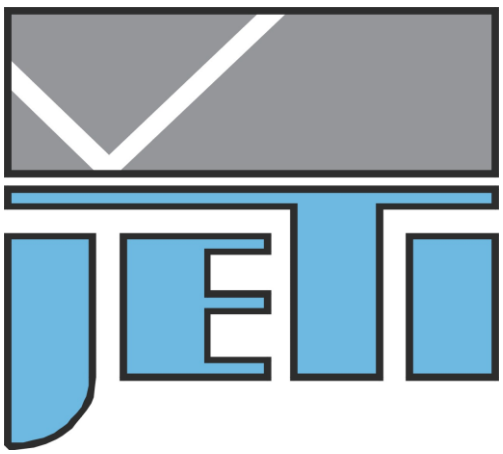
Programmer's Guide

JETI PE6000 Library
pe6000lib.dll

Version 1.0.x



JETI Technische Instrumente GmbH
Göschwitzer Str. 48
D-07745 Jena
Tel.: +49-3641-2329200
Fax: +49-3641-2329201
e-mail: sales@jeti.com
internet: www.jeti.com

Table of contents

1 JETI PE6000 Library Overview

The JETI PE6000 Library provides a software solution for interfacing PE6000 devices from JETI Technische Instrumente GmbH. No firmware command expertise is required. Instead, a simple, high-level Application Program Interface (API) is used to provide complete connectivity. The API is provided in the form of a Windows Dynamic Link Library (DLL). The library can be used by any programming language that can handle DLL's such C/C++, VisualBasic, or LabVIEW.
To get access to the functions the needed DLL files must be copied to the Windows System Folder or to the working directory of the calling application.

Please note that depending on the bit width of the calling application (32bit or 64bit), the corresponding 32- or 64bit DLL must be used.
Furthermore, the corresponding USB library libusb-1.0.dll must be installed in the same folder as pe6000lib.dll in the appropriate bit depth.

2 Function Reference

Convention for calling : __stdcall

Type	Size in Bit	Minimum	Maximum
int	32	-2^{31}	$2^{31}-1$
unsigned int	32	0	$2^{32}-1$
short	16	-2^{15}	$2^{15}-1$
unsigned short	16	0	$2^{16}-1$
char	8	-2^{7}	$2^{7}-1$
unsigned char	8	0	2^{8-1}
float (IEEE standard)	32	-3.40282E+38	3.40282E+38

2.1 PE6000_GetDLLVersion

This function returns the current version number of the pe6000lib DLL.

Prototype

int PE6000_GetDLLVersion (unsigned short *wMajorVersion, unsigned short *wMinorVersion, unsigned short *wBuildNumber)

Parameters

Input

Name	Type	Description	Call
wMajorVersion	unsigned short *	address of a unsigned short variable that will contain the major version	By reference
wMinorVersion	unsigned short *	address of a unsigned short variable that will contain the minor version	By reference
wBuildNumber	unsigned short *	address of a unsigned short variable that will contain the build number	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.2 PE6000_GetNumDevices

This function returns the number of connected PE6000 devices.

Prototype

int PE6000_GetNumDevices (int* iNumDevices)

Parameters

Input:

Name	Type	Description	Call
iNumDevices	int *	Pointer to a variable where the number of devices will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.3 PE6000_GetDeviceInfo

This function returns device info (serial numbers) of the connected PE6000 devices.
Before opening a specific device, PE6000_GetNumDevices() can be used to determine the count of connected PE6000. If more than one device was found, this function can be used to get information by using a zero-based index.

Prototype

int PE6000_GetDeviceInfo (int iDeviceNum, char * cBoardSN, char * cSpecSN, char * cDevSN)

Parameters

Input

Name	Type	Description	Call
iDeviceNum	int	number of the connected PE6000, zero-based index	By value
cBoardSN	char *	address of a char array of 16 characters to store the board serial number string	By reference
cSpecSN	char *	address of a char array of 16 characters to store the spectrometer serial number string	By reference
cDevSN	char *	address of a char array of 16 characters to store the device serial number string	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.4 PE6000_OpenDeviceEx

This function opens a specific PE6000 device..
If PE6000_GetNumDevices() has found more than one device, a specific device can be opened with the help of the corresponding index.

Prototype

int PE6000_OpenDeviceEx (int iDeviceNum)

Parameters

Input

Name	Type	Description	Call
iDeviceNum	int	number of the connected PE6000, zero-based index	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.5 PE6000_OpenDevice

This function opens a PE6000 device. If several devices are connected, the first one found is opened.

Prototype

int PE6000_OpenDevice (void)

Parameters

Input: none

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.6 PE6000_CloseDevice

This function will close the device previously opened with PE6000_OpenDevice()

Prototype

int PE6000_CloseDevice (void)

Parameters

Input: none

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.7 PE6000_GetUSBSpeed

This function will return the currently used USB speed class as reported by the device. Please note that for maximum throughput USB Super Speed should be returned.

Prototype

int PE6000_GetUSBSpeed (void)

Parameters

Input: none

Return Value

Type	Description
int	0 – could not determine USB speed 1 – USB Full Speed 2 – USB High Speed 3 – USB Super Speed

2.8 PE6000_GetID

This function will return the device ID string.

Prototype

int PE6000_GetID (char * cID)

Parameters

Input

Name	Type	Description	Call
cID	char *	address of a char array of 256 characters to store the device ID string	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.9 PE6000_GetFirmwareVersion

This function will return the device firmware string.

Prototype

int PE6000_GetFirmwareVersion (char * cFirmware)

Parameters

Input

Name	Type	Description	Call
cFirmware	char *	address of a char array of 256 characters to store the firmware version string	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.10 PE6000_GetSerialDevice

This function will return the device serial numbers as strings.

Prototype

int PE6000_GetSerialDevice (char * cBoardSerialNr, char * cSpecSerialNr, char * cDeviceSerialNr)

Parameters

Input

Name	Type	Description	Call
cBoardSerialNr	char *	address of a char array of 16 characters to store the board serial number string	By reference
cSpecSerialNr	char *	address of a char array of 16 characters to store the spectrometer serial number string	By reference
cDeviceSerialNr	char *	address of a char array of 16 characters to store the device serial number string	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.11 PE6000_SetSerialDevice

This function will set the device serial numbers as strings.

Prototype

int PE6000_SetSerialDevice (char * cBoardSerialNr, char * cSpecSerialNr, char * cDeviceSerialNr)

Parameters

Input

Name	Type	Description	Call
cBoardSerialNr	char *	address of a char array of 16 characters with the board serial number string to set	By reference
cSpecSerialNr	char *	address of a char array of 16 characters with the spectrometer serial number string to set	By reference
cDeviceSerialNr	char *	address of a char array of 16 characters with the device serial number string to set	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.12 PE6000_GetPDAGain

This function will return the PDA gain setting. It can be set to low (0) or high (1).

Prototype

int PE6000_GetPDAGain (unsigned char * bPDAGain)

Parameters

Input

Name	Type	Description	Call
bPDAGain	unsigned char *	Pointer to a variable where the PDA gain range will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.13 PE6000_SetPDAGain

This function will set PDA gain. It can be set to low (0) or high (1).

Prototype

int PE6000_SetPDAGain (unsigned char bPDAGain)

Parameters

Input

Name	Type	Description	Call
bPDAGain	unsigned char	the PDA gain setting valid values: 0 (low) or 1 (high)	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.14 PE6000_GetADCVoltage

This function will return the ADC input voltage range in V.

Prototype

int PE6000_GetADCVoltage (unsigned char * bADCV)

Parameters

Input

Name	Type	Description	Call
bADCV	unsigned char *	Pointer to a variable where the ADC input voltage range will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.15 PE6000_SetADCVoltage

This function will set the ADC input voltage range in V.

Prototype

int PE6000_SetADCVoltage (unsigned char bADCV)

Parameters

Input

Name	Type	Description	Call
bADCV	unsigned char	the input voltage range in [V], valid values: 2 or 4	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.16 PE6000_GetADCOffset

This function will return the ADC offset value in mV.

Prototype

int PE6000_GetADCOffset (short * sOffset)

Parameters

Input

Name	Type	Description	Call
sOffset	short *	Pointer to a variable where the ADC offset value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.17 PE6000_SetADCOffset

This function will set the ADC offset value in mV.

Prototype

int PE6000_SetADCOffset (short sOffset)

Parameters

Input

Name	Type	Description	Call
sOffset	short	the ADC offset value in [mV], valid range: -300 to 300	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.18 PE6000_GetADCGain

This function will return the ADC gain value.

Prototype

int PE6000_GetADCGain (float * fGain)

Parameters

Input

Name	Type	Description	Call
fGain	float *	Pointer to a variable where the ADC gain value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.19 PE6000_SetADCGain

This function will set the ADC gain value in mV.

Prototype

int PE6000_SetADCGain (float fGain)

Parameters

Input

Name	Type	Description	Call
fGain	float	the ADC gain value, valid range: 1.0 to 5.0	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.20 PE6000_GetTint

This function will return the currently used integration time in ms.

Prototype

int PE6000_GetTint (float * fTint)

Parameters

Input

Name	Type	Description	Call
fTint	float *	Pointer to a variable where the integration time value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.21 PE6000_SetTint

This function will set the integration time in ms used for the following measurements.

Prototype

int PE6000_SetTint (float fTint)

Parameters

Input

Name	Type	Description	Call
fTint	float	the integration time in ms, valid range: 0.001 to 1000.0	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.22 PE6000_GetPixelCount

This function will return the sensor pixel count.

Prototype

int PE6000_GetPixelCount (unsigned short * wPixelCount)

Parameters

Input

Name	Type	Description	Call
wPixelCount	unsigned short *	Pointer to a variable where the sensor pixel count will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.23 PE6000_SetPixelCount

This function will set the sensor pixel count value.

Prototype

int PE6000_SetPixelCount (unsigned short wPixelCount)

Parameters

Input

Name	Type	Description	Call
wPixelCount	unsigned short	the pixel count value, valid values: 128, 256, 512	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.24 PE6000_GetFrameSize

This function will return the currently used frame size (number of spectra per frame).
To achieve the highest possible transmission speed, several measured spectra are combined into one frame before transmission. This parameter indicates how many spectra are contained in one frame.

Prototype

int PE6000_GetFrameSize (unsigned char * bFrameSize)

Parameters

Input

Name	Type	Description	Call
bFrameSize	unsigned char *	pointer to a variable where the frame size value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.25 PE6000_SetFrameSize

This function will set the frame size for the following measurements.
To achieve the highest possible transmission speed, several measured spectra are combined into one frame before transmission. This parameter indicates how many spectra are contained in one frame.
The valid range depends on the number of sensor pixels.

Prototype

int PE6000_SetFrameSize (unsigned char bFrameSize)

Parameters

Input

Name	Type	Description	Call
bFrameSize	unsigned char	frame size (number of spectra per frame), valid ranges: (128 pixel) – 32 to 192 (256 pixel) – 16 to 96 (512 pixel) – 8 to 48	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.26 PE6000_GetTemperatureSource

This function will return the temperature sensor to be used for temperature measurement.
It can be set to internal PDA sensor (0) or external board sensor (1).

Prototype

int PE6000_GetTemperatureSource (unsigned char * bTempSource)

Parameters

Input

Name	Type	Description	Call
bTempSource	unsigned char *	pointer to a variable where the temperature source value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.27 PE6000_SetTemperatureSource

This function will set the temperature sensor to be used for temperature measurement.
It can be set to internal PDA sensor (0) or external board sensor (1).

Prototype

int PE6000_SetTemperatureSource (unsigned char bTempSource)

Parameters

Input

Name	Type	Description	Call
bTempSource	unsigned char	temperature sensor setting: 0 – internal PDA sensor 1 – external board sensor	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.28 PE6000_ParaSave

This function saves all set parameters in the internal flash memory.

Prototype

int PE6000_ParaSave (void)

Parameters

Input: none

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.29 PE6000_MeasureTemperature

This function returns the measured temperature in °C.

Prototype

int PE6000_MeasureTemperature (float * fTemperature)

Parameters

Input

Name	Type	Description	Call
fTemperature	float *	pointer to a variable where the temperature value will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.30 PE6000_StartFetching

This function starts the collection of the measurements.
Measurements are transmitted together as frames. The number of frames can be specified here (uiCount). The number 0 means that the transmission is continuous until PE6000_StopFetching() stops the transmission.
To minimise the necessary USB queries, several frames can also be combined into one buffer (sizeBuff). sizeBuff must always be greater than or equal to the number of frames (uiCount), unless continuous transmission is used.
After transmission (whether continuous or with a certain number of frames), PE6000_StopFetching() must be called to release the memory required for buffering.

Prototype

int PE6000_StartFetching (unsigned int uiCount, unsigned int sizeBuff)

Parameters

Input

Name	Type	Description	Call
uiCount	unsigned int	number of frames to fetch	By value
sizeBuff	unsigned int	number of frames per buffer	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.31 PE6000_StopFetching

This function stops the continuous collection of measurements.
If only a certain number of measurements have been transferred (see PE6000_StartFetching()), this function must also be called to release the memory required for buffering.

Prototype

int PE6000_StopFetching (void)

Parameters

Input: none

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.32 PE6000_BufferStatus

This function returns the status of the internal DLL receive buffer. When the buffer is ready to be fetched, the function returns a non-zero return value.

Prototype

int PE6000_BufferStatus (unsigned int * transStatus, unsigned int * corruptTrans, unsigned int * discardTrans)

Parameters

Input

Name	Type	Description	Call
transStatus	unsigned int *	last USB transaction status (normally 0 for valid transaction, otherwise can indicate error or interrupt of communication)	By reference
corruptTrans	unsigned int *	counter. Any time USB transaction completes with unexpected number of bytes this counter would increment. Normally should be 0	By reference
discardTrans	unsigned int *	counter. Anytime top level application does not read out data from relevant section of buffer in time (buffer is overwritten) this counter is incremented. Normally should be 0	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.33 PE6000_BufferReturn

Returns the buffer values.
If buffer status is not 0, this function would return next available section of buffer for processing.
Always check buffer status (PE6000_BufferStatus()) prior using this function.
Function may return undefined data and cause program to crash if check is not performed beforehand!

Prototype

int PE6000_BufferReturn (char * bufferReturn)

Parameters

Input

Name	Type	Description	Call
bufferReturn	char *	returned buffer	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.34 PE6000_GetOut1

This function will return the OUT1/LED1 status.

Prototype

int PE6000_GetOut1 (unsigned char * bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char *	Pointer to a variable where the OUT1/LED1 status will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.35 PE6000_SetOut1

This function will set the OUT1/LED1 status.
It can be set to 0 (OUT1 low, LED1 on) or 1 (OUT1 high, LED1 off).

Prototype

int PE6000_SetOut1 (unsigned char bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char	the OUT1/LED1 setting valid values: 0 – OUT1 low, LED1 on 1 – OUT1 high, LED1 off	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.36 PE6000_GetOut2

This function will return the OUT2/LED2 status.

Prototype

int PE6000_GetOut2 (unsigned char * bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char *	Pointer to a variable where the OUT2/LED2 status will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.37 PE6000_SetOut2

This function will set the OUT2/LED2 status.
It can be set to 0 (OUT2 low, LED2 on) or 1 (OUT2 high, LED2 off).

Prototype

int PE6000_SetOut1 (unsigned char bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char	the OUT2/LED2 setting valid values: 0 – OUT2 low, LED2 on 1 – OUT2 high, LED2 off	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.38 PE6000_GetOut3

This function will return the OUT3/LED3 status.

Prototype

int PE6000_GetOut3 (unsigned char * bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char *	Pointer to a variable where the OUT3/LED3 status will be stored	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.39 PE6000_SetOut3

This function will set the OUT3/LED3 status.
It can be set to 0 (OUT3 low, LED3 on) or 1 (OUT3 high, LED3 off).

Prototype

int PE6000_SetOut1 (unsigned char bOut)

Parameters

Input

Name	Type	Description	Call
bOut	unsigned char	the OUT3/LED3 setting valid values: 0 – OUT3 low, LED3 on 1 – OUT3 high, LED3 off	By value

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.40 PE6000_WriteDataBlock

This function writes a data block with 256 byte of user data to the internal EEPROM.
Up to 128 data blocks are available, therefore the user can save up to 32KB user data.

Prototype

int PE6000_WriteDataBlock (unsigned char bBlockNr, unsigned char* cData)

Parameters

Input

Name	Type	Description	Call
bBlockNr	unsigned char	block number to write, valid range: 0…127	By value
cData	unsigned char *	address of a char array of 256 characters to write to EEPROM	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

2.41 PE6000_ReadDataBlock

This function reads a data block with 256 byte of user data from the internal EEPROM.

Prototype

int PE6000_ReadDataBlock (unsigned char bBlockNr, unsigned char* cData)

Parameters

Input

Name	Type	Description	Call
bBlockNr	unsigned char	block number to read, valid range: 0…127	By value
cData	unsigned char *	address of a char array of 256 characters to read from EEPROM	By reference

Return Value

Type	Description
int	0 on success, nonzero otherwise

3 Data Format

The returned buffers from the function PE6000_BufferReturn() has the following form :

- 1024 Byte header
  - The first 8 bytes are an unsigned 64bit timestamp in nanoseconds
  - The next 4 bytes are an unsigned 32bit error counter, indicating lost spectra during continuous operation
  - The next 4 bytes are a 32bit float number indicating the temperature of a connected NTC
  - The remaining bytes of the header are currently unused and could contain further informations in the future
- Frame data
  - Contains the spectral data of the frame
  - The size depends on the frame size (PE6000_GetFrameSize()) and the pixel number (PE6000_GetPixelCount()).

4 Service

In case of any questions or technical problems please contact:

JETI Technische Instrumente GmbH
Göschwitzer Str. 48
D-07745 Jena
Tel. +49 3641 23292 00
e-mail : sales@jeti.com
Internet: www.jeti.com

August 24, 2023