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 2.4.0

Dec, 2022

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
2.0.0	Jan.11.2011	S.S. D	Add function to get frame data and its property.
2.0.1	Oct.11.2013	S.S.D	Add callback function to get frame data
2.0.2	Nov.22 2013	S.S.D	Add two manual get functions
2.2.2	Oc.2018	S.S.D	Update to support Bioscience functions
2.3.1	Sep. 2020	S.S.D	Update black field data handling algorithm.
2.4.0	Dec.2022	William	Modify black field data handling algorithm. Update calib para file format. Improve ETC algorithm.

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 has two sub directories x86 and x64, each directory contains the following files:

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_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 Guide.pdf

\Examples directory

\Delphi --- This directory contains x86 and x64 Delphi project example.
\VC++ --- This directory contains x86 and x64 VC++ example code
\Labview --- Labview 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 calibration file 'ModuleNo_SerialNo'.cal, which is 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 devices 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_InitDevice;
MTSSE_GetDeviceModuleNoSerialNo;
MTSSE_SetDeviceActiveStatus;  // 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_UnInitDevice;
```

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.

HEADER FILES:
The content of “MT_Spectrometer_SDK.h” is as following:

```
typedef int SDK_RETURN_CODE;
typedef unsigned int DEV_HANDLE;

#ifdef SDK_EXPORTS
#define SDK_API extern "C" __declspec(dllexport) SDK_RETURN_CODE _cdecl
#define SDK_HANDLE_API extern "C" __declspec(dllexport) DEV_HANDLE _cdecl
#define SDK_POINTER_API extern "C" __declspec(dllexport) unsigned short * _cdecl
#else
#define SDK_API extern "C" __declspec(dllimport) SDK_RETURN_CODE _cdecl
#define SDK_HANDLE_API extern "C" __declspec(dllimport) DEV_HANDLE _cdecl
#define SDK_POINTER_API extern "C" __declspec(dllimport) unsigned short * _cdecl
#endif

typedef struct
{
  double* RawData;
  double* CalibData;
  double* AbsInten;
}tFrameRecord;

typedef struct
{
  int DeviceID;
  int ExposureTime;
  int TimeStamp;
  int TriggerOccurred;
  int TriggerEventCount;
  int OverSaturated;
  int LightShieldValue;
} TFrameDataProperty;

typedef void (* DeviceFrameDataCallBack)(int Row, int Col,
                                    TFrameDataProperty* Attributes, void **FramePtr );

SDK_API MTSSE_InitDevice(HWND ParentHandle );
SDK_API MTSSE_UnInitDevice( void );
SDK_API MTSSE_GetDeviceModuleNoSerialNo( int DeviceID, char* ModuleNo, char* SerialNo );
SDK_API MTSSE_GetDeviceSpectrometerWavCalPara( int DeviceID, int SpectrometerID, double*
&WavCalibValue);
SDK_API MTSSE_SetDeviceActiveStatus( int DeviceID, int ActiveFlag );
SDK_API MTSSE_InstallDeviceFrameHooker(int DeviceID, DeviceFrameDataCallBack DeviceHookerProc);
SDK_API MTSSE_SetDeviceAverageFrameNum( int DeviceID, int AverageFrameCount );
SDK_API MTSSE_SetDeviceWorkMode( int DeviceID, int WorkMode );
SDK_API MTSSE_SetDeviceExposureTime( int DeviceID, int ExposureTime );
SDK_API MTSSE_StartFrameGrab( int GrabType );
SDK_API MTSSE_StopFrameGrab( void );
SDK_API MTSSE_SetDeviceSpectrometerAutoDarkStatus( int DeviceID, int SpectrometerID, int AutoDrkFlag );
SDK_API MTSSE_SetDeviceSpectrometerETCStatus( int DeviceID, int SpectrometerID, int ETCFlag );
SDK_API MTSSE_SetDeviceSpectrometerDarkData( int DeviceID, int SpectrometerID, double *DarkData );
SDK_API MTSSE_GetDeviceSpectrometerFrameDataEx (int DeviceID, int SpectrometerID, int WaitUntilDone, int
DataIndex, double *Data);
SDK_API MTSSE_GetDeviceSpectrometerFrameData(int DeviceID, int SpectrometerID, int WaitUntilDone,
tFrameRecord* &FrameData );
SDK_API MTSSE_GetDeviceSpectrometerFrameDataProperty( int DeviceID,int SpectrometerID,int
WaitUntilDone,TFrameDataProperty* FrameProperty,tFrameRecord* &FrameData);
SDK_API MTSSE_SaveDeviceSpectrometerWavCalPara( int DeviceID, int SpectrometerID, double*
WavCalibArray);//
SDK_API MTSSE_GetDeviceSpectrometerCIE1931Coords( int DeviceID, int SpectrometerID, double* FrameData,
double &x, double &y );
```

```
SDK_API MTSSE_GetDeviceSpectrometerCIE1976Coords( int DeviceID, int SpectrometerID, double* FrameData,
double &u, double &v );
SDK_API MTSSE_GetDeviceSpectrometerCCT( int DeviceID, int SpectrometerID, double* FrameData, int &CCT);
SDK_API MTSSE_GetDeviceSpectrometerCRIs( int DeviceID, int SpectrometerID, double* FrameData, double
*CRIs);
SDK_API MTSSE_SetDeviceGPIOConfig( int DeviceID, int Config );
SDK_API MTSSE_SetDeviceGPIOInOut( int DeviceID, int Output, unsigned char *Input );
```

Note: Please check the header file itself of latest information, we may add functions from time to time.

EXPORT Functions:

SDK_API MTSSE_InitDevice(HWND ParentHandle);

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 device currently attached to the USB 2.0 Bus, indexing from 1. If there's no Mightex USB spectrometer device attached, the return value is 0.
Note: There's NO device handle needed for calling further spectrometer related functions, after invoking MTSSE_ InitDevice, camera engine reserves resources for the attached devices. For example, if the returned value is 2, which means there are TWO devices currently present 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, etc.

SDK_API MTSSE_UnInitDevice(void);
This is the function to release all the resources reserved by **MTSSE_ InitDevice,** user should invoke it before application terminates or before recalling the **MTSSE_InitDevice.**
Arguments: None
Return: Always return 1.

SDK_API MTSSE_InstallDeviceFrameHooker(int DeviceID, DeviceFrameDataCallBack DeviceHookerProc);
Argument: DeviceID – the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function
DeviceHookerProc – Callback function installed.
Return: -1 if the function failes.
1 if the function succeeds.
Note: 1) The callback function will only be invoked while the frame grabbing is started, host will be notified every time the spectrometer get a new frame.
2) The callback has the following prototype:

```
typedef void (* DeviceFrameDataCallBack)(int Row, int Col,  TFrameDataProperty* Attributes, void **FramePtr );
```

The TFrameDataProperty is defined as

```
typedef struct
{
        int DeviceID;
        int ExposureTime;
        int TimeStamp;
        int TriggerOccurred;
        int TriggerEventCount;
        int OverSaturated;
        int LightShieldValue;
} TFrameDataProperty;
```

Here, **DeviceID** – This is the spectrometer's index (1 based), which identifies the spectrometer who generates the frame.
ExposureTime – This is the exposure time of current frame, in second based.
TimeStampe – Spectrometer firmware will mark each frame with a time stamp, a number ranging from 0 - 65335(and it's automatically rounded back) which is generated by the internal timer, the unit of it is of 100us.
TriggerOccurred – While the spectrometer is in **NORMAL** mode, the spectrometer firmware will keep record if an external trigger signal is asserted. If an external signal is captured, the spectrometer firmware will set this flag to one, otherwise it is zero.
TrigerEventCount – While the spectrometer is in **TRIGGER** mode, the spectrometer will only grab frame based on the external trigger signal. Each trigger will increase this count by ONE. Note that this count is reset to ZERO whenever host set the spectrometer work mode to **TRIGGER** mode.
OverSaturated – A flag to show if there is data saturation happened. The CCD sensor only works properly under conditional lighting, if it's over exposured, the CCD will be over saturated and in this case, the frame data grabbing back doesn't make sense, as the electronics accumulated for a frame is NOT released completely and accumulatively affects the next frame. While this flag is set to 1, host should reduce the exposure time (or setting proper external lighting condition) to make it back to 0.
LightShiedValue - The CCD provides 13 pixels with Light shielded, this field provides an average

value of these pixels, user might use this value for black compensation.

Note: 1) The impact of **AverageFrameCount** on frame data property.
When the frame data is an averaging result(**AverageFrameCount** > 1),
TriggerOccured, any external trigger is asserted during any frame will set this flag to ONE.
TriggerEventCount is the total count of trigger event from all frames.
OverSaturated, if any data of any frame is saturated will set this flag to ONE.
LightShieldValue, is the averaging result of LightShieldValue of all frames on **AverageFrameCount.**

The Arguments of call back functions:

Row,Col – the row and column size of a frame, for linear spectrometer, the **ROW** = 1, and **Col** is the linear CCD pixels count, please refer to the Spectrometer's specification.

FramePtr - the pointer to address of frame structure, which is defined as

```
typedef struct
{
        double* RawData;
        double* CalibData;
        double* AbsInten;
}tFrameRecord;
```

Note: Each frame is comprised of three data structure, rawData, CalibData and AbsInten Data. The length of each data structure is the frame column size, please refer to the Spectrometer's specification for linear CCD's line pixel count.

SDK_API MTSSE_StartFrameGrab(int GrabType)
For getting a spectrum from the device, user should invoke this function first, followed by the following **MTSSE_GetDeviceSpectrometerFrameData ()**. When this function is invoked, the spectrometer engine will start to grab **AvgCnt** spectrums (set by the **AvgCnt**, see MTSSE_SetDeviceAverageFrameNum).
Arguments:
GrabType -- 1, get one frame.
-- 0x8888, get frames continuously until **MTSSE_StopFrameGrab()** is called.
Return: Always return 1.
Important: As it might take considerable time to get a spectrum (which depends on the exposure time **ExpTime** 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_ GetDeviceSpectrometerFrameData ()**, to check to see whether the spectrum is ready.

SDK_API MTSSE_StopFrameGrab(void);
This function is called to stop grabbing frame when **GrabType** in **SDK_API MTSSE_StartFrameGrab** is 0x8888.
Arguments: None.
Return: Always return 1

SDK_API MTSSE_GetDeviceModuleNoSerialNo(int DeviceID, char *ModuleNo, char *SerialNo)
For a present device, user might get its Module No. and Serial No. by invoking this function.
Argument: DeviceID – the index of the device, Please refer to the notes of **MTSSE_InitDevice()** 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 DeviceID)
1, if the call succeeds.

SDK_API MTSSE_SetDeviceActiveStatus(int DeviceID, int ActiveFlag) ;
Currently connected devices can be turn on/off by this function. The default state of device is off; User should invoke this function to set the device to active state for further operation.
Arguments: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
ActiveFlag -- 1: Set device to active status.
0: Set Device to inactive status.
Return: -1, if the function fails (e.g. Invalid device number).
1, if the function succeeds.

SDK_API MTSSE_SetDeviceWorkMode(int DeviceID, int WorkMode);
This function set work mode to the device specified by DeviceID.

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

SDK_API MTSSE_SetDeviceExposureTime(int DeviceID, int ExposureTime);

User may set the exposure time of the device specified by DeviceID.
Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
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 DeviceID)
1 if the call succeeds.

SDK_API MTSSE_SetDeviceAverageFrameNum(int DeviceID, int AverageFrameCount) ;
User may use this function to set average frame number of device specified by DeviceID.
Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function. AverageFrameCount– the average frame count- **AvgCnt**.
Return: -1 if the function fails (e.g. invalid DeviceID or AverageFrameCount < 0).
1 if the call succeeds.

SDK_API MTSSE_SetDeviceSpectrometerAutoDarkStatus(int DeviceID, int SpectrometerID, int AutoDrkFlag) ;
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: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer**, should always be set to 1.**
AutoDrkFlag– the dark compensation status.
1: set the auto dark compensation to true.
0: set the auto dark compensation to false.
Return: -1 if function fails.
1 if function succeeds.
Note: The **SpectrometerID** should always be set to 1.

SDK_API MTSSE_SetDeviceSpectrometerETCStatus(int DeviceID, int SpectrometerID, int ETCFlag) ;
If ETC calibration file for the target spectrometer 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: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer**, should always be set to 1.**
ETCFlag – the status of ETC to be set.
1: sets the ETC Compensation to true.
0: sets the ETC Compensation to false.
Return: -1, if function fails, e.g., ETC calibration file does not exist or has not been located.
1, if function succeeds.

SDK_API MTSSE_GetDeviceSpectrometerFrameData(int DeviceID, int SpectrometerID, int WaitUntilDone, tFrameRecord* &FrameData);
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: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer, **should always be set to 1.**

FrameRecordData - the pointer to the spectrum data structure if spectrum is ready or nil if spectrum is not ready. It is defined as:

```
typedef struct
{
  double* CCDData;
  double* CalibData;
  double* AbsIntenData
}tFrameRecord;
```

And each item again should points to a double data array which defined as double data[3648].
CCDData is the raw pixel data,
CalibData is the CCDDataa with ETC calibration (if etc flag was turned on), if ETC is off, it should equal the CCDData.
AbsIntenData returns the absolute intensity data if AIC coefficients exists

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 FrameRecordData, 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 FrameRecordData), otherwise, it will return nil.

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

1, if function succeeds.

SDK_API MTSSE_GetDeviceSpectrometerFrameDataEx (int DeviceID, int SpectrometerID, int WaitUntilDone, int DataIndex, double *Data);

This function behaves the same as **MTSSE_GetDeviceSpectrometerFrameData** , except for that this function can be used to get any one of the three items in the FrameData by specifying **DataIndex** . The definition of arguments other than DataIndex and Data are the same as the ones of **MTSSE_GetDeviceSpectrometerFrameData**, the other two are listed as follows,

Arguments: DataIndex – can be of value 0, 1 or 2, used to specify the data item to get from last grabbing action.
Set DataIndex to 0 to get *CCDData.*
Set DataIndex to 1 to get *CalibData,*
Set DataIndex to 2 to get *AbsIntenData.*
Data – the pointer to the data array specified by DataIndex.

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

1, if function succeeds.

SDK_API MTSSE_GetDeviceSpectrometerFrameDataProperty(int DeviceID,int SpectrometerID,int WaitUntilDone,TFrameDataProperty* FrameProperty,tFrameRecord* &FrameData);

This function behaves the same as **MTSSE_GetDeviceSpectrometerFrameData,** except for that this function can also get the frame data property when getting the frame data. The arguments other than FrameProperty can be found from previous function.

Arguments: FrameProperty is defined as

```
typedef struct
{
     int CameraID;
     int ExposureTime;
     int TimeStamp;
     int TriggerOccurred;
     int TriggerEventCount;
     int OverSaturated;
     int LightShieldValue;
} TFrameDataProperty;
```

See callback function for the frame data property.

SDK_API MTSSE_SetDeviceSpectrometerDarkData(int DeviceID, int SpectrometerID, double *DarkData);

When device is set to get the frame data, it will also calculate the absolute intensity data if wavelength calibration coefficients were found and AIC coefficients were found. User might want to subtract the dark field light from the current calculation. The function is used to set the dark data to be subtracted for absolute intensity calculation.

Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer, **should always be set to 1.**

DarkData – The dark field data to be sent to the device. It should contain MAX_CCD Points data. User can use the data from **GetDeviceSpectrometerFrameData.**
Return: - 1 If the function fails.
1 If the function success.

SDK_API MTSSE_GetDeviceSpectrometerWavCalPara(int DeviceID, int SpectrometerID,
double* &WavCalibData);
User can use this function to view the saved wavelength calibration coefficients.
Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer, **should always be set to 1.**
WavCalibData – which points to the spectrometer's wavelength calibration coefficients data structure, which is defined as

struct
{
double A[4];
double B[4];
};
Where A stores the pixel to wavelength coefficients and B stores the wavelength to pixel coefficients.

Return: -1 If the function fails
1 If the function success.

SDK_API MTSSE_SaveDeviceSpectrometerWavCalPara(int DeviceID, int SpectrometerID,
double * WavCalibArray);
User can use this function to set the device wavelength calibration coefficients.
Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
SpectrometerID – the index of the spectrometer, **should always be set to 1.**
WavCalibArray – the address of the valid wavelength calibration coefficients, which should be defined in the above format.
Return: -1 if the function fails.
1 if the function success.
Note: When user use this function to set the new wavelength calibration coefficients, it is recommended to restart the SSE engine.

SDK_API MTSSE_SetDeviceGPIOConfig(int DeviceID, int Config);
User may call this function to configure GPIO pins.
Argument : DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
Config– 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_SetDeviceGPIOInOut(int DeviceID, int Output; unsigned char *Input);
User may call this function to set GPIO output pin states and read the input pins states.
Argument : DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.
Output– 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".
Input– 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.

SDK_API MTSSE_GetDeviceSpectrometerCIE1931Coords(int DeviceID, int SpectrometerID,
double * FrameData, double &x, double &y);
SDK_API MTSSE_GetDeviceSpectrometerCIE1976Coords(int DeviceID, int SpectrometerID,
double * FrameData, double &up, double &vp);
User may call these two functions to get the CIE colorimetry coordinates .
Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.

SpectrometerID – the index of the spectrometer, **should always be set to 1.**

FrameData – the address of the frame data to be sent to device for calculating the CIE colorimetry coordinates. It should contain exact data points of the CCD points.

x ---- CIE 1931 colorimetry x coordinate;

y ---- CIE 1931 colorimetry y coordinate;

up ----CIE 1976 colorimetry u' coordinate;

vp ----CIE 1976 colorimetry v' coordinate.

Return: -1 if function fails.

1 if function succeeds.

Note: x,y,up and vp values are rounded to 4 digits after the decimal point.

SDK_API MTSSE_GetDeviceSpectrometerCCT(int DeviceID, int SpectrometerID, double * FrameData, int &CCT);

User may call this function to get the CIE CCT.

Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.

SpectrometerID – the index of the spectrometer, **should always be set to 1.**

FrameData – the address of the frame data to be sent to device for calculating the CIE CCT. It should contain exact data points of the CCD points.

CCT – the returned CCT value.

Return: -1 when CCT calculation beyond the allowable area or

1 if function succeeds.

SDK_API MTSSE_GetDeviceSpectrometerCRIs(int DeviceID, int SpectrometerID, double * FrameData, double *CRIs);

User may use this function to get the CIE color rendering Indexes (CRI).

Argument: DeviceID -- the index of the device, Please refer to the notes of **MTSSE_InitDevice()** function.

SpectrometerID – the index of the spectrometer, **should always be set to 1.**

FrameData – the address of the frame data to be sent to device for calculating the CIE CCT. It should contain exact data points of the CCD points.

CRIs – the address of the CIE color rendering indexes structure, which should be defined as

```
struct
 {
   double CRIa;
   double CRIs[14];
 };
```

Return: -1 If function fails.

1 If function success.