mode<br>3: PM + E mode<br>4: FIT + E mode | Int |
| WSENS | Displays Head sensibility (mV/W) | Float |
| OPDAC | Displays Output DAC (mV/W) | Float |
| PMSEW | Displays maximum power value the Head can withstand (W) | Float |
| STHFW | Displays START threshold (W) in FIT mode | Float |
| HOFTF | Displays FIT inhibition time (sec) | Int |
| ENOMJ | Displays Head nominal energy (J) | Float |
| JSENS | Displays Head sensibility (mV/J) | Float |
| ODACJ | Displays Output DAC (mV/J) | Float |
| EMSEJ | Displays maximum energy value the Head can withstand (J) | Float |
| STHEJ | Displays START threshold (J) in E mode | Float |
| HOFTE | Displays E inhibition time (sec) | Int |
| SETLAM x | Wavelength selection:<br>x=1: CO2 selected<br>x=2: Erb selected<br>x=3: YAG selected<br>x=4: LD selected<br>x=5: VIS selected<br>x=6: EXC selected<br>x=7: UCF selected | **"ok"** if properly selected<br><br>**"nval"** if not available (the first available is selected) |
| LAMBDA | Displays selected wavelength:<br>01: CO2<br>02: Erb<br>04: YAG<br>08: LD<br>16: VIS<br>32: EXC<br>64: UCF | Int |
| SETUCF wxyz | UCF set:<br>wxyz=UCF*1000; four digit integer (range 0,001-9.999) | **"ucfw.xyzok"** if carry out;<br>**"ucf1"** if w,x,y,z are not number; |

| | | |
|---|---|---|
| | | **"ucf2"** if not 4 digit |
| PNOMW | Displays Head nominal power (W) | Float |
| ZERO | Arms PCLink in FIT or E mode; zeroing PCLink in PM mode | "ok" |
| OUTPM | Displays measured power/energy (W/J) | Float |
| TEMP | Displays Head temperature x 10 (°C) | Int |
| WTFIT | Displays FIT waiting time (sec) | Int |
| VISCA | Displays measured value's format mode:<br>0, 1, 2: W for power, J for energy<br>3, 4, 5: mW for power, mJ for energy<br>0-3: no decimal point (ex. 10 W/J)<br>1-4: one decimal number (ex. 10.3 W/J)<br>2-5: two decimal number (ex. 10.35 W/J) | Int |
| LEDPRO | Displays process parameters state:<br>1: OFF<br>2: OK (min threshold<measured value<max)<br>3: High (measured value > max threshold)<br>4: Low (measured value < min threshold) | Int |
| STATUS | Displays condition byte:<br>bit 0: arm/zeroing done; (1) yes, (0) no<br>bit 1: measure running; (1) yes, (0) no<br>bit 2: Head connected; (1) yes, (0) no<br>bit 3: cool alarm running; (1) yes, (0) no<br>bit 4: wait before start a new measure; (1) yes<br>bit 5:not used; default value (0)<br>bit 6: overflow alarm; (1) yes, (0) no<br>bit 7: termistor connected; (1) yes, (0) no | Int |
| STATUSE | Displays condition byte:<br>bit 0: PCLink mode; (1) Energy,(0) PM/FIT<br>bit 1: Tuning; (1) yes, (0) no<br>bit 2-7: not used; default values (0) | Int |
| SETX1 x | Amplifier gain selection:<br>x=0: x1 gain selected<br>x=1: x10 gain selected<br>A successive ZERO must be performed | "ok" |
| X1D | Displays gain set:<br>0: x1 gain<br>1: x10 gain | Int |
| TUNING | Set Tuning Mode:<br>Available if KEyFUNction= 0, 3 | **"ok"** if properly selected<br><br>**"nov"** if not selected |
| TUNE | Set actual measured value as PCLink zero:<br>Available if KEyFUNction= 0, 3 and Tuning Mode selected | **"ok"** if performed<br><br>**"nov"** if not |
| POWER | Set PCLink in Power Meter Mode | **"ok"** if properly selected |

| | | |
|---|---|---|
| | | **"nov"** if not selected |
| ENERGY | Set PCLink in Energy Mode | **"ok"** if properly selected<br><br>**"nov"** if not selected |
| EPOWER | Set PCLink in Power Meter Mode if another Mode of operation is selected | **"ok"** if properly selected<br><br>**"nov"** if not selected |

### 2.4 Error Message

the following error message may be sent by the PCLink if a communication error occurs:

**??;**

where:

**??** : communication error

**";"** : End of answer

An error message may be sent for the following error conditions:

- ✓ Input command not started with * character
- ✓ Input command does not correspond with the command list
- ✓ Input command not in capitals

### 2.5 "no head" Error Message

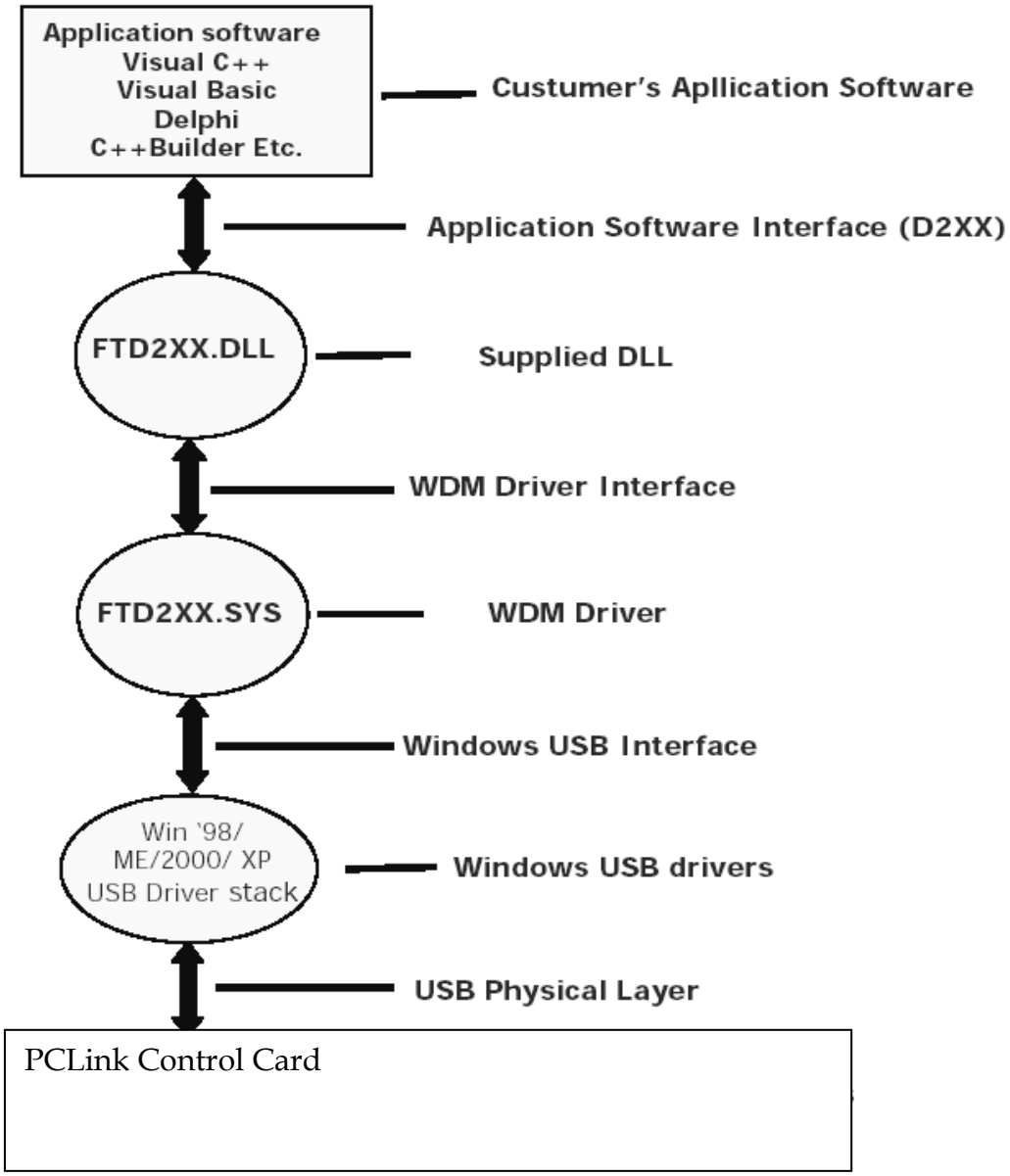
If the PCLink is switched on and the detector head is not plugged in: the PCLink send the "no head" string to the PC every 1 sec. unless the detector head is plugged in.

# *3 Annex 1: FTD2XX.DLL Dynamic Library*

**TABLE OF CONTENTS**

The **FTD2XX.DLL** Dynamic Library for Windows allows you to write your application. The architecture of the FTD2XX.DLL drivers consists of a Windows WDM driver that communicates with the device via the Windows USB Stack and a DLL which interfaces the Application Software (written in VC++, C++ Builder, Delphi, VB etc.) to the WDM driver. The FTD2XX.DLL interface provides a simple, easy to use, set of functions to access **PCLink** control card.

## D2XX Driver Architecture



## DLL Functions

### FT_ListDevices

**Description** Gets information concerning the devices currently connected. This function can return such information as the number of devices connected, and device strings such as serial number and product description.

**Syntax** FT_STATUS **FT_ListDevices** (PVOID *pvArg1*, PVOID *pvArg2,* DWORD *dwFlags*)

**Parameters**
***pvArg1*** meaning depend on the *dwFlags* value (see note below)
***pvArg2*** meaning depend on the *dwFlags* value (see note below)
***dwFlags*** Determines format of returned information (see note below)

**Return Value** FT_OK if successful, otherwise the return value is an FT error code

**Note** Remarks This function can be used in a number of ways to return different types of information.
In its simplest form, it can be used to return the number of devices currently connected. If **FT_LIST_NUMBER_ONLY** bit is set in **dwFlags**, the parameter **pvArg1** is interpreted as a pointer to a **DWORD** location to store the number of devices currently connected.
It can be used to return device string information. If **FT_OPEN_BY_SERIAL_NUMBER** bit is set in **dwFlags**, the serial number string will be returned from this function. If **FT_OPEN_BY_DESCRIPTION** bit is set in **dwFlags**, the product description string will be returned from this function. If neither of these bits is set, the serial number string will be returned by default. It can be used to return device string information for a single device. If **FT_LIST_BY_INDEX** bit is set in **dwFlags**, the parameter **pvArg1** is interpreted as the index of the device, and the parameter **pvArg2** is interpreted as a pointer to a buffer to contain the appropriate string. Indexes are zerobased, and the error code **FT_DEVICE_NOT_FOUND** is returned for an invalid index.
It can be used to return device string information for all connected devices. If **FT_LIST_ALL** bit is set in **dwFlags**, the parameter **pvArg1** is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings, and the parameter **pvArg2** is interpreted as a pointer to a **DWORD** location to store the number of devices currently connected. Note that, for **pvArg1**, the last entry in the array of pointers to buffers should be a **NULL** pointer so the array will contain one more location than the number of devices connected.

### FT_Open

**Description** Opens the device and return a handle which will be used for subsequent accesses.

**Syntax** FT_STATUS **FT_Open** (int *iDevice*, FT_HANDLE **ftHandle*)

**Parameters**
*iDevice* indicates the number of the device to be opened. Must be 0 if only one device is attached. For multiple devices 1, 2 etc.
*ftHandle* Pointer to a variable of type FT_HANDLE where the handle will be stored. This handle must be used to access the device.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code

**Note** Although this function can be used to open multiple devices by setting iDevice to 0, 1, 2 etc. there is no ability to open a specific device. To open named devices, use the function **FT_OpenEx.** With the **FT_OpenEx** function (not described in this user manual) it is possible to open a device also trough its *serial number* or

trough its description. For further information, please contact **LASERPOINT.srl.**

### FT_Close

**Description** Closes the communication with a open device.

**Syntax** FT_STATUS **FT_Close** (FT_HANDLE *ftHandle*)

**Parametres**
***ftHandle*** pointer to the communication *handle* of the device to close.
**Return Value** FT_OK if successful, otherwise the return value is an FT error code

### FT_Read

**Description** Reads a string from the device.

**Syntax** FT_STATUS **FT_Read** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*, LPDWORD *lpdwBytesReturned*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the device to read.
***lpBuffer*** pointer to the buffer that receives the data from the device.
***DwBytesToRead*** Number of bytes to be read from the device.
***lpdwBytesReturned*** Pointer to a variable of type DWORD which receives the number of bytes read from the device.

**Return Value** FT_OK if successful, FT_IO_ERROR otherwise.

**Note FT_Read** always returns the number of bytes read in **lpdwBytesReturned**. This function does not return until **dwBytesToRead** have been read into the buffer. The number of bytes in the receive queue can be determined by calling **FT_GetStatus** or **FT_GetQueueStatus**, and passed to **FT_Read** as **dwBytesToRead** so that the function reads the device and returns immediately. When a read timeout value has been specified in a previous call to **FT_SetTimeouts**, **FT_Read** returns when the timer expires or **dwBytesToRead** have been read, whichever occurs first. If the timeout occurred, **FT_Read** reads available data into the buffer and returns **FT_OK**. An application should use the function return value and **lpdwBytesReturned** when processing the buffer. If the return value is **FT_OK**, and l**pdwBytesReturned** is equal to **dwBytesToRead** then **FT_Read** has completed normally. If the return value is **FT_OK**, and **lpdwBytesReturned** is less then **dwBytesToRead** then a timeout has occurred, and the read has been partially completed. Note that if a timeout occurred and no data was read, the return value is still **FT_OK**. A return value of **FT_IO_ERROR** suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

### FT_Write

**Description** Writes a string to the device.

**Syntax** FT_STATUS **FT_Write** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the device to write.
***lpBuffer*** pointer to the buffer which contains the bytes to be written in the device.
***DwBytesToWrite*** number of bytes to write to the device.
***lpdwBytesWritten*** pointer to a variable of type DWORD which receives the number of bytes written to the device

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_ResetDevice
**Description** Sends a Reset command to the device.

**Syntax** FT_STATUS **FT_ResetDevice** (FT_HANDLE *ftHandle*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the device to reset .

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetBaudRate
**Description** Sets the *baudrate* for the device.

**Syntax** FT_STATUS **FT_SetBaudRate** (FT_HANDLE *ftHandle*, DWORD *dwBaudRate*)

**Parameters**
***FtHandle*** pointer to the communication *handle* of the device to set out.
***dwBaudRate*** value of the *baudrate* to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetDataCharacteristics
**Description** Sets the data characteristics for the device**.**

**Syntax** FT_STATUS **FT_SetDataCharacteristics** (FT_HANDLE *ftHandle*, UCHAR *uWordLength*, UCHAR *uStopBits*, UCHAR *uParity*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the device to set out .
***uWordLength*** number of *bits* per word. It must set as *FT_BITS_8* (in the case of 8 bit schosen) or as *FT_BITS_7* (in the case of 7 bits chosen).
***uStopBits*** number of stop *bits*. It must set as *FT_STOP_BITS_1* (when one stop bit is requested) or as *FT_STOP_BITS_2* (when two stop bits are requested).
***uParity*** number of parity *bits*. It must set as *FT_PARITY_NONE* (no parity bit) or as *FT_PARITY_ODD* (parity bit is odd) or as *FT_PARITY_EVEN* (parity bit is even) or as *FT_PARITY_MARK* (always high parity bit) or as *FT_PARITY_SPACE* (always low parity bit).

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetFlowControl
**Description** Sets the flow control the chip serial communication of chip USB/RS232.

**Syntax** FT_STATUS **FT_SetDataCharacteristics** (FT_HANDLE *ftHandle*, USHORT *usFlowControl*, UCHAR *uXon*, UCHAR *uXoff*)

**Parameters**
***FtHandle*** pointer to the communication *handle* of the device to set out.
***usFlowControl*** set the kind of flow control. It must be set as *FT_FLOW_NONE* (no flow control) or as *FT_FLOW_RTS_CTS* (*hardware* RTS/CTS flow control) or as *FT_FLOW_DTR_DSR* (*hardware* DTR/DSR flow control) or as *FT_FLOW_XON_XOFF* (*software* XON/XOFF flow control)
***uXon*** shows the character uses as Xon signal. It must be set only when the flow control is *software* XON/XOFF kind (otherwise, it must be set as zero).
***uXoff*** shows the character uses as Xoff signal. It must be set only when the flow control is *software* XON/XOFF kind (otherwise, it must be set as zero).

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetDTR

**Description** Sets the Data Terminal Ready (DTR) control signal. (Data Terminal Ready).

**Syntax** FT_STATUS **FT_SetDTR** (FT_HANDLE *ftHandle*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the DTR device to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_ClrDTR

**Description** This function clears the Data Terminal Ready (DTR) control signal (*Data Terminal Ready*).

**Syntax** FT_STATUS **FT_ClrDTR** (FT_HANDLE *ftHandle*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the DTR device to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetRTS

**Description** Sets the Request To Send (RTS) control signal. (Request To Send).

**Syntax** FT_STATUS **FT_SetDTR** (FT_HANDLE *ftHandle*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the RTS device to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_ClrRTS

**Description** Clears the Request To Send (RTS) control signal (*Request To Send*).

**Syntax** FT_STATUS **FT_SetDTR** (FT_HANDLE *ftHandle*)

**Parameters**
***FtHandle*** pointer to the communication *handle* of the RTS device to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_SetTimeouts

**Description** Sets the read and write timeouts for the device.

**Syntax** FT_STATUS **FT_SetBaudRate** (FT_HANDLE *ftHandle*, DWORD *dwReadTimeout*, DWORD *dwWriteTimeout*)

**Parameters**
***FtHandle*** pointer to the communication *handle* of the device to set out .
***dwReadTimeout*** value of the Read timeout, in milliseconds, to set out.
***dwWriteTimeout*** value of the Write timeout, in milliseconds, to set out.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

### FT_GetQueueStatus

**Description** Shows the number of characters in the receive queue.

**Syntax** FT_STATUS **FT_GetQueueStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*)

**Parameters**
***FtHandle*** pointer to the communication *handle* of the device to set out .
***lpdwAmountInRxQueue*** Pointer to a variable of type DWORD which receives the number of characters in the receive queue.

**Return Value** FT_OK if successful, otherwise the return value is an FT error code.

## FT_GetStatus

**Description** Shows the device status including number of characters in the receive queue, number of characters in the transmit queue, and the current event status.

**Syntax** FT_STATUS **FT_GetStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue* , LPDWORD *lpdwAmountInTxQueue*, LPDWORD *lpdwEventstatus*)

**Parameters**
***ftHandle*** pointer to the communication *handle* of the device to set out .
***lpdwAmountInRxQueu*** Pointer to a variable of type DWORD which receives the number of characters in the receive queue.
***LpdwAmountInTxQueue*** Pointer to a variable of type DWORD which receives the number of characters in the transmit queue.
***lpdwEventstatus*** Pointer to a variable of type DWORD which receives the current state of the event status.

**Return Value** FT_OK if successful, otherwise t