Simply Brighter

(In Canada)
2343 Brimley Road
Suite 868
Toronto, Ontario M1S 3L6
CANADA
Tel: 1-416-840 4991
Fax: 1-416-840 6541

(In US)
1032 Serpentine Lane
Suite 113
Pleasanton, CA 94566
USA
Tel: 1-925-218 1885
Email: sales@mightex.com

BioLED Light Source Control Module SDK Manual

Version: 1.2.1

Dec. 22, 2020

Relevant Products

Part Numbers
BLS-XX0X-U/S

Revision History

Revision	Date	Author	Description
1.0.0	Feb. 03, 2012	William Dai	Initial Revision
1.0.0	2014/4/22	William Dai	Add support for x64
1.1.0	2014/11/3	William Dai	Add get EPSM counter command, Send String command.
1.1.1	2015/05/04	W.D	Updates some functions
1.1.2	2015/07/30	LPD	Add get Channel Title function
1.2.0	2017/12/5	WD	Bundled updates.
1.2.1	2020/12/22	LPD	Add SendCommand function. Add Python module and related.

Mightex BioLED Control Module is designed to drive various kinds of Mightex Light heads. Windows based PC software is also provided for user to control the output lighting of light head.

IMPORTANT:

The SDK is for BioLED Control Module (BLS-XXXX-U) only.

SDK FILES:

The SDK includes the following files:

\LIB directory:
Mightex_BLSDriver_SDK.h --- Header files for all data prototypes and DLL export functions.
\x86, and \x64 sub directories, which contains following dll files in x86 and x64 platform respectively.
Mightex_BLSDriver_SDK.dll --- DLL file exporting all API functions. (in "cdecl" call convention)
Mightex_BLSDriver_SDK_stdcall.dll---DLL file exporting all API functions. (in "stdcall" call convention)
Mightex_BLSDriver_SDK.lib --- Import lib file, user may use it for VC++ development.
Mightex_BLSDriver_SDK.cp37-xx.pyd -- DLL file for python3.
Hiddll.dll --- DLL file used by "Mightex_BLSDriver_SDK.dll" and "Mightex_BLSDriver_SDK.*.pyd".

\Documents directory:
Mightex bioLED Controller SDK Manual.pdf (This manual).

\Examples directory
\Delphi --- Delphi project example.
\VC++ --- VC++ project example.
\VB_Application – VB project example, note that for VB application, user **MUST** use the "Mightex_BLSDriver_SDK_Stdcall.dll", which is included in the Lib directory(x86 or x64).
\Python – Python3.x example

Note that these examples are for demonstration of the DLL functions only, device fault situations are not handled in these examples, user should handle them properly.

HEADER FILE:

The C header file "Mightex_BLSDriver_SDK.h" is as following:

```
typedef int SDK_RETURN_CODE;

#ifdef SDK_EXPORTS
#define SDK_API extern "C" __declspec(dllexport) SDK_RETURN_CODE _cdecl
#else
#define SDK_API extern "C" __declspec(dllimport) SDK_RETURN_CODE _cdecl
```

```
#endif

#define  MAX_PULSE_COUNT    21  // For SX Modules, it's 3 instead of 128.

#defineDISABLE_MODE     0
#define NORMAL_MODE    1
#define TRIGGER_MODE    3

SDK_API MTUSB_BLSDriverInitDevices( void );
SDK_API MTUSB_BLSDriverOpenDevice( int DeviceIndex );
SDK_API MTUSB_BLSDriverCloseDevice( int DevHandle );
SDK_API  MTUSB_BLSDriverGetSerialNo(  int DevHandle, unsigned char *SerNumber, int
Size );
SDK_API MTUSB_BLSDriverGetModuleType(int DevHandle);
SDK_API MTUSB_BLSDriverGetChannels( int DevHandle );
SDK_API  MTUSB_BLSDriverGetChannelTitle(int  DevHandle,  int  Channel,  unsigned  char
*ChnlTitle);
SDK_API MTUSB_BLSDriverSetMode( int DevHandle, int Channel, int Mode );
SDK_API MTUSB_BLSDriverSetNormalCurrent( int DevHandle, int Channel, int Current);
SDK_API  MTUSB_BLSDriverSetPulseProfile(  int  DevHandle,  int  Channel,  int  Polarity,  int
PulseCnt, int ReptCnt);
SDK_API  MTUSB_BLSDriverSetPulseDetail(  int  DevHandle,  int  Channel,  int  PulseIndex,  int
Time0, int Time1, int Time2, int Curr0, int Curr1, int Curr2);
SDK_API  MTUSB_BLSDriverSetFollowModeDetail(  int  DevHanlde,  int  Channel,  int  ION,  int
IOFF);
SDK_API MTUSB_BLSDriverSoftStart( int DevHandle, int Channel );
SDK_API MTUSB_BLSDriverResetDevice( int DevHandle );
SDK_API MTUSB_BLSDriverStorePara( int DevHandle );
SDK_API MTUSB_BLSDriverGetEPSMCount(int DevHandle, int Channel);
SDK_API MTUSB_BLSDriverSendCommand( int DevHandle, char *Command);
```

For python, the function are declared as:

```
MT_BLSInitDevices()
MT_BLSOpenDevice( DeviceIndex )
MT_BLSCloseDevice( DevHandle )
MT_BLSGetSerialNo( DevHandle )
MT_BLSGetModuleType( DevHandle )
MT_BLSGetChannels( DevHandle )
MT_BLSGetChannelTitle(  DevHandle, Channel )
MT_BLSSetMode( DevHandle, Channel, Mode )
MT_BLSSetNormalCurrent( DevHandle, Channel, Current )
MT_BLSSetPulseProfile( DevHandle,Channel,Polarity,PulseCnt,  ReptCnt )
MT_BLSSetPulseDetail( DevHandle, Channel, PulseIndex,Time0, Time1, Time2, Curr0, Curr1,
Curr2 )
MT_BLSSetFollowModeDetail(  DevHandle, Channel, ION, IOFF )
MT_BLSSoftStart( DevHandle, Channel )
MT_BLSResetDevice( DevHandle )
MT_BLSStorePara( DevHandle )
MT_BLSGetEPSMCount( DevHandle,Channel )
```

MT_BLSSendCommand(DevHandle, Command)

EXPORT Functions:
Note: The above exported functions are explained here both in C and Python prototypes.

C: SDK_API MTUSB_BLSDriverInitDevices (void);
Python: MT_BLSInitDevices()
Parameter: None
Return: it returns the number of BIOLED control modules currently connected to PC. When there's no device connected, it's ZERO.
Note: User should always invoke this function before calling any other functions.

C: SDK_API MTUSB_BLSDriverOpenDevice (int DeviceIndex);
Python: MT_BLSOpenDevice(DeviceIndex)
Parameter: DeviceIndex – the index is from [0, (TotalDevices–1)], here TotalDevices is the value returned by the function of MTUSB_ BLSDriverInitDevices(void).
Return: it returns HANDLE of this device, the HANDLE can be used as first parameter (DevHandle) of all other APIs to operate this control module.
-1 means Device open failed, this might be caused by device error.
Note: User should always open the device before operating on the device, note that ZERO is also a valid value for handle.

C: SDK_API MTUSB_BLSDriverCloseDevice (int DevHandle);
Python: MT_BLSCloseDevice (DevHandle)
Parameters: DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.
Return: it returns –1 means this function call failed, otherwise device is closed correctly.
Note: User should close any opened devices, while finishing operation of the device.

C: SDK_API MTUSB_BLSDriverGetSerialNo (int DevHandle, unsigned char *SerNumber, int Size);
Python: MT_BLSGetSerialNo (DevHandle)
Parameters:
DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.
SerNumber – The pointer to char buffer which will be filled with module serial number.
Size – The Size of the SerNumber buffer, the buffer should at least holds 16 characters.
Return for C: it returns –1 means this function call failed, otherwise the call is successful.
Return for Python: it returns device's serial No. in type of bytes if the call succeeded, and -1 if the call failed.

C: SDK_API MTUSB_BLSDriverGetModuleType (int DevHandle);
Python: MT_BLSGetModuleType(DevHandle)
Parameters:
DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.
Return: The device type of this device.
Note: Device Types

```
AA_MODULE     =  0;
SA_MODULE     =  2;
PL_MODULE     =  13;  //BLS-PL
IO_MODULE     =  14;  //BLS-IO
```

C: SDK_API MTUSB_BLSDriverGetChannels (int DevHandle);
Python: MT_BLSGetChannels (DevHandle)
Parameters:
DevHandle – The device handle returned by **MTUSB_BLSDriverOpenDevice**.
Return: it returns –1 means this function call failed, otherwise the return value is the channel number of this device.

C: SDK_API MTUSB_BLSDriverGetChannelTitle(int DevHandle, int Channel, unsigned char *ChnlTitle);
Python: MT_BLSGetChannelTitle(DevHandle, Channel)
Parameters:
DevHandle – The device handle returned by **MTUSB_BLSDriverOpenDevice**.
Channel – The channel number of the channel user wants to operate, it's ONE based, that is, ONE means the first channel of the device.
ChnlTitle – The pointer to an unsigned char buffer for holding the channel title, which should hold at least 64 characters.
Return for C: it returns negative value means this function call failed,
Otherwise the return value is the channel title length (of characters).
Return for Python: it returns channel title in type of bytes if function call succeeded, and -1 if function call failed.

C: SDK_API MTUSB_BLSDriverSetMode (int DevHandle, int Channel, int Mode);
Python: MT_BLSSetMode(DevHandle, Channel, Mode)
Parameters:
DevHandle – The device handle returned by **MTUSB_BLSDriverOpenDevice** .
Channel – The channel number of the channel user wants to operate, it's ONE based, that is, ONE means the first channel of the device.
Mode – The setting mode:
0 --- DISABLE
1 --- NORMAL
2 --- Not Used, Reserved.
3 --- TRIGGER (OR it can be called "Pulse Mode").
Return: it returns –1 means this function call fails because user uses an invalid handle.
returns 1 means device error occurred during the API invoking,
otherwise the call is successful.

C: SDK_API MTUSB_BLSDriverSetNormalCurrent (int DevHandle, int Channel, int Current);
Python: MT_BLSSetNormalCurrent (DevHandle, Channel, Current)
Parameters:
DevHandle – The device handle returned by **MTUSB_BLSDriverOpenDevice** .
Channel – The channel number of the channel user wants to operate, it's ONE based.
Current – The setting current of this channel, note it's in "0.1%" unit, e.g. 500 means current set to 50.0% of the defined Maximum current of this channel.
Return: it returns –1 means this function call fails because user uses an invalid handle.
returns 1 means device error occurred during the API invoking,
otherwise the call is successful.

C: SDK_API MTUSB_BLSDriverSetPulseProfile(int DevHandle, int Channel, int Polarity, int PulseCnt, int ReptCnt);

Python: MT_BLSSetPulseProfile(DevHandle, Channel, Polarity, PulseCnt, ReptCnt)

Parameters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Channel – The channel number of the channel user wants to operate, it's ONE based.

Polarity – When an external trigger signal is used to start the output of the defined pulse combination, user can use it to define the assertion edge of the trigger signal as:

1: Falling edge on trigger signal (BND connector) will assert the start of output.

0: Rising edge on trigger signal (BND connector) will assert the start of output.

PulseCnt – The total pulse number of a pulse profile. The pulse count should not be biggger than 21, which is defined in the header file as *MAX_PULSE_COUNT.*

ReptCnt – The pulse profile can be repeated output, the repeat number is defined by ReptCnt.

Return: it returns -1 means this functions call fails because user uses an invalid handle.

returns 1 means device error occurred during the API invoking,

Otherwise the call is successful.

C: SDK_API MTUSB_BLSDriverSetPulseDetail(int DevHandle, int Channel, int PulseIndex, int Time0, int Time1, int Time2, int Curr0, int Curr1, int Curr2)*;*

Python: MT_BLSSetPulseDetail(DevHandle, Channel, PulseIndex, Time0, Time1, Time2, Curr0, Curr1, Curr2)

Parameters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Channel – The channel number of the channel user wants to operate, it's ONE based.

PulseIndex – the pulse index of a pulse profile, it is Zero based, which means 0 the first pulse, PulseCnt-1 means the last pulse.

Time0, Time1, Time2 – the 3 time sections of a pulse, note it's in "us" unit, but all values should be multiples of "20us".

Curr0, Curr1, Current2 - The current of each time section. Note it's in "0.1%" unit, e.g. 500 means current set to 50.0% of the defined Maximum current of this channel.

User might refer to User manual (Figure 4.13) for the Pulse definition.

Return: it returns -1 means this function call fails because user uses an invalid handle, invalid channel number or invalid pulse index, or invalid pulse parameter(s).

returns 1 means device error occurred during the API invoking,

Otherwise the call is successful.

Notes: 1) for pulse definition, please refer to application user manual.

2)for setting pulse profile, user should always invoke **MTUSB_BLSDriverSetPulseProfile** and followed by **BLSDriverSetPulseDetail.** After all the pulses have been set, *When* ***the channel is set as "TRIGGER" mode(Pulse Mode),*** user might invoke the **MTUSB_BLSDriverSoftStart** to start output of the defined pulses on this channel, or an external Trigger can start the output too.

3) When I2 is bigger than 100.0% (I2 >= 1000), please refer to the application user manual IntelliPulsing Rules.

C: SDK_API MTUSB_BLSDriverSetFollowModeDetail(int DevHanlde, int Channel, int ION, int IOFF)*;*

Python: MT_BLSSetFollowModeDetail(DevHanlde, Channel, ION, IOFF)

Parameters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Channel – The channel number of the channel user wants to operate, it's ONE based.

ION – the current of the trigger in on state. It's in "0.1%" unit, e.g. 500 means 50.0%.

IOFF – the current of the trigger in off state. It's in "0.1%" unit, e.g. 50 means 5.0%.

Return: it returns -1 means this function call fails because user uses an invalid handle or an invalid channel number,

it returns **1 means device error occurred during the API invoking,**

Otherwise the call is successful.

Note: 1). Please refer to application user manual for the details of follower mode definition.

2). Follower mode is a sub-mode of TRIGGER mode (Pulse Mode), when a certain channel is in "TRIGGER" mode, it can works in two flavors:

Pulse Mode --- User can use **MTUSB_BLSDriverSetPulseProfile()** and **MTUSB_BLSDriverSetPulseDetail()** to define the parameters of the pulse combination and let the channel works as "**Pulse Mode**". And the pulses will be output when an defined external assertion occurs OR user invoke the **MTUSB_BLSDriverSoftStart()**.

Follower Mode --- User can use **MTUSB_BLSDriverSetFollowModeDetail()** to define the parameters (Ion and Ioff) of the "Follower Mode" and let the channel works in "Follower Mode".

C: SDK_API MTUSB_BLSDriverSoftStart(int DevHandle, int Channel);
Python: MT_BLSSoftStart(DevHandle, Channel)

Parameteters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Channel – The channel number of the channel user wants to operate, it's ONE based.

Return: it returns -1 means this function call fails because user uses an invalid handle or an invalid channel number.

returns 1 means device error occurred during the API invoking,

Otherwise the call is successful.

Note: When the channel is in TRIGGER mode, After setting the pulse profile, user might invoke this function to start pulse output. This is an equivalent to a valid external trigger assertion, thus it can be called as Soft Trigger too.

C: SDK_API MTUSB_BLSDriverResetDevice (int DevHandle);
Python: MT_BLSResetDevice (DevHandle)

Parameters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Return: it returns –1 means this function call failed, otherwise the call is successful.

Note: it returns –1 means this function call fails because user uses an invalid handle,

it returns 1 means device error occurred during the API invoking,

otherwise the call is successful.

This API is used to reboot the device, it's mainly used for technical service purpose only.

C: SDK_API MTUSB_BLSDriverStorePara(int DevHandle);
Python: MT_BLSStorePara(DevHandle)

Parameters:

DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.

Return: it returns –1 means this function call failed, otherwise the call is successful.

Note: it returns –1 means this function call fails because user uses an invalid handle,

returns 1 means device error occurred during the API invoking,

otherwise the call is successful.

For any parameters set by the above APIs (including the NORMAL mode parameters and TRIGGER mode parameters such as pulse profiles…etc.), they're not stored in device's nv-memory until user invokes this API, calling this API will make the device remember the

parameters even the device is power cycled, user might find this is useful in the scenario there's no host.

C: SDK_API MTUSB_BLSDriverGetEPSMCount(int DevHandle , int Channel);
Python: MT_BLSGetEPSMCount(DevHandle , Channel)
Parameter:
DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.
Channel – The channel number of the channel user wants to operate, it's ONE based.
Return: it returns current device's current EPSM count.
This function is to be used for BLS-PL/IO device to be working in Polygon Mode.

C: SDK_API MTUSB_BLSDriverSendCommand(int DevHandle, char *Command);
Python: MT_BLSSendCommand(DevHandle, Command)
Parameter:
DevHandle – The device handle returned by **MTUSB_ BLSDriverOpenDevice**.
Command – The predefined string command to be sent to device.
Return: it returns: 0 if command is successfully received and handled by device.
: -1 if function called failed.
This function is to be used for special services only.

Using DLL APIs in NI LabVIEW

As the BIOLED Control driver is designed as a HID device, and the SDK(DLL files) provides full sets of APIs for controlling the device. For LabView users, it's recommended to use CLF nodes to call those APIs. The CLF node can be added by right click the "Block Diagram" of a VI, select the "Connectivity|Library&Executables|Call Library Function Node".

With this CLF node, user can link it with a ".DLL" file and a particular export function of this DLL file,
In addition, user should set it's return type and arguments correctly, for the return type and arguments information, please refer to the above description of each API.