Simply Brighter

(In Canada)
111 Railside Road
Suite 201
Toronto, ON M3A 1B2
CANADA
Tel: 1-416-840 4991
Fax: 1-416-840 6541

(In US)
1241 Quarry Lane
Suite 105
Pleasanton, CA 94566
USA
Tel: 1-925-218 1885
Email: sales@mightex.com

Sirius™ **SLC-XXXX-S/U**

# Mightex Sirius™ Multi-Channel LED Controller SDK Description

Version: 1.1.4

Oct.12, 2018

**Relevant Products**

| Part Numbers |
| --- |
| SLC-AA02-U, SLC-AA04-U,SLC-AA02-S,SLC-AA04-S,<br>SLC-AV02-U, SLC-AV04-U,SLC-AV02-S,SLC-AV04-S,<br>SLC-SA02-U, SLC-SA04-U,SLC-SA02-S,SLC-SA04-S,<br>SLC-SV02-U, SLC-SV04-U,SLC-SA02-S,SLC-SV04-S,<br>SLC-MA01-U,SLC-MA02-U,SLC-MA12-U,SLC-MA16-U,<br>SLC-MA12-S,SLC-MA16-S,SLC-CA01-U,SLC-CA02-U,SLC-CA12-U,SLC-CA16-U,SLC-CA12-S,SLC-CA16-S, SLC-MA04-MU, SLC-CA04-MU, SLC-FA02-U, SLC-FA04-U, SLC-FV02-U, SLC-FV04-U,<br>SLC-FA02-S, SLC-FA04-S, SLC-FV02-S, SLC-FV04-S<br>SLC-XA02-U, SLC-XA04-U, SLC-XV02-U, SLC-XV04-U<br>SLC-XA02-S, SLC-XA04-S, SLC-XV02-S, SLC-XV04-S<br>SLC-HA02-S, SLC-HA02-U, SLC-HV02-S, SLC-HV02-U<br>SLC-QA02-S, SLC-QA02-U, SLC-QA04-S,SLC-QA04-U |

## Revision History

| Revision | Date | Author | Description |
|---|---|---|---|
| 1.0.0 | Nov. 18, 2007 | JT Zheng | Initial Revision |
| 1.1.0 | Apr. 18, 2009 | JT Zheng | LabView supporting |
| 1.1.1 | Nov. 12, 2010 | JT Zheng | More return value description |
| 1.1.2 | Nov. 16, 2011 | JT Zheng | Adding SLC-MA04/CA04-MU modules |
| 1.1.3 | Jun. 19, 2012 | JT Zheng | Clarify Strobe RepeatCnt definition |
| 1.1.4 | Oct. 12, 2018 | JT Zheng | New Logo |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Mightex Sirius™ Multi-Channel LED Controller was designed to drive various kinds of LEDs on current market, including Mightex Sirius™ Light Sources, as well as LEDs from other vendors. Windows based PC software is provided for user to control the channel operation easily. External trigger is provided for each channel, allowing real time applications, such as machine vision application, are easily applied.

**IMPORTANT:**

The SDK is for Sirius USB LED Controller ( SLC-XXXX-U) only, for RS232 module, it's recommended to use RS232 command sets for customer application development. Some APIs are only for certain modules, ***please refer to User manual for detailed features of modules***. For example, the "MA/CA" module (SLC-MAxx-U/S or SLC-CAxx-U/S) doesn't have "Trigger mode", so all the related APIs should not be invoked on this module, and for some modules, the steps in Strobe/Trigger mode are only limited to 2, while others allow 128 steps.

**SDK FILES:**

The SDK includes the following files:

\LIB directory:

| | |
|---|---|
| Mightex_LEDDriver_SDK.h | --- Header files for all data prototypes and dll export functions. |
| Mightex_LEDDriver_SDK.dll | --- DLL file exports functions. |
| Mightex_LEDDriver_SDK.lib | --- Import lib file, user may use it for VC++ development. |
| Hiddll.dll | --- DLL file used by "Mightex_LEDDriver_SDK.dll" . |

\Documents directory:

MighTex Sirius Multi-Channel LED Controller SDK Description.pdf

\Examples directory

\Delphi --- Delphi 5.0 project example.
\VC++ --- VC++ 6.0 project example.
\VB_Application – VB6.0 project example, note that for VB application, user **MUST** use the "Mightex_LEDDriver_SDK_Stdcall.dll", which is included in this directory.

*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 "Mightex_LEDDriver_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_PROFILE_ITEM     128   // For SX Modules, it's 3 instead of 128.

#defineDISABLE_MODE                    0
#define NORMAL_MODE                    1
#define STROBE_MODE                    2
#define TRIGGER_MODE                   3
```

```
#define MODULE_AA                    0
#define MODULE_AV                    1
#define MODULE_SA                    2
#define MODULE_SV                    3
#define MODULE_MA                    4
#define MODULE_CA                    5
#define MODULE_HA                    6
#define MODULE_HV                    7
#define MODULE_FA                    8
#define MODULE_FV                    9
#define MODULE_XA                   10
#define MODULE_XV                   11
#define MODULE_QA                   12

#pragma pack(1)
typedef struct {
  // Normal Mode Parameters
    int Normal_CurrentMax;
    int Normal_CurrentSe;

 // Strobe Mode Parameters
    int Strobe_CurrentMax;
    int Strobe_RepeatCnt;
    int Strobe_Profile[MAX_PROFILE_ITEM][2];

// Trigger Mode Parameters
    int Trigger_CurrentMax;
    int Trigger_Polarity;
    int Trigger_Profile[MAX_PROFILE_ITEM][2];
} TLedChannelData;
#pragma pack()

// Export functions:
SDK_API MTUSB_LEDDriverInitDevices( void );
SDK_API MTUSB_LEDDriverOpenDevice( int DeviceIndex );
SDK_API MTUSB_LEDDriverCloseDevice( int DevHandle );
SDK_API MTUSB_LEDDriverSerialNumber( int DevHandle, char *SerNumber, int Size );
SDK_API MTUSB_LEDDriverDeviceChannels( int DevHandle );
SDK_API MTUSB_LEDDriverSetMode( int DevHandle, int Channel, int Mode );
SDK_API MTUSB_LEDDriverSetNormalPara( int DevHandle, int Channel,TLedChannelData *LedChannelDataPtr) ;
SDK_API MTUSB_LEDDriverSetNormalCurrent( int DevHandle, int Channel, int Current );
SDK_API MTUSB_LEDDriverSetStrobePara( int DevHandle, int Channel, TLedChannelData *LedChannelDataPtr );
SDK_API MTUSB_LEDDriverSetTriggerPara( int DevHandle, int Channel,TLedChannelData *LedChannelDataPtr );
SDK_API MTUSB_LEDDriverResetDevice( int DevHandle );
SDK_API MTUSB_LEDDriverStorePara( int DevHandle );
SDK_API MTUSB_LEDDriverRestoreDefault( int DevHandle );
SDK_API MTUSB_LEDDriverGetLoadVoltage( int DevHandle, int Channel );
SDK_API MTUSB_LEDDriverGetCurrentPara( int DevHandle, int Channel,TLedChannelData
                                        *LedChannelDataPtr,int *Mode );
SDK_API MTUSB_LEDDriverSendCommand(int Dev Handle, char* Command);
```

Basically, only ONE data structure TledChannelData is needed for mode (NORMAL, STROBE and TRIGGER) parameter setting, note that “#pragma (1)” should be used (as above) for the definition of this structure.
**Note:** For AA/AV/SA/SV/HA/HV/MA/CA modules, the resolution of current is 1mA, thus in the API of setting current, 100 means 100mA, however, for FA/FV/XA/XV modules, the resolution of the current is 0.1mA, setting of 100 actually means 10.0mA.

**EXPORT Functions:**

**SDK_API MTUSB_LEDDriverInitDevices( void );**

Parameter: None
Return: it returns the number of SLC series LED drivers currently connected to PC, while there's no device connected, it's ZERO.
Note: User should always invoke this function before calling any other functions.

**SDK_API MTUSB_LEDDriverOpenDevice( int DeviceIndex );**

Parameter: DeviceIndex – the index from 0 – (TotalDevices–1), here TotalDevices is the value returned by the function of MTUSB_LEDDriverInitDevices( void ).
Return: it returns HANDLE of this device, the HANDLE can be used as first parameter of all other APIs to operate this device.
**-1 means Device open failed, this might be caused by device error.**
Note: User should always open the device before doing all device related operations, note that ZERO is also a valid value for handle.

**SDK_API MTUSB_LEDDriverCloseDevice( int DevHandle );**

Parameters: DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
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.

**SDK_API MTUSB_LEDDriverSerialNumber( int DevHandle, char *SerNumber, int Size );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
SerNumber – The pointer to char buffer which will filled with module serial number.
Size – The Size of the SerNumber buffer, the buffer should at least holds 16 characters.
Return: it returns –1 means this function call failed,
otherwise the call is successful.

**SDK_API MTUSB_LEDDriverDeviceChannels( int DevHandle );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Return: it returns –1 means this function call failed,
otherwise the return value is the channel number of this device.

**SDK_API MTUSB_LEDDriverDeviceModuleType( int DevHandle );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Return: it returns –1 means this function call failed,
otherwise the return value is the Module type which is defined in the header file:

| | |
|---|---|
| MODULE_AA | 0 |
| MODULE_AV | 1 |
| MODULE_SA | 2 |
| MODULE_SV | 3 |
| MODULE_MA | 4 |
| MODULE_CA | 5 |

…
please refer to the header file for all the modules supported by the SDK.

Note: User should always get the Module type, and invokes following APIs accordingly, for example, the "MA" Module doesn't support all Trigger Mode operations.

**SDK_API MTUSB_LEDDriverSetMode( int DevHandle, int Channel, int Mode );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Channel – The channel number of the channel user wants to operate, it's ONE based.
Mode – The setting mode:
0 --- DISABLE
1 --- NORMAL
2 --- STROBE
3 --- TRIGGER
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.
**Important**: For **SLC-MA04/CA04-MU** module, user should send "ECHOOFF" command with the API of "**MTUSB_LEDDriverSendCommand(int Dev Handle, char* Command)** " to put the module in to "PC Mode" before it sends any control command to LED Driver. Please refer the LED Driver User Manual for the SLC-MA04/CA04-MU control.

**SDK_API MTUSB_LEDDriverSetNormalPara( int DevHandle, int Channel,TLedChannelData *LedChannelDataPtr) ;**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Channel – The channel number of the channel user wants to operate, it's ONE based.
LedChannelDataPtr – The pointer to variable of TledChannelData structure, the setting parameters should be first filled in this variable.
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.

**SDK_API MTUSB_LEDDriverSetNormalCurrent( int DevHandle, int Channel, int Current );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
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.
Current – The setting 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.

**SDK_API MTUSB_LEDDriverSetStrobePara( int DevHandle, int Channel, TLedChannelData *LedChannelDataPtr );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Channel – The channel number of the channel user wants to operate, it's ONE based.
LedChannelDataPtr – The pointer to variable of TledChannelData structure, the setting parameters should be first filled in this variable.
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.

**SDK_API MTUSB_LEDDriverSetTriggerPara( int DevHandle, int Channel,TLedChannelData *LedChannelDataPtr );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().

Channel – The channel number of the channel user wants to operate, it's ONE based.
LedChannelDataPtr – The pointer to variable of TledChannelData structure, the setting parameters should be first filled in this variable.
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.

**SDK_API MTUSB_LEDDriverResetDevice( int DevHandle );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
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.

**SDK_API MTUSB_LEDDriverStorePara( int DevHandle );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
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.

**SDK_API MTUSB_LEDDriverRestoreDefault( int DevHandle );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
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.

**SDK_API MTUSB_LEDDriverGetLoadVoltage( int DevHandle, int Channel );**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Channel – The channel will be read
Return: it returns –1 means this function call failed,
otherwise the call is successful. The returned voltage is in "mv".

*SDK_API MTUSB_LEDDriverGetCurrentPara( int DevHandle, int Channel,TLedChannelData *LedChannelDataPtr,int *Mode );*

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Channel – The channel will be read
LedChannelDataPtr – Reference of variable of TledChannelData type, caller should assign such a variable, while this function will fill it with the current parameters of the driver.
Mode – Reference of variable of "integer" which contains the current mode of the 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

**SDK_API MTUSB_LEDDriverSendCommand(int Dev Handle, char* Command);**

Parameters:
DevHandle – The device handle returned by MTUSB_LEDDriverOpenDevice().
Command – The ASCII String, please refer to RS232 command set for the command strings.
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.

**Note:** Although the SDK is for USB module only, it's allowed to input ASCII string command via this function call, the command set is completely the same as the RS232 command set.
Example:
MTUSB_LEDDriverSendCommand( handle, "NORMAL 1 100 10" );
This function call will set the Normal Mode parameter of Channel 1 of the module to Imax = 100mA, Iset = 10mA.

# Command Sets:

## Important understanding:

It's important to understand that the Device has 4 working modes : **DISABLE**, **NORMAL**, **STROBE** and **TRIGGER**, each mode has a set of parameters (except the DISABLE mode), such as Imax, Iset…etc., user can set those parameters at any time, no matter the parameters of the mode is the current working mode or not. If the current working mode is ***NOT*** the mode of the parameters you're setting, You won't see the physical output change on channel output until the mode is set to be current working mode. The setting parameters are stored in device's internal memory, but not Non-Volatile memory yet, So after power cycle, the settings will be lost, device will reset all settings to factory default, or the last "STORE" parameters, If user wants to store all current settings (including all parameters of all modes of all channels) to Non-Volatile memory, user can use "STORE" command (refer to command set details) to do that.

For a factory delivery device, it's in DISABLE mode as default, and the parameters for each mode are as following:

| Mode | | Imax (mA) | Iset ( Profile) | |
|---|---|---|---|---|
| DISABLE* | | - | - | |
| NORMAL | | 20 | 10 | |
| STROBE | | 20 | 0/0 (No points) | |
| TRIGGER | | 20 | 0/0 (No points) | |

**** For complete details about the operation modes and the parameters, please refer to the user manual.***

**For device with USB interface, user can send the command (ASCII string) to device by invoking the API "MTUSB_LEDDriverSendCommand()", user should do InitDevice and OpenDevice properly prior to invoking this Send command API.**

**For device with RS232 interface, user can send the command (ASCII string) to the UART port**
**There's no need to invoke any SDK APIs. Actually the above APIs are for USB device only.**
**The hardware setting of the RS232 port is as following:**

***The device has 3 lines RS232 serial port as following:***
*It has a DB9 Female connector which has the following definition:*

*Pin2 ---- TXD*
*Pin3 ---- RXD*
*Pin5 ---- GND*

*With a Male To Female* ***straight line cable (Note, do NOT use null modem cable)****, user can connect it to a standard PC DB9 COM port.*

*Baud rate is set to 9600 bps, the parameter is 9600, N, 8, 1 (No hardware flow control)*

**For RS232 device, it's recommended for user to use the commands via Hyper-terminal to communicate/test with the device before using these command sets to develop user own application.**

## Command Set Format:

We use "Command Set" as our software interface to control the device, each command is actually an ASCII string which is ended with <LF> <CR> ( Hex Code: 0A, 0D ). This enables us to use standard PC Hyper Terminal to control the Device. Device can work as an echo server (controlled by Echo On/Off command) with "EchoOn" command, In this case, it will each back each ASCII character it receives.

### Echo Mode

**"EchoOn"** Mode is mainly for Debug/Service purpose, which allows user to use tools like Hyper terminal, In this mode, Device will echo back each character is received( except for "Back Space"), and It will send back an additional ">" as ***Prompt*** when it got <CR>.

**"EchoOff"** Mode is mainly for Host Software control purpose, it won't echo back any characters it receives. ***It's the Default mode after restart of the device*** (include Soft and Hard restart).

**Examples:**

**"EchoOn"** mode:

| | | |
|---|---|---|
| Host Send: | NORMAL 1 1000 500<LF><CR> | |
| Device Send back: | | |
| | NORMAL 1 1000 500<LF><CR> | // This is Echo back of receiving characters |
| | >##<LF><CR> | // Response |
| | > | // An Additional Prompt. |

**"EchoOff"** mode:

| | |
|---|---|
| Host Send: | NORMAL 1 1000 500<LF><CR> |
| Device Send back: | ##<LF><CR> |

### Command Structure

All Commands are in the following format:

Command DATA1 DATA2 … DATAn<LF><CR>

Note:

- Command is One or Multiple ASCII Characters (ASCII String).
- DATA1,… DATAn are data fields for command, they're ASCII characters and separated by SPACE. Note that some commands might not have DATA fields. The UNIT of the data is specified in each command's description.
- All the Commands must be ended by <LF> and <CR>
- Commands are **NOT** case sensitive.
- <LF> <CR> are 0x0A and 0x0D in hex, 10 and 13 in decimal.

### Response Structure

In both Echo modes, Device will always return a "Response" to the last input command, the response has the following format:

**Response<SP> <CR><LF>**

Here, Response can be one of the following:

## ---- [##] The latest command is valid and execution is OK.

#! ---- [#!] The command is valid and executed, but there's an error occurred during execution. Host can use "Error" command to get the Error code.

#? ---- [#?] The latest command is a valid command but the argument is NOT in valid range (e.g. Channel No. out of range)

#ASCII String ---- [#xxxx] The latest command is valid and execution is OK, Device returns a string as a result, please refer to the "Command details" for the contents of string for each command.

ASCII String ---For the command "**DEVICEINFO**", the device will return all the device information to host in a long ASCII string.

When user input an invalid command (command NOT in device's command set), the device will return [<SP><SP><SP><SP>xxxx is not defined], here, "xxxx" is the command user input.

## Command Set Details:

### ECHO MODE COMMAND:

***. Echo Control**

**Format: ECHOON<LF><CR>**

**Format: ECHOOFF<LF><CR>**

It sets the Device in "EchoOn" or "EchoOff" Mode, Please note that after a reset, the device is in "EchoOff" as default.

***. Important:** For SLC-MA04/CA04-MU module, when host sends these two commands to module, the module will enter "PC Mode" in which Host gets full control of the channel outputs.

### CURRENT WORKING MODE COMMAND:

***. Get Current Working Mode**

**Format: ?MODE CHLNo<LF><CR>**

CHLNo : Channel Number, start from 1, e.g. for a 4 channel device, it can be 1 – 4. ( All the CHLNo in other commands have the same definition)

Example: ***?MODE 1<LF><CR>***

**Return: #mode<CR><LF>**

Mode: 0 – **DISABLE**
1 – **NORMAL**
2 – **STROBE**
3 – **TRIGGER**

***. Set Current Working Mode**

**Format: MODE CHLno mode<LF><CR>**

CHLno: Channel Number.

Mode: it's the current mode for this channel:

0 – **DISABLE**
1 – **NORMAL**
2 – **STROBE**
3 - **TRIGGER**

This command sets the channel to the specified mode, for **STROBE** mode, this command is also used to start the programmed strobe profile, so when working in STROBE mode, host may use this command multiple times (re-enter the same mode) to re-start the profile.

### NORMAL MODE COMMAND:

***. Set Normal Mode Parameter**

**Format: NORMAL CHLno Imax Iset<LF><CR>**

CHLno: Channel Number

Imax: The maximum current allowed for **NORMAL** mode, in mA (or 0.1mA)

Iset: The working current for **NORMAL** mode, in mA (or 0.1mA).

Example: ***NORMAL 1 1000 500<LF><CR>***

***. Set Normal Mode Working Current**

**Format: CURRENT CHLno Iset<LF><CR>**

CHLno: Channel Number

Iset: The working current for **NORMAL** Mode, in mA (or 0.1mA).

Example: ***CURRENT 1 500<LF><CR>***

***. Get Normal Mode Parameters**

**Format: ?CURRENT CHLno<LF><CR>**

CHLno: Channel Number.

**Return: #Cal1 Cal2 Imax Iset<CR><LF>**

Example: ***?CURRENT 1<LF><CR>***

***#0 0 1000 500<CR><LF>***

This command returns the Imax and Iset of Normal mode of the channel, user can ignore the first 2 parameters, which are used for calibration only.

## STROBE MODE COMMAND:

***. Set Strobe Mode Parameters**

**Format: STROBE CHLno Imax Repeat<LF><CR>**

CHLno: Channel Number.

Imax: The maximum current for **STROBE** mode, in mA(or 0.1mA).

Repeat: Repeat Count for running the profile. It can be from 0 to 99999999. And the number 9999 is special, it means repeat forever. Note that when it's 0, the programmed wave form will output once, when it's 1, the wave form will be repeated once, which will be output twice and so on.

Example: ***STROBE 1 1000 5 <LF><CR>*** /*it will output 6x programmed wave forms */

Note: Each Channel has a programmable profile for STROBE mode, The profile contains 128 Set Points, and each set point has Iset/Tset pair, A ZERO/ZERO pair means it's the end Of the profile. If user doesn't program the Profile for a certain channel, the default is All Zero/Zero pairs, which means the Channel is always OFF. User should use the following "Set Strobe Profile" command to set a customized profile, and then enter **STROBE** mode. The profile will be executed (repeatedly) while device enter ( or reenter ) the **STROBE** mode with the "MODE" command. The unit for Iset is in "mA", for Tset is in "us".

***. Set Strobe Profile**

**Format: STRP CHLno STPno Iset Tset<LF><CR>**

CHLno: Channel Number.

STPno: As there might be 128 steps as maximum, user may input step count at STPno. That implies there might be 128 such commands to construct a 128 pair profile. Note that this STPno is started from 0, the valid range is 0 – 127.(As we also expect a (0,0) pair to mark the end of a profile, that means the actual usable pairs are 127)

Iset, Tset: This is Iset/Tset pair.

Example:

***STRP 1 0 500 2000<LF><CR>*** /* (500mA, 2000uS ) */

***STRP 1 1 10 100000<LF><CR>*** /* (10mA, 100000uS) */

***STRP 1 2 0 0 <LF><CR>*** /* (0,0) –End */

***. Get Strobe Mode Parameters**

**Format: ?STROBE CHLno<LF><CF>**

CHLno: Channel Number

**Return: #Imax Repeat<CR><LF>**

Example:

*?STROBE 1<LF><CR>*

*#1000 5<CR><LF>*

***. Get Strobe Mode Profile Parameters**

**Format: ?STRP CHLno<LF><CR>**

CHLno: Channel Number

**Return: #Iset1 Tset1 <CR><LF>**

**Iset2 Tset2<CR><LF>**

**……**

Example:

*?STRP 1<LF><CR>*

*#500 2000<CR><LF>*

*10 100000<CR><LF>*

*0 0<CR><LF>*

**TRIGGER MODE COMMAND:**

***. Set Trigger Mode Parameters**

**Format: TRIGGER CHLno Imax Polarity<LF><CR>**

CHLno: Channel Number

Imax: The maximum current allowed for **TRIGGER** mode, in mA(or 0.1mA).

Polarity: polarity of trigger:

0 - Rising edge of external trigger signal asserts,

1 - Falling edge of external trigger signal asserts.

Example: TRIGGER 1 1000 0<LF><CR>

Note: Each Channel has a programmable profile for TRIGGER mode, The profile contains 128 Set Points, and each set point has Iset/Tset pair, A ZERO/ZERO pair means it's the end Of the profile. If user doesn't program the Profile for a certain channel, the default is All Zero/Zero pair, which means the Channel is always OFF. User should use the following "Set Strobe Profile" command to set a customized profile, and then enter TRIGGER mode. The profile will be executed while an external trigger occurs and the device is in TRIGGER mode. The Unit for Iset is in "mA", for Tset is in "us".

***. Set Trigger Profile**

**Format: TRIGP CHLno STPno Iset Tset<LF><CR>**

CHLno: Channel Number.

STPno: As there might be 128 steps as maximum, user may input step count at STPno. That implies there might be 128 such commands to construct a 128 pair profile. Note that this STPno is started from 0, the valid range is 0 – 127.

Iset, Tset: Current/Time pair.

Example:

*TRIGP 1 0 500 2000<LF><CR>* /* (500mA, 2000uS ) */

*TRIGP 1 1 10 100000<LF><CR>* /* (10mA, 100000uS) */

*TRIGP 1 2 0 0 <LF><CR>* /* (0,0) –End */

***. Get Trigger Mode Parameters**

**Format: ?TRIGGER CHLno<LF><CR>**

CHLno: Channel Number

**Return: #Imax Polarity<LF><CR>**

Example:

*?TRIGGER 1<CR><LF>*
*#1000 0<CR><LF>*

***. Get Trigger Profile**
**Format: ?TRIGP CHLno<LF><CR>**
CHLno: Channel Number.
**Return: #Iset1 Tset1 <CR><LF>**
**Iset2 Tset2<CR><LF>**
**……**

Example: *?TRIGP 1<LF><CR>*
*#500 2000<CR><LF>*
*10 100000<CR><LF>*
*0 0<CR><LF>*

**OTHER COMMAND:**

***. Get Channel Load Voltage**
**Format: LoadVoltage CHLno<LF><CR>**
CHLno: Channel Number ( 1 – 4 ).
For XV Module (e.g. AV04 or SV04), Host uses this command to get the current voltage on the load of the specified channel, it will return:
#CHLno:vvvvv<CR><LF>
Here, vvvvv is the voltage in "mv".
**Note:** As the controller polls the load voltage in a 20ms interval, this feature is proper for NORMAL mode or slow Strobe mode only.

***. Reset Device**
**Format: Reset<LF><CR>**
Host uses this command to Soft Reset the device.

***. Restore Factory Default**
**Format: RESTOREDEF<LF><CR>**
That command will reset the device's mode and all related parameters to its factory default (Refer to table for the default values in First page), note that these parameters becomes current settings, to make it to become "STORE" setting, user must use "STORE" command to save the current setting to NV memory.

***. Store All settings to NV memory**
**Format: STORE<LF><CR>**
This command will store the current settings in volatile memory to Non-Volatile memory.

***. Device Information**
**Format: DEVICEINFO<LF><CR>**
Example: *DEVICEINFO<LF><CR>*
Device will return the Device Type, Firmware version, Serial Number…etc. in ONE line (only one <CR><LF>).

For **SLC-MA04/CA04-MU** Module, there's one more command:

***. Set Fan PWM Ratio**
**Format: FanPWM PWMLevel<LF><CR>**
Example: *FanPWM 5<LF><CR>*
PWMLevel is a number from 0 – 10 which means ratio of 0%(Full Off) to 100%(Full On), In the example above, it sets the PWM ratio to 50%.

## Using DLL APIs in  NI LabVIEW

As the USB LED 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.

Except for the **MTUSB_LEDDriverInitDevices( )**,  **MTUSB_LEDDriverOpenDevice( int DeviceIndex )** and **MTUSB_LEDDriverCloseDevice( int DevHandle )** functions, it's recommended to use the **MTUSB_LEDDriverSendCommand(int Dev Handle, char* Command)** function to control the LED Driver, Note that user should construct the command string ( ASCII string ) and then invoke this function. For the command string, please refer to the above command set description.

The "LEDDriver_LVExample.vi" included in the CDROM show the way to use the CLF nodes to construct a LabView applications, user might use it as a start point for his own applications.