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 CCD Line Camera SDK Manual

Version 1.1.1

May. 16, 2008

Relevant Products

Part Numbers
TCN-1304-U, TCN-1209-U

Revision History

Revision	Date	Author	Description
1.0.0	Mar. 13, 2007	JT Zheng	Initial Revision
1.1.0	Jan. 18, 2008	JT Zheng	Add TCN-1209-U Modal
1.1.1	May. 16, 2008	JT Zheng	Add CCDUSB_GetCurrentFrame()

Mightex USB 2.0 CCD Line camera is designed for low cost spectrometer and machine vision applications, With USB 2.0 high speed interface and powerful PC camera engine, the camera delivers CCD Line image data at high frame rate. GUI demostration application and SDK are provided for user's application developments.

IMPORTANT:

Mightex USB Camera 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 CCD Line Camera User Manual.***

SDK FILES:

The SDK includes the following files:

\LIB directory:

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

\Documents directory:

MighTex CCD Line Camera SDK Manual.pdf

\Examples directory

\Delphi --- Delphi project example.
\VC++ --- VC++ 6.0 project example.

\Firmware directory: The latest firmware.

Note

1). The Camera engine supports Multiple Cameras, user may invoke functions to get the number of cameras currently present on USB and add the subset of the cameras into current "Working Set", all the cameras are in "Working Set" are active cameras which camera engine will get frames from them.
2). Although Line cameras are USB Devices, camera engine is **NOT** supporting Plug&Play of the cameras, it's NOT recommended to Plug or Unplug cameras 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 FILE:

The "CCD_USBCamera_SDK.h" is as following:

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

#ifdef SDK_EXPORTS
#define SDK_API extern "C" __declspec(dllexport) SDK_RETURN_CODE _cdecll
#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

#define GRAB_FRAME_FOREVER                    0x8888
```

```
typedef struct {
                                                    int Revision;
   // For Image Capture
   int Resolution;
   int ExposureTime;
   // GPIO Control
   unsigned char GpioConfigByte; // Config for Input/Output for each pin.
   unsigned char GpioCurrentSet; // For output Pins only.
} TImageControl;

typedef struct {
   int CameraID;
   int ExposureTime;
   int TimeStamp;
   int TriggerOccurred;
   int TriggerEventCount;
   int OverSaturated;
   int LightShieldPixelAverage;
} TProcessedDataProperty;


typedef TImageControl *PImageCtl;

typedef void (* DeviceFaultCallBack)( int FaultType );
typedef void (* FrameDataCallBack)(int FrameType, int Row, int Col,
                                   TProcessedDataProperty* Attributes, unsigned char *BytePtr );

// Export functions:
SDK_API CCDUSB_InitDevice( void );
SDK_API CCDUSB_UnInitDevice( void );
SDK_API CCDUSB_GetModuleNoSerialNo( int DeviceID, char *ModuleNo, char *SerialNo);
SDK_API CCDUSB_AddDeviceToWorkingSet( int DeviceID );
SDK_API CCDUSB_RemoveDeviceFromWorkingSet( int DeviceID );
SDK_API CCDUSB_StartCameraEngine( HWND ParentHandle );
SDK_API CCDUSB_StopCameraEngine( void );
SDK_API CCDUSB_SetCameraWorkMode( int DeviceID, int WorkMode );
SDK_API CCDUSB_StartFrameGrab( int TotalFrames );
SDK_API CCDUSB_StopFrameGrab( void );
SDK_API CCDUSB_ShowFactoryControlPanel( int DeviceID, char *passWord );
SDK_API CCDUSB_HideFactoryControlPanel( void );
SDK_API CCDUSB_SetExposureTime( int DeviceID, int exposureTime, bool Store );
SDK_API CCDUSB_InstallFrameHooker( int FrameType, FrameDataCallBack FrameHooker );
SDK_API CCDUSB_InstallUSBDeviceHooker( DeviceFaultCallBack USBDeviceHooker );
SDK_POINTER_API CCDUSB_GetCurrentFrame( int Device, unsigned short*  &FramePtr);
SDK_API CCDUSB_SetGPIOConifg( int DeviceID, unsigned char ConfigByte );
SDK_API CCDUSB_SetGPIOInOut( int DeviceID, unsigned char OutputByte,
                  unsigned char *InputBytePtr );.
```
// Please check the header file itself of latest information, we may add functions from time to time.

Basically, only ONE data structure *TProcessedDataProperty* data structure is defined and used for the all following functions, mainly for frame property. Note that "#pragma (1)" should be used (as above) for the definition of this structure, as DLL expects the variable of this data structure is "BYTE" alignment.

EXPORT Functions:

CCD_USBCamera_SDK.dll exports functions to allow user to easily and completely control the Line camera and get image frame. Part of the features such as firmware version queries, firmware upgrade…etc. are provided by a built-in windows, which is:

Mightex CCD Line Camera Factory Control Window

User may simply invoke **CCDUSB_ShowFactoryControlPanel()** and **CCDUSB_HideFactoryControlPanel()** to show and hide this window.

SDK_API CCDUSB_InitDevice(void);

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: None
Return: The number of Mightex CCD Line cameras currently attached to the USB 2.0 Bus, if there's no Mightex USB camera attached, the return value is 0.
Note: There's **NO** device handle needed for calling further camera related functions, after invoking CCDUSB_InitDevice, camera engine reserves resources for all the attached cameras. For example, if the returned value is 2, which means there TWO cameras 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. By default, all the devices are in "inactive" state, user should invoke *CCDUSB_AddCameraToWorkingSet(deviceID)* to set the camera as active.

SDK_API CCDUSB _UnInitDevice(void);

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

SDK_API CCDUSB_GetModuleNoSerialNo(int DeviceID, char *ModuleNo, char *SerialNo);

For a present device, user might get its Module Number and Serial Number by invoking this function.
Argument: DeviceID – the number (ONE based) of the device, Please refer to the notes of **CCDUSB_InitDevice()** function for it.
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)
1 if the call succedds.
Note: User should NOT use it while the camera engine is started.

SDK_API CCDUSB_AddDeviceToWorkingSet(int DeviceID);

For a present device, user might add it to current "Working Set" of the camera engine, and the camera becomes active.
Argument: DeviceID – the number (ONE based) of the device, Please refer to the notes of **CCDUSB_InitDevice()** function for it.
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succedds.
Note: Camera Engine will only grab frames from active cameras (the cameras in "Working Set").

SDK_API CCDUSB_RemoveDeviceFromWorkingSet(int DeviceID);

User might remove the camera from the current "Working Set", after invoking this function, the camera become inactive.
Argument: DeviceID – the number (ONE based) of the device, Please refer to the notes of **CCDUSB_InitDevice()** function for it.
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succedds.
Note: Camera Engine will only grab frames from active cameras (the cameras in "Working Set").

SDK_API CCDUSB_StartCameraEngine(HWND ParentHandle);

We have a multiple threads camera engine internally, which is responsible for all the frame grabbing, raw data processing…etc. functions. User MUST start this engine for all the following camera related operations

Argument: ParentHandle – The window handle of the main form of user's application, as the engine relies on Windows Message Queue, it needs a parent window handle which mostly should be the handle of the main window of user's application.

Return: -1: If the function fails (e.g. invalid device handle)
-2: There're different CCD Camera types (TCx_1304 and TCx_1209) in the working set.
1: The call succedds.

SDK_API CCDUSB_StopCameraEngine(void);

This function stops the started camera engine.

Argument: None.

Return: it always returns 1.

Important: For properly operating the cameras, usually the application should have the following sequence for device initialization and opening:

CCDUSB_InitDevice(); // Get the devices
CCDUSB_AddCameraToWorkingSet(deviceID);
MTUSB_StartCameraEngine();
…… Operations ……
MTUSB_StopCameraEngin();
MTUSB_UnInitDevice()

Note that we don't need to explicitly open and close the opened device, as CCDUSB_InitDevice() will actually open all the current attached cameras and reserve resources for them, however, by default, all of them are inactive, user needs to set them as active by invoking *CCDUSB_AddCameraToWorkingSet(deviceID).*

SDK_API CCDUSB_SetCameraWorkMode(int DeviceID, int WorkMode);

By default, the Camera is working in "**NORMAL"** mode in which camera deliver frames to PC continuously, however, user may set it to **"TRIGGER"** Mode, in which the camera is waiting for an external trigger signal and capture ONE frame for each trigger signal.

Argument: *DeviceID*e – the device number which identifies the camera.
WorkMode – 0: **NORMAL** Mode, 1: **TRIGGER** Mode.

Return: -1 If the function fails (e.g. invalid device number)
1 if the call succeeds.

Note: Basically, **NORMAL** mode and **TRIGGER** mode are the same, the only difference is that:

NORMAL mode – When Camera is in this mode, camera will always grab frame as long as there's available memory for a new frame, while host (in most cases, it's a PC) is continuing get frame from the camera, the camera is continuously grabbing frames from CCD sensor.

TRIGGER mode – When Camera is in this mode, camera will only grab a frame from CCD sensor while there's an external trigger asserted (and there's available memory for a new frame), in this case, host will usually poll the camera for a new coming frame.

Note: This function has an important side-effect that it will empty the on-camera frame buffer, it will let camera firmware to neglect the grabbed frames in frame buffer. This is very useful (especially in NORMAL mode), while user wants to get a new frame. (the frames in buffer might be "old" frames).
For example, if user wants to get a single new frame in NORMAL mode, user might do:

```
CCDUSB_SetCameraWorkMode( 1, 0); // 0 is for NORMAL mode, the camere might be  in NORMAL
                                 // orginally, so this  is only for cleanning up the frame buffer.
CCDUSB_StartFrameGrab( 1); // Get one frame only, user might put frame number other than 1 too.
```

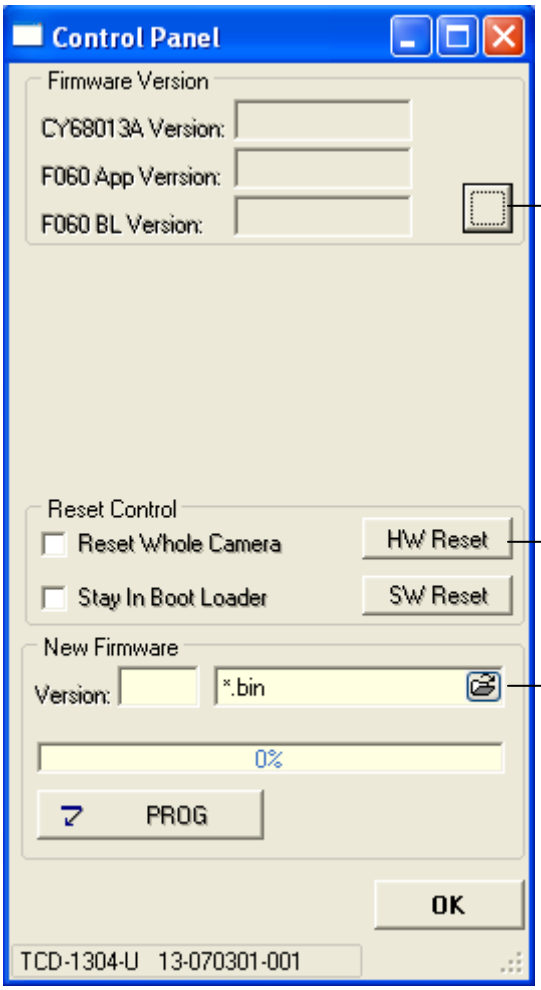
SDK_API CCDUSB_StartFrameGrab(int TotalFrames);
SDK_API CCDUSB_StopFrameGrab(void);

When camera engine is started, the engine prepares all the resources, but it does NOT start the frame grabbing , until CCDSB_StartFrameGrab() function is invoked. After it's successfully called, camera engine starts to grab frames from cameras in current "Working Set". User may call CCDUSB_StopFrameGrab() to stop the engine from grabbing frames from camera.

Argument: TotalFrames – This is for CCDUSB_StartFrameGrab() only, after grabbing this frames, the camera engine will automatically stop frame grabbing, if user doesn't want it to be stopped, set this number to 0x8888, which will doing frame grabbing forever (in NORMAL mode), until user calls CCDUSB_StopFrameGrab().
Return: -1 If the function fails (e.g. invalid device number or if the engine is NOT started yet)
1 if the call succedds.

SDK_API CCDUSB_ShowFactoryControlPanel(int DeviceID, char *passWord);

For user to access the factory features conveniently and easily, the library provides its second dialog window which has all the features on it.
Argument: DeviceID – the device number.
Password – A pointer to a characters which is the password, "661016" is recommended in most cases.
Return: -1 If the function fails (e.g. invalid device number)
1 If the call succeeds.



Click to button will get firmware version information from camera.

These two buttons are used for Reset the camera, they're mainly used for upgrading the firmware, please refer to the user manual for the details. Note that it's applicable for TCN-1304-U modal only.

User can upgrade firmware with those controls, for details, please refer to the user manual.

SDK_API CCDUSB_HideFactoryControlPanel(void);

This function hides the factory control panel, which is shown by invoking **CCDUSB_ShowFactoryControlPanel().**
Argument: None.
Return: it always returns 1.

SDK_API CCDUSB_SetExposureTime(int DeviceID, int exposureTime, bool Store);

User may set the exposure time by invoking this function.
Argument: DeviceNo – the device number which identifies the camera will be operated.
exposureTime – the Exposure Time is set, note it's in "Microsecond", and as the camera's minimum resolution for exposure is 100us (or 300us), so it should be multiple of 100us (including 100us).
Store – Whether camera should put it in its NV memory, 0 means "Do not put", 1 means "Put".
Return: -1 If the function fails (e.g. invalid device number)
1 if the call succedds.
Important: In most cases, it's recommended to set the third parameter as "0", and let the PC to memorize and set back camera's parameters (including working mode and exposure time) while it starts the application (e.g. after initialize the camera engine).
Note: For TCN-1209-U modal, the minimum exposure time is actually 300us, so if user set ET to 100us or 200us, the camera will use the minimum ET (300us) instead.

SDK_API CCDUSB_InstallFrameHooker(int FrameType, FrameDataCallBack FrameHooker);
Argument: FrameType – 0: Raw Data
1: Processed Data.
FrameHooker – Callback function installed.
Return: -1 If the function fails (e.g. invalid Frame Type).
1 if the call succedds.
Important: The call back function will only be invoked while the frame grabbing is started, host will be notified every time the camera engine get a new frame (from any cameras in current working set).
Note:
1). Current there's NO difference between Raw Data and Processed Data, actually, no matter the FrameType is, camera engine always return the ADC data (16bit) of the frame.
2). The callback has the following prototype:
typedef void (FrameDataCallBack)(int FrameType, int Row, int Col,*
TProcessedDataProperty Attributes, unsigned char *BytePtr);*

The TProcessedDataPropertys defined as:
```
typedef struct {
   int CameraID;
   int ExposureTime;
   int TimeStamp;
   int TriggerOccurred;
   int TriggerEventCount;
   int OverSaturated;
   // Since V1.1.0.0, we have the following new field
   int LightShieldPixelAverage;
} TProcessedDataProperty;
```

Arguments of Call Back function:
FrameType – The same value passed in when the callback in stalled.
Row, Col – the Row and Column size of the frame, in our case, it's a Line camera, we always have
Row = 1, Col = 3648 or 2048.
Attributes – This is an important data structure which contains information of this particular frame, it has the following elements:
CameraID – This is the camera number (the same as the deviceID used in all the APIs), as camera engine might get frames from more than one cameras (there might be multiple cameras in current working set), this identifies the camera which generates the frame.

Exposure Time – The exposure time camera was used for generating this frame, as there's frame buffer on device, and on PC camera engine, it's especially useful for application to know the exposure time of a certain frame during the PC is adjusting a camera's exposure time.

TimeStamp – Camera firmware will mark each frame with a time stamp, this is a number from 0 – 65535 (and it's automatically round back) which is generated by the internal timer of the firmware, the unit of it is 100us (which is also the minimum step for exposure time). For example, if one Frame's time stamp is 1000 and the next frame's stamp is 1200, the time interval between them is 200x100us = 20ms.

TriggerOccurred – While the camera is in **NORMAL** mode, camera is grabbing frames continuously (as long as host is fetching frames from it), If an external trigger signal asserted during a frame grabbing, camera will set this flag to ONE (otherwise it's ZERO). This gives host an choice to poll the trigger assertion even it's in **NORMAL** mode. This flag should NOT be used while it's in **TRIGGER** mode.

TriggerEventCount – While the camera is in **TRIGGER** mode, camera will grab ONE frame after it's triggered by the external signal, each trigger will increase this count by ONE, this gives host a clue for the frame and its trigger signal. Note that this count is reset to ZERO whenever host set the work mode to TRIGGER (so host can reset the count by invoking **CCDUSB_SetCameraWorkMode(DeviceID, 1),** even the camera is already in **TRIGGER** mode). Also note that the frame read time is around 7.5ms (3694 x 2us) for TCN-1304-U and ~135us for TCN-1209-U, if trigger signals are generated more frequently than this interval, the count is still physically reflect the trigger signal assertions. For example, the TriggerEventCount for a frame is 1, next frame's TriggerEventCount might be 6, which means assertion 2 to 5 are occurred during the reading time of first frame…and they're actually ignored. The 6th assertion occurs right after the reading of the frame, so it generates the next frame grabbing.

OverSaturated – 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.

LightShieldPixelAverage – The CCD provide 13 pixels (or 16 pixels) with Light shielded, this field provides an average value of these pixels, user might use this value for black compensation. Note that this field is added since library V1.1.0.0, however, as the callback returns a pointer to *TprocessedDataProperty, this new field won't affect the application developed based on previous version of DLL.*

BytePtr – The pointer to the memory buffer which holds the frame data, although it's a pointer to Byte, the data pointed by it is actually 16bit words. (3694 words or 2048 words)

Note that this callback is invoked in the main thread of the host application, GUI operations are allowed in this callback, however, blocking in this callback will slow down the frame rate of the camera engine.

SDK_API CCDUSB_InstallUSBDeviceHooker(DeviceFaultCallBack USBDeviceHooker);

User may call this function to install a callback, which will be invoked by camera engine while camera engine encounter camera faults.

Argument: USBDeviceHooker – the callback function registered to camera engine.

Return: It always return 1.

Note:

1). The camera engine doesn't support Plug&Play of the cameras, while the camera engine is starting work, any plug or unplug of the CCD Line cameras is NOT recommended. If plug or unplug occurs, the camera engine will stop its grabbing and invoke the installed callback function. This notifies the host the occurrence of the device configuration change and it's recommended for host to arrange house clean works. Host might simply do house keeping and terminate OR host might let user to re-start the camera engine. (Please refer to the Delphi and VC++ example code for this)

2). The callback function has the following prototype:

typedef void (DeviceFaultCallBack)(int FaultType);*

The FaultType is always ZERO in current version.

SDK_POINTER_API CCDUSB_GetCurrentFrame(int Device, unsigned short* &FramePtr);

User may call this function to get the one frame, this is the latest frame while host invokes this API..

Argument : DeviceID – The device number, which identifies the camera's frame is being grabbed.

Return: NULL: If the function fails

Valid Pointer: A valid pointer to an internal array containing the frame property and frame image, this pointer points to the same memory as FramePtr.

Important: User might use this API to get a frame of the specified device, this is an easy way to grab frame data, however, when user uses this API, it's not guarantee each frame can be captured, the returned frame data is the latest

frame at the moment the API is invoked. The returned pointer is as both the function return valu 俄 and the 2nd argument, it's a pointer points to a "unsigned short" (we call it "word") array, the array has two parts as following:
FrameDataProperty: It occupies the first 8 words which contains the 7 fields in *TprocessedDataProperty* and one more reserved word (always ZERO).
FrameRawData: After the 8 words, the following words are raw data of the frame, note that for TCX-1304-X, it has 3648 words, while for TCX-1209-X, it has 2048 words only.
This function might help customer to get frame data with programming tools which is hard to implement the "Callback" mechanism (e.g. LabView, please refer to the LabView example for it). For programming tools such as VC++, Delphi or VB, we recommend user to use "callback" mechanism, which guarantees the callback of each frame.

SDK_API CCDUSB_SetGPIOConifg(int DeviceID, unsigned char ConfigByte);

User may call this function to configure GPIO pins.
Argument : DeviceID – The device number, which identifies the camera is being operated.
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 CCDUSB_SetGPIOInOut(int DeviceID, unsigned char OutputByte,
unsigned char *InputBytePtr);

User may call this function to set GPIO output pin states and read the input pins states.
Argument : DeviceID – The device number, which identifies the camera is being operated.
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.