Mightex
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

Mightex Spectrometer Software Engine SDK Manual

Version 1.0.3

Nov. 23, 2009

Relevant Products

Part Numbers
SSE-1304-U

Revision History

Revision	Date	Author	Description
1.0.0	Sept. 28, 2009	JT Zheng	Initial Revision
1.0.1	Oct. 10, 2009	JT Zheng	RRC/ETC enable
1.0.2	Oct. 20, 2009	JT Zheng	GPIO APIs added
1.0.3	Nov. 23, 2009	JT Zheng	Correct GPIO API Name typo

Mightex USB 2.0 CCD spectrometer is designed for low cost spectrum applications, With USB 2.0 high speed interface and powerful PC software engine, the device delivers spectrum at high frame rate. SSE Application software is provided for user's quick operations, In addition, SDK is also provided for user's developments.

IMPORTANT:

Mightex USB spectrometer is using USB 2.0 for data collection, USB 2.0 hardware MUST be present on user's PC and Mightex device driver MUST be installed properly before developing application with SDK. **For installation of Mightex device driver, please refer to *Mightex Spectrometer User Manual*.**

SDK FILES:

The SDK includes the following files:

\Library directory:

MT_Spectrometer_SDK.h	--- Header file which contains "C" prototypes of the APIs
MT_Spectrometer_SDK.lib	--- lib file for APIs.
MT_Spectrometer_SDK.dll	--- DLL file exports APIs.
CCD_USBCamera_SDK.dll	--- DLL file used by "MT_Spectrometer_SDK.dll" .
LE_txtImporter.dll	--- DLL file used by "MT_Spectrometer_SDK.dll" .
LE_XlsImporter.dll	--- DLL file used by "MT_Spectrometer_SDK.dll" .
LE_Colorimetry.dll	--- DLL file used by "MT_Spectrometer_SDK.dll" .
LinearCameraUsblib.dll	--- DLL file used by "MT_Spectrometer_SDK.dll" .

\Documents directory:

MighTex SSE SDK Manual.pdf

\Examples directory

\Delphi --- Delphi project example.
\VC++ --- VC++ (VC++ 6.0) example code
\LabView --- LV example code

Important: When a spectrometer is shipped from factory, The CD ROM contains "\Application" directory which has the following contents:

\Application

It has EXE file named "Mightex_SSE_App.exe" and related DLL files, and two pre-defined sub-directories as following:

\Appdata : it has five pre-defined files and one pre-defined sub-directory.

Para.ini : Main Parameter file for SSE.
Pixelmode.ini : Parameter file for SSE
WavelengthSets.ini : Pre-defined wavelength sets.
CalibrationFile1.cal : Example of calibration file.
ImportFile1.mtp : Example of import file.
\ModuleNo_SerialNo : This is a sub-directory which includes RRC and ETC calibration files, those files are generated in factory when the device is calibrated.

\Data : This is an empty sub-dir, user might put spectrum data files under it, for example, user can use it as "Time line" save/load path.

When user develops his own application with spectrometer engine (the DLLs), user should copy all the DLL files (in Library directory) into user's own application's directory, and user should also copy whole "\appdata" directory (and all the files, sub-directory under it) into this directory (the same directory as the DLLs), thus the DLLs can find these files to get all necessary information (e.g. calibration data).
Please refer to the example codes (Delphi, VC++ or LabView) for user applications.

Note

1). The spectrometer engine supports Multiple devices, while allows operation on ONLY ONE device at a time. User may invoke functions to get the number of SSE devices currently present on USB and each device's module no. and serial no., and select one of the device as "Working Device" which is activated for user to acquire spectrum from it. While user wants to set another device as "Working Device", user should stop the devices.

The following procedure demonstrates this process:

MTSSE_StartDevices;
MTSSE_GetModuleNo;
MTSSE_GetSerialNo;
MTSSE_SetWorkingDevice; // User select one device as current "Working Device"

…
(Operations on the active spectrometer, such as set work mode, set exposure time, grabbing spectrum, etc.)
…

MTSSE_StopDevices;

2)Although spectrometer are USB Devices, spectrometer engine is **NOT** supporting Plug&Play of the spectrometers, it's NOT recommended to Plug or Unplug spectrometer while the camera engine is grabbing frames from the cameras.

3). The code examples are for demonstration of the DLL functions only, device fault conditions are not fully handled in these examples, user should handle those error conditions properly.

EXPORT Functions:

MT_Spectrometer_SDK.dll exports functions to allow user to easily and completely control the spectrometer and get image frame.
In the Header file, we have the following Macro definitions:

typedef int SDK_RETURN_CODE;
#define SDK_API extern "C" __declspec(dllimport) SDK_RETURN_CODE _cdecl
*#define SDK_POINTER_API extern "C" __declspec(dllimport) unsigned short * _cdecl*

SDK_API MTSSE_StartDevices(HWND PAHandle)

This is first function user should call for his own application, this function communicates with the installed device driver and reserve resources for further operations.
Arguments: PAHandle – the handle of the parent window who invokes the engine, when there's no parent window, it can be set to NULL.
Return: The number of SSE spectrometer currently attached to the USB 2.0 Bus, indexing from 1. If there's no Mightex USB spectrometer attached, the return value is 0.
Note: There's NO device handle needed for calling further spectrometer related functions, after invoking MTSSE_StartDevices, camera engine reserves resources for the attached devices. For example, if the returned value is 2, which means there TWO devices currently presented on USB, user may use "1" or "2" as DeviceID to call further device related functions, "1" means the first device and "2" is the second device.

SDK_API MTSSE_StopDevices(void)

This is the function to release all the resources reserved by **MTSSE_StartDevices,** user should invoke it before application terminates.
Arguments: None
Return: Always return 0.

SDK_API MTSSE_GetModuleNo(const int iDevice, char *ModuleNo)
SDK_API MTSSE_GetSerialNo(const int iDevice, char *SerialNo)
For a present device, user might get its Module Number and Serial Number by invoking these functions.
Argument: iDevice – the index of the device, Please refer to the notes of **MTSSE_StartDevices()** function.
ModuleNo – the pointer to a character buffer, the buffer should be available for at least 16 characters.
SerialNo – the pointer to a character buffer, the buffer should be available for at least 16 characters.
Return: -1 If the function fails (e.g. invalid device number)
0 if the call succeeds.

SDK_API MTSSE_SetWorkingDevice(int iDevice)

For currently connected spectrometers, only one spectrometer can work at a time. User can use the 'iDevice' to set current working spectrometer.
Arguments: iDeive-the index of the spectrometer.
Return:-1 If the function fails(e.g. Invalid device number).
0 if the function succeeds.

SDK_API MTSSE_SetDeviceWorkMode(int WorkMode)

This function set work mode to the current working spectrometer.
By default, the spectrometer is working in "**NORMAL"** mode in which spectrometer can deliver spectrum to PC by software commands, however, user may set it to **"TRIGGER"** Mode, in which the spectrometer is waiting for an external trigger signal and capture ONE spectrum for each trigger signal assertion.
Argument: WorkMode – 0: **NORMAL** Mode, 1: **TRIGGER** Mode.
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succeeds.

SDK_API MTSSE_SetExposureTime(int iMicronSec)

User may set the exposure time of the current working spectrometer by invoking this function.
Argument:exposureTime – the Exposure Time to be set, note it's in "Microsecond", and as the Spectrometer's minimum resolution for exposure is 100us, so it should be multiple of 100us (including 100us).
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succeeds.

SDK_API MTSSE_SetFrameAvgCnt(int AvgCnt)

User may use this function to set average frame number of current working spectrometer.
Argument: AvgCnt – the average frame count.
Return:-1 If the function fails(e.g AvgCnt < 0).
1 if the call suceeds.

SDK_API MTSSE_SetAutoDarkStatus(int bAutoDarkStatus)

User may use this function to set status of "Dark compensation" of the current spectrometer, please refer to SSE User manual for the detailed description of "Dark Compensation" function.
Argument: bAutoDarkStatus – the auto dark compensation status.
1: set the auto dark compensation to true.
0: set the auto dark compensation to false.
Return: Always return 1.

SDK_API function MTSSE_SetRRCStatus(int bRRCStatus)

If RRC calibration file for the target spectrometer provided by Mightex exists, User may use this function to set status of "RRC compensation" of the current spectrometer, please refer to SSE User manual for the detailed description of "RRC Compensation" function.
Arguments: bRRCStatus – the status of RRC to be set.
1: sets the RRC Compensation to true.
0: sets the RRC Compensation to false.
Return: 1, If RRC.cal file exists and has been located.
-1, If RRC.cal file does not exist or has not been located.
Important Notes: RRC Correction will require the RRC calibration file provided by Mightex.

SDK_API MTSSE_SetLETCStatus(int bETCStatus)

If ETC calibration file for the target spectrometer provided by Mightex exists, User may use this function to set status of "ETC compensation" of the current spectrometer, please refer to SSE User manual for the detailed description of "ETC Compensation" function.
Arguments: bETCStatus – the status of ETC to be set.
1: sets the ETC Compensation to true.
0: sets the ETC Compensation to false.
Return: 1, If ETC.cal file exists and has been located.
-1, If ETC.cal file does not exist or has not been located.
Important Notes: ETC Correction will require the ETC calibration file provided by Mightex.

SDK_API MTSSE_StartGrabSpectrum(void)

For getting a spectrum from the device, user should invoke this function first, followed by the following **MTSSE_GetSpectrum()**, When this function is invoked, the spectrometer engine will start to grab spectrums (set by the AvgCnt set before).
Arguments: No.
Return: Always return 1.

Important: As it might take considerable time to get a spectrum (it depends on the exposure time and avgcnt set by user), this function will return immediately before the spectrum is grabbed, user might do something else and then invoke the following **MTSSE_GetSpectrum()**, to check to see whether the spectrum is ready.

SDK_POINTER_API MTSSE_GetSpectrum(unsigned short *SpectrumDataPtr, int WaitUntilDone)

After sending the above **MTSSE_StartGrabSpectrum**() command, user can invoke this function to check whether the spectrum is ready and to get the spectrum.
Arguments: SpectrumDataPtr - the pointer to the spectrum data array if spectrum is ready or nil if spectrum is not ready.

WaitUntilDone – the frame grabbing process may need considerable time (depends on the exposure time setting and average frame number setting). User might set:

1: The API will be blocked until the spectrum is grabbed, the returned spectrum is pointed by the SpectrumDataPtr (and the return value), If the returned pointer is nil, "Time out" error occurs.

0: The API will not be blocked, if the spectrum is ready, the spectrum will be returned in SpectrumDataPtr (and the function's return value), otherwise, it will return nil.

Return: nil, if the grabbing command is not finished (when WaitUntilDone is "0") or "Time out" error occurs (when WaitUntilDone is "1").

The pointer to the spectrum data array. (The same as the SpectrumDataPtr)

SDK_API MTSSE_CalcPixelValue(int Wavelength)
SDK_API MTSSE_CalcWavelengthValue(int Pixel)

By using these two functions, user can easily convert wavelength to pixel value or inverse.
Arguments:
Pixel – Pixel value returned or to be converted, it can be from 1 to 3648, as the CCD sensor has 3648 pixels.
Wavelength: wavelength value returned or to be converted, note that is in nanometer * 100 format, e.g. for wavelength of 610.23nm, the wavelength value is 61023.
Return: -1 if function fails.(e.g. Conversion error or operator error)
1 if call succeeds.

SDK_API MTSSE_SetGPIOConfig(unsigned char ConfigByte);

User may call this function to configure GPIO pins.
Argument : ConfigByte – Only the 4 LSB are used, bit0 is for GPIO1, bit1 is for GPIO2 and so on. Set a certain bit to 0 configure the corresponding GPIO to output, otherwise it's input.
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succedds.

SDK_API MTSSE_SetGPIOInOut(unsigned char OutputByte, unsigned char *InputBytePtr);

User may call this function to set GPIO output pin states and read the input pins states.
Argument : OutputByte – Only the 4 LSB are used, bit0 is for GPIO1, bit1 is for GPIO2 and so on. Set a certain bit to 1 will output High on the corresponding GPIO pin, otherwise it outputs Low. Note that it's only working for those pins are configured as "Output".
InputBytePtr – the Address of a byte, which will contain the current Pin States, only the 4 LSB bits are used, note that even a certain pin is configured as "output", we can still get its current state.
Return: -1 If the function fails (e.g. invalid device number)handle or it's camera WITHOUT GPIO)
1 if the call succedds.

Note: See "Mightex Spectrometer User Manual" for GPIO connector definition.