

Mightex USB Camera SDK Manual

Mightex USB 2.0 color camera is mainly designed for microscopy and other scientific applications, in which cost-effective and ease of use are important. With USB 2.0 high speed interface and powerful PC software processing, the camera delivers excellent quality images at high frame rate. GUI 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 USB Camera User Manual](#).**

SDK FILES:

The SDK includes the following files:

\LIB directory:

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

\Documents directory:

MighTex USB Camera SDK Manual.pdf

\Examples directory

\Delphi	--- Delphi 5.0 project example.
\VC++	--- VC++ 6.0 project example.

\Firmware directory: The latest firmware.

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

HEADER FILE:

The "MT_USBCamera_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
#else
#define SDK_API extern "C" __declspec(dllimport) SDK_RETURN_CODE _cdecl
#define SDK_HANDLE_API extern "C" __declspec(dllimport) DEV_HANDLE _cdecl
#endif

#define RAWDATA_IMAGE 0
#define BMPDATA_IMAGE 1

#pragma pack(1)
typedef struct {
    int Revision;
    // For Image Capture
    int Resolution;
    int BinMode;
}
```

```

    int XStart;
    int YStart;
    int GreenGain;
    int BlueGain;
    int RedGain;
    int MaxExposureTimeIndex;
    int ExposureTime;
    // For Image Rendor
    bool ImageRendorFitWindow;
    int Gamma;
    int Contrast;
    int Bright;
    int SharpLevel;
    bool BWMode;
    bool HorizontalMirror;
    bool VerticalFlip;
    // For Capture Files.
    int CatchFrames;
    bool IsAverageFrame;
    bool IsCatchRAW;
    bool IsRawGraph;
    bool IsCatchJPEG;
    bool CatchIgnoreSkip;
} TImageControl;
#pragma pack()

typedef TImageControl *PImageCtl;
typedef void (* CallbackFunc)( int ImageSequenceNo, char *FileName );

// Export functions:
SDK_API MTUSB_InitDevice( void );
SDK_API MTUSB_UnInitDevice( void );
SDK_HANDLE_API MTUSB_OpenDevice( int deviceID );
SDK_HANDLE_API MTUSB_ShowOpenDeviceDialog( void );
SDK_API MTUSB_GetModuleNo( DEV_HANDLE DevHandle, char *ModuleNo );
SDK_API MTUSB_GetSerialNo( DEV_HANDLE DevHandle, char *SerialNo );
SDK_API MTUSB_StartCameraEngine( HWND ParentHandle, DEV_HANDLE DevHandle );
SDK_API MTUSB_StopCameraEngine( DEV_HANDLE DevHandle );
SDK_API MTUSB_SetCameraWorkMode( DEV_HANDLE DevHandle, int WorkMode );
SDK_API MTUSB_SetExternalParameters( DEV_HANDLE DevHandle, bool AutoLoop, bool IsRawGraph;
    bool IsJPEG, char *FilePath, char *FileName);
SDK_API MTUSB_WaitingExternalTrigger( DEV_HANDLE DevHandle, bool StartWait, CallbackFunc Aproc );
SDK_API MTUSB_ShowFrameControlPanel( DEV_HANDLE DevHandle, bool IsTriggerModeAllow, char *Title, int
    Left, int Top);
SDK_API MTUSB_HideFrameControlPanel( DEV_HANDLE DevHandle );
SDK_API MTUSB_ShowVideoWindow( DEV_HANDLE DevHandle, int Top, int Left, int Width, int Height );
SDK_API MTUSB_StartFrameGrab( DEV_HANDLE DevHandle );
SDK_API MTUSB_StopFrameGrab( DEV_HANDLE DevHandle );
SDK_API MTUSB_GetFrameSetting( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetFrameSetting( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetResolution( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetStartPosition( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetGain( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetExposureTime( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetGammaValue( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetGammaTable( DEV_HANDLE DevHandle, unsigned char *GammaTable );
SDK_API MTUSB_SetShowMode( DEV_HANDLE DevHandle, PImageCtl SettingPtr);
SDK_API MTUSB_SetWhiteBalance( DEV_HANDLE DevHandle );
SDK_API MTUSB_SetFrameRateLevel( DEV_HANDLE DevHandle, int RateLevel );
SDK_API MTUSB_GetCurrentFrameRate( DEV_HANDLE DevHandle );
SDK_API MTUSB_GetLastBMPFrame( DEV_HANDLE DevHandle, char *FileName );
SDK_API MTUSB_GetCurrentFrame( DEV_HANDLE DevHandle, int FrameType, unsigned char *Buffer );

```

```
SDK_API MTUSB_SaveFramesToFiles( DEV_HANDLE DevHandle, PImageCtl SettingPtr,
char *FilePath, char *FileName );
```

Basically, only ONE data structure TimageControl data structure is defined and used for the all following functions, mainly for camera parameters setting. 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:

MT_USBCamera_SDK.dll exports functions to allow user to easily and completely control the various parameters of frame grabbing, image render and snapshot catching. For user’s quick development of application, the DLL has three built in windows, which are:

- Mightex USB Camera Device Open Dialog Window
- Mightex USB Camera Full Control Panel Dialog Window
- Mightex USB Video Window

The first two windows are not necessarily used if user wants to have his own GUI for similar purposes, there’re sets of other functions which provides equivalent features, however, by using these two windows, especially the “Control Panel” window, it’s extremely easy and quick to develop an application, this is a time-saving solution.

SDK_API MTUSB_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 USB cameras currently attached to the USB 2.0 Bus, if there’s no Mightex USB camera attached, the return value is 0.

SDK_API MTUSB_UnInitDevice(void);

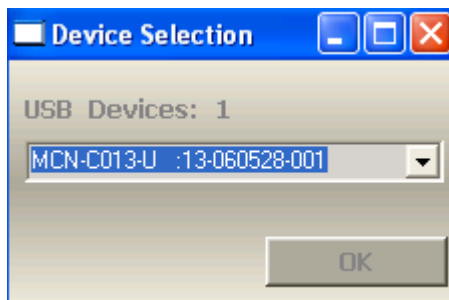
This is the function to release all the resources reserved by MTUSB_InitDevice(), user should invoke it before application terminates.

Arguments: None

Return: Always return 0.

SDK_HANDLE_API MTUSB_ShowOpenDeviceDialog(void);

User may call this function to show the Device Open Dialog, which lets user to select the camera will be operated. The dialog is as following:



It will show all the attached cameras in ModuleNo:SerialNo format in the pull down combo box, user may select the one he wants to operate and click the [OK] button.

Argument: None.

Return: The handle of the opened device.

Note that the device handle returned is an index to the internal data structure, and it will be used as the first parameter for all other device operation functions:

Important: The current device driver only support **ONE** opened device, opening a device will close the previous opened device automatically (if there's an opened one), so if there's more than one Mightex USB cameras attached, user have to switch between them for operations.

SDK_HANDLE_API MTUSB_OpenDevice(int deviceID);

If user doesn't want to use the previous Open Device Dialog for opening a selected device, user may use This function to open the device.

Argument: deviceID – this is the index of the device, it's a ZERO based index, for example, while invoking MTUSB_InitDevice() and it returns 2 (the number of devices currently attached), deviceID can be 0 or 1, means the first and the second device.

Return: Device Handle

Important: See the notes on the previous function description for the device handle.

SDK_API MTUSB_GetModuleNo(DEV_HANDLE DevHandle, char *ModuleNo);

For an opened device, user might get its Module Number by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
ModuleNo – 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 handle)
1 if the call succeeds.

SDK_API MTUSB_GetSerialNo(DEV_HANDLE DevHandle, char *SerialNo);

For an opened device, user might get its Serial Number by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
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 handle)
1 if the call succeeds.

SDK_API MTUSB_StartCameraEngine(HWND ParentHandle, DEV_HANDLE DevHandle);

We have a multiple threads camera engine internally, which is responsible for all the frame grabbing, raw data to RGB data conversion...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.
DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

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

Important: After starting the camera engine, the application will use considerable amount of the system resources (e.g. RAM), For the PC running the application, it's recommended to be Pentium IV, 1.5G or up and have 512M memory. (Please refer to product spec. for the minimum requirement of the PC).

SDK_API MTUSB_StopCameraEngine(DEV_HANDLE DevHandle);

This function stops the started camera engine.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

Return: -1 If the function fails (e.g. invalid device handle or if the engine is NOT started yet)
1 if the call succeeds.

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

```

MTUSB_InitDevice(); // Get the devices
MTUSB_OpenDevice(); // Using the device index returned by the previous MTUSB_InitDevice() call.
MTUSB_StartCameraEngine(); // Using the device handle returned by MTUSB_OpenDevice()
..... Operations .....
MTUSB_StopCameraEngin();
MTUSB_UnInitDevice()

```

Note that we don't need to explicitly close the opened device, because:

- 1). If user want to open another device, open device will automatically close the previous opened device,
- 2). MTUSB_UnInitDevice() will close the opened device, and release all other resources.

SDK_API MTUSB_SetCameraWorkMode(DEV_HANDLE DevHandle, int WorkMode);

By default, the Camera is working in “**Video**” mode in which camera deliver frames to PC continuously, however, user may set it to “**External Trigger**” Mode, in which the camera is waiting for an external trigger signal and capture ONE frame of image.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
WorkMode – 0: Video Mode, 1: External Trigger Mode.

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

SDK_API MTUSB_SetExternalWaitingExternalTrigger(DEV_HANDLE DevHandle, bool AutoLoop, bool IsRawGraph, bool IsJPEG, char *FilePath, char *FileName);

While the camera is in “External Trigger” Mode, invoking this function starting the waiting for external signal.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
AutoLoop – Whether is “Automatic Loop” or only One Shot Wait.
IsRawGraph – If it's true, the saved BMP or JPEG file is the raw graph which is not adjusted by Gamma, Contrast, Bright or Sharp settings.
IsJPEG – Is the saved file in JPEG format (other than BMP format).
FilePath, FileName – the directory and name of the saved file. Note that for path, the ending “\” is NOT needed, for filename, the extension (jpg or bmp) is NOT needed.

Return: -1: If the function fails (e.g. invalid device handle or camera is NOT in External Trigger Mode or it's during waiting for external trigger)
1: Call succeeds.

SDK_API MTUSB_WaitingExternalTrigger(DEV_HANDLE DevHandle, bool StartWaiting, CallBackFunc Aproc);

While the camera is in “External Trigger” Mode, invoking this function starting the waiting for external signal.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
StartWaiting – True: Start to wait for external trigger signal, .
False: Abort the waiting state if it's in it.
Aproc – Call back installed by caller, it's invoked after a frame is caught.

Return: -1: If the function fails (e.g. invalid device handle or camera is NOT in External Trigger Mode)
1: Call succeeds.

Important: This function will return immediately, while an external trigger occurs and a frame is caught, the call back function will be invoked, If “AutoLoop” is NOT set, user has to invoke this function again (with StartWaiting set to True) to start next catch, If “AutoLoop” is set, camera engine will still in “waiting” state and the call back function will be invoked repeatedly after each frame is caught. The call back function has two arguments: Frame Sequential number(make sense only when autoloop is set) and image file name (for the image just caught).

Calling this function with StartWaiting set to False will notify camera engine to abort the wait if it's still in "waiting" state (this can be either "AutoLoop" is set, OR although it's one-shot mode, but there's no image caught yet), in this case the call back will still be invoked with first argument (Frame sequential number) set to ZERO.

A brief summary for External mode:

The camera can be set in "External Trigger" Mode, the camera engine has the following behaviors in this mode:

AutoLoop is set to TRUE:

*In this case, the camera engine will always stay in "waiting" state, each time it captured a frame (triggered by external signal), camera engine invokes the callback (if it's installed). The only way out of "waiting" state is to Invoke the **MTUSB_WaitingExternalTrigger()** function with the StartWaiting set to FALSE, this notifies The engine to exit from "waiting" state. Camera engine will also call the callback with the first argument set to ZERO. After it's aborted, the callback function will be unhooked from camera engine automatically, which means next time Host must re-install the hooker by calling **MTUSB_WaitingExternalTrigger()**.*

AutoLoop is set to FALSE

*In this case, Camera engine is in "One-Shot Waiting" state, which means it will wait until a frame is captured (triggered by an external signal) or user abort it. In both cases, the callback (if installed) will be invoked, and the callback will be unhooked from camera engine, which means user has to reinstall the callback next time by call the **MTUSB_WaitingExternalTrigger()** function.*

SDK_API MTUSB_ShowFrameControlPanel(DEV_HANDLE DevHandle, bool IsTriggerModeAllow, char *Title, int Left, int Top);

For user to develop application conveniently and easily, the library provides its second dialog window which has all the camera controls on it, if user use this window in his application, it's NOT necessarily to use most of other functions.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

IsTriggerModeAllow – Set to control whether the Trigger Mode Selection is visible on control panel. We provide this parameter for user doesn't want to have "External Trigger" mode available on control panel.

Title – The Title will be displayed on the control panel.

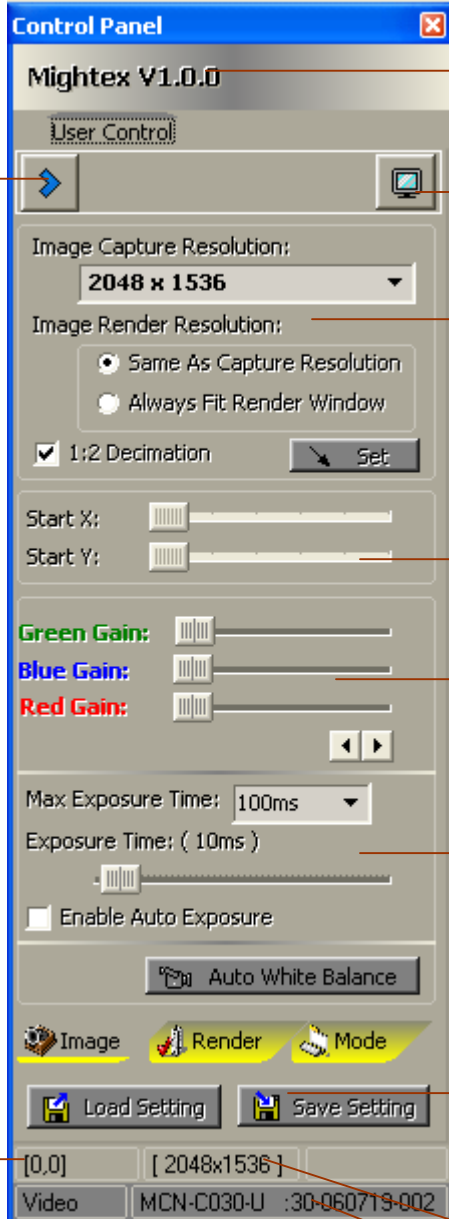
Left, Top – the Top-Left position of the control panel.

Return: -1 If the function fails (e.g. invalid device handle)

1 If the call succeeds.

Important: Close this control panel will close the whole application (it will post a message of SC_CLOSE to it's parent window), so if user wants to have this control panel shown in application, the parent window is NOT necessarily visible. If user wants to hide this panel, don't close it, but invoke **MTUSB_HideFrameControlPanel()** function instead.

The panel is as following, in the "Image" page, it has the controls:



Start (and Stop) Frame Grab button

The Title, I use "Mightex V1.0.0" as example

The "Show Video Window" button, click it show the Video window.

User may select the resolution here, currently, the provided resolution includes 32x32, ..., 640x480, 800x600, 1024x768 and 1280x1024(for 1.3M series), And the selectable "1:2 Decimation"(2x skip) mode. User may also select the resolution of rendering, it can be always fit the video window, OR always keep the same resolution as the capture image. User must use **Set** button to set the settings to camera engine. (Note that for minimum resolution 32x32, the 1:2 Decimation is not allowed)

User may use these two slider to select the start position of the capture image. (ROI feature) While it's NOT in full resolution.

User may use these three slider to manually adjust the RGB gains (0x – 16x). The **Auto** is used for adjusting all gains (RGB gains) proportionally.

User may use these controls to select the maximum exposure time range and the current exposure time, the **Auto White Balance** button is used for Automatic White Balance (AWB) set, user needs to set proper exposure time and put a white paper as the object, click this button will automatically set the RGB gains to get ideal white color. The "Enable Auto Exposure" checkbox allows user to enable auto exposure feature.

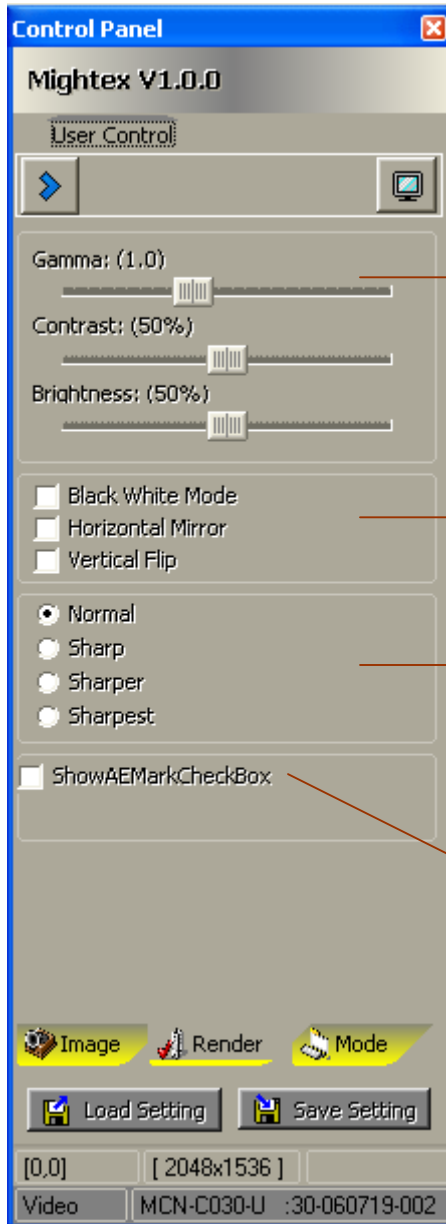
These two buttons are used to save/load all the current settings to/from a user defined file, user may set proper parameters (exposure time, gains...etc.) under a certain environment and save the parameters to a file named this environment, e.g. "Sunny Outside.set" or "In Room.set", And user may load them back later.

Start Position of ROI

Current selected resolution and frame rate.

The opening device's Module No. and Serial No.

The control panel has the second page of “Render” as following:



Gamma, Contrast and Brightness control for the video window.

User may set the display frame (in video window) in “Black and White” mode, “Horizontal Mirror” mode and “Vertical Flip” mode by checking these boxes.

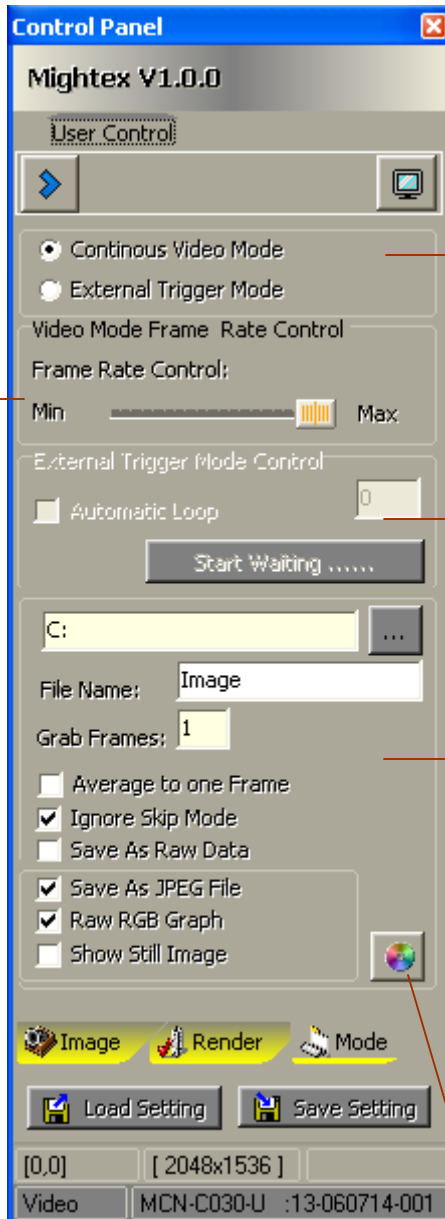
User may select the “Sharp” level, however, as the Sharp algorithm needs considerable PC resources, while selecting Sharp, Sharper or Sharpest, PC’s resource will be almost 100% occupied by the it. Not only the frame rate will reduce significantly and that will affect the running speed of other PC applications.

So it’s recommended to use “Normal” in most cases.

ShowAEMarkCheckBox will always show the auto exposure detecting area on the image, while auto exposure feature is enabled, camera will detect the illumination of this area and figure the optimized exposure time.

The control panel has the third page of “Mode” as following:

In Video Mode, the actual frame rate is very depending on the PC resources, while we set the default of each resolution to “Max” frame rate, that will give user the maximum frame rate, however, that might use all CPU’s resources (for data collection from USB and image data processing), user may use this slider to adjust the frame rate to modest level, which lets PC has some bandwidths for other applications. As different PC may have different frame rates under the same settings of this slider, this slider gives user a comparative control on the particular PC which is running the application. The actual frame rate is shown on the pane below.



Camera Mode selection, the default is “Video” mode, user may also select “External Trigger” mode.

While camera is in “External Trigger” mode, user can click **Start Waiting** button, that will make software keeps to wait for an external signal (hard connected to the external trigger pins on camera’s connector), a falling edge of the trigger signal will start ONE frame of snapshot catching, and PC will save it to the specified location. If “Automatic Loop” is checked, the software will remain in the “waiting” state even after a frame is captured, waiting for the next capture. If “Automatic Loop” is NOT checked, it finishes the “waiting” after ONE capture, and user has to re-click the button for next capture.

The directory to store the captured files. While in “Video” mode, the camera engine continuously grabbing frames from camera, and user may ask it to save the grabbed frames to file in “Raw”, “JPEG” or “BMP” format. In “External Trigger” mode, only **ONE** frame in “JPEG” or “BMP” format can be saved for each external trigger.

User can specifies directory, filename and file number, as well as other settings:

- “**Average to one frame**” – Save one image only, but it’s the average of all grabbing frames.
- “**Ignore Skip Mode**” – user can check it if user wants to get a full resolution frame (e.g. 1280x1024), while the current video resolution is in 1:2 Decimation mode (e.g. 1280x1024 with 1:2 decimation).
- “**Save As Raw Data**” – the Frame is save as RAW data file (Only valid for “Video” mode).
- “**Save As JPEG file**” – Save file as JPEG image.
- “**Raw RGB Graph**” – Ignore the Gamma/Contrast/Bright adjustments)
- “**Show Still Image in Form**” – If this is checked, a still image window will be display after frames grabbing, and user can view them instantly. (Valid for “Video” mode only)

In “Video” mode, user can use this button for taking snapshots.

Important: After the camera engine is started, the control panel is created and hided, user can use this function to show it up, and it’s always a good idea to show it during the development time, even user don’t want it to be shown in final application GUI.

Please refer to the examples (VC++ or Delphi examples) for the using of the control panel, as well as other APIs.

SDK_API MTUSB_HideFrameControlPanel(DEV_HANDLE DevHandle);

This function hides the control panel, note the control panel is always there once the camera engine is started, hiding it only make it invisible.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

Return: -1 If the function fails (e.g. invalid device handle or if the engine is NOT started yet)
1 if the call succeeds.

SDK_API MTUSB_ShowVideoWindow(DEV_HANDLE DevHandle, int Top, int Left, int Width, int Height);


This function shows the video window, user may customize it's position and size with the input arguments.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

Left, Top – the Top-Left position of the video window.

Width, Height – The width and height of the video window.

Return: -1 If the function fails (e.g. invalid device handle or if the engine is NOT started yet)
1 if the call succeeds.

Important: On control panel, there's a button  corresponding to this function. The video window is shown as following:





Note that while the render resolution is chosen as "Same as Capture Resolution", the Video window may not be big enough to show the whole frame, user can move the image around by moving the mouse with the left button down.

SDK_API MTUSB_StartFrameGrab(DEV_HANDLE DevHandle);
SDK_API MTUSB_StopFrameGrab(DEV_HANDLE DevHandle);

When camera engine is started, in Video mode, the engine prepares all the resources, but it does NOT start the frame grabbing , until MTUSB_StartFrameGrab() function is invoked. After it's successfully called, user should see video on the video window (if it's showed). User may call MTUSB_StopFrameGrab() to stop the engine from grabbing frames from camera.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

Return: -1 If the function fails (e.g. invalid device handle or if the engine is NOT started yet)
 1 if the call succeeds.

Important: On control panel, there's a button  corresponding to "MTUSB_StartFrameGrab()", and after it's started the button becomes , click it invoking "MTUSB_StopFrameGrab()".

SDK_API MTUSB_GetFrameSetting(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may get the current set of parameters by invoking this function, please note that the TImageControl data structure contains all the parameters for controlling Frame Grabbing, Video rendering and File savings.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
 SettingPtr – the Pointer to variable of TImageControl structure.

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

Important:

- 1). There're two ways to change those settings, from the control panel OR by calling the following MTUSB_Setxxx functions, either way, the settings returned from this function are the current latest settings of the camera engine.
- 2). The TImageControl data structure has the following elements, please refer to the c style comments in /* */ for their definition:

```
typedef struct {
    int Revision; /* Reserved for internal use only */

    // For Image Capture
    int Resolution; /* This is an index to resolution settings, we have the following definition for it:
        0 – 32 x 32
        1 – 64 x 64
        2 – 160 x 120
        3 – 320 x 240
        4 – 640 x 480
        5 – 800 x 600
        6 – 1024 x 768
        7 – 1280 x 1024
        8 – 1600 x 1200 For 3M Camera only
        9 – 2048 x 1536 For 3M Camera only
        */
    int BinMode; /* 1 – No Skip mode, 2 – 2X skip or binning mode (1:2 decimation) */
    int XStart; /* Start Column of the ROI, should be even number or a value of multiple of 4 when it's in Skip mode*/
    int YStart; /* Start Row of the ROI, should be even number or a value of multiple of 4 when it's in Skip mode*/

    int GreenGain; /* Green Gain Value: 0 – 128, the actual gain is GreenGain/8 */
    int BlueGain; /* Blue Gain Value: 0 – 128, the actual gain is BlueGain/8 */
    int RedGain; /* Red Gain Value: 0 – 128, the actual gain is RedGain/8 */
    int MaxExposureTimeIndex; /* The index for maximum exposure time:
        0 – 5ms
        1 – 10ms
        2 – 100ms
        3 – 750ms
    */
};
```

```

        */
int ExposureTime; /* The current exposure time in Micro second, e.g. 10000 means 10ms */

// For Video image render
bool ImageRenderFitWindow; /* True if the image always fit video window, False if the image will keep the same
                             resolution as the grabbing resolution
        */
int Gamma; /* Gamma value: 0 – 20, means 0.0 – 2.0 */
int Contrast; /* Contrast value: 0 – 100, means 0% -- 100% */
int Bright; /* Brightness : 0 – 100, means 0% -- 100% */
int SharpLevel; /* SharpLevel: 0 – 3, means Normal, Sharp, Sharper and Sharpest */
bool BWMode; /* Black White mode? */
bool HorizontalMirror; /* Horizontal Mirror? */
bool VerticalFlip; /* Vertical Flip? */

// For Capture Files.
int CatchFrames; /* Number of frames to be captured */
bool IsAverageFrame; /* Save only one frame, but it's the average of all grabbed frames */
bool IsCatchRAW; /* Save as RAW Data File? */
bool IsRawGraph; /* Save as JPG or BMP, but not corrected by Gamma, contrast, bright and sharp algorithm */
bool IsCatchJPEG; /* Save as JPEG File? */
bool CatchIgnoreSkip; /* Always capture full resolution? */
} TImageControl;

```

SDK_API MTUSB_SetFrameSetting(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the all parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: This function set all of the parameters of SettingPtr to camera engine, and is effective immediately after the call, if the frame grabbing is started, it's immediately affected by those settings.

SDK_API MTUSB_SetResolution(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the resolution (including capture and render) parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: Although the second input argument is a pointer to TimageControl structure, only three elements “Resolution”, “BinMode” and “ImageRenderFitWindow “ are used by this function, all others are ignored.

SDK_API MTUSB_SetStartPosition(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the start position of ROI parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: Although the second input argument is a pointer to TimageControl structure, only two elements “XStart” and “YStart “ are used by this function, all others are ignored.

SDK_API MTUSB_SetGain(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set RGB Gains parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: Although the second input argument is a pointer to TImageControl structure, only three elements “GreenGain”, “BlueGain” and “RedGain “ are used by this function, all others are ignored.

SDK_API MTUSB_SetExposureTime(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the exposure time parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: Although the second input argument is a pointer to TImageControl structure, only two elements “MaxExposureTimeIndex” and “ExposureTime “ are used by this function, all others are ignored.

SDK_API MTUSB_SetGammaValue(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the Gamma, Contrast and Brightness parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
SettingPtr – the Pointer to variable of TImageControl structure.

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

Important: Although the second input argument is a pointer to TImageControl structure, only four elements “Gamma”, “Contrast”, “Bright “ and “SharpLevel” are used by this function, all others are ignored.

SDK_API MTUSB_SetGammaTable(DEV_HANDLE DevHandle, unsigned char *GammaTable);

User may set the internal Gamma Table by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
GammaTable – the Pointer to 256 bytes array which contains the Gamma table.

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

Important: Camera engine has an internal Gamma table to do the Gamma correction, all the output value is get as GammaTable[InputValue], while InputValue is the ADC value read from CMOS sensor.

SDK_API MTUSB_SetShowMode(DEV_HANDLE DevHandle, PImageCtl SettingPtr);

User may set the BWMode, HorizontalMirror and VerticalFlip parameters by invoking this function.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
SettingPtr – the Pointer to variable of TImageControl structure.

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


Important: Although the second input argument is a pointer to TImageControl structure, only three elements “BWMode”, “HorizontalMirror” and “VerticalFlip “ are used by this function, all others are ignored.

SDK_API MTUSB_SetWhiteBalance(DEV_HANDLE DevHandle);

User may call this function for Automatic White Balance set.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

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

Important: This is the equivalent to the button of  Auto White Balance in control panel, note that user should set proper exposure time and put a white paper at proper distance before this function is invoked.

SDK_API MTUSB_SetFrameRateLevel(DEV_HANDLE DevHandle, int RateLevel);

User may call this function to set the current frame grabbing rate.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
RateLevel – Can be from 0 – 10, while 0 means minimum frame rate, 10 means maximum rate.

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

Important: In the current design, the actual frame rate is mainly depending on the PC resources. The frame rate might be different on a slow PC and a fast PC, the default setting of the camera engine is to set the Maximum frame rate, however, that might not be ideal as almost all the CPU resources will be used by camera engine, which will make other PC applications “hunger” of CPU time, user might want to reduce the frame rate a little bit to politely give other application time to run.

SDK_API MTUSB_SetAutoExposure(DEV_HANDLE DevHandle, bool AutoExposureOn, bool ShowExposureMark);

User may call this function to set the current frame grabbing rate.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
AutoExposureOn – True/False to turn the “Auto Exposure” feature On/Off.
ShowExposureMark – True/False to show/hide the Exposure Mark.

Return: -1 If the function fails (e.g. invalid device handle or camera engine is in “External Trigger” mode)
1 if the call succeeds.

SDK_API MTUSB_GetCurrentFrameRate(DEV_HANDLE DevHandle);

User may call this function to get the current frame grabbing rate.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()

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

Important: This is an average frame rate of the current grabbing, as PC is not a real time system, which might switch to other applications, so the actual frame rate is vary a little bit from time to time. This function returns the frame rate at the calling moment. With the control panel, frame rate is also shown on the bottom right corner.

SDK_API MTUSB_GetLastBMPFrame(DEV_HANDLE DevHandle, char *FileName);

User may call this function to get the bitmap format frame of the last captured frame.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
FileName – the full file name for the bitmap file.

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

Important: While the frame grabbing is running OR it’s stopped, we can always get the last (for the time this function is invoking) frame of the Video window in Bitmap format. Note that this function may mainly be used in situation of user stop the video as the frame is exactly the user’s interesting. Note that this bitmap frame is adjusted with user’s setting of Gamma, contrast, bright and sharp level, if user wants to get un-adjusted image data, user might invoke

MTUSB_SetGammaValue() function with Gamma set to 10 (mean 1.0), contrast and bright set to 50 (mean 50%) and SharpLevel set to 0 (mean Normal).

SDK_API MTUSB_GetCurrentFrame(DEV_HANDLE DevHandle, int FrameType, unsigned char *Buffer);

User may call this function to get a frame in real time.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
 FrameType – 0: Raw Data
 1: 24bit Bitmap Data
 Buffer – Byte buffer to hold the whole frame of image.

Return: -1 If the function fails (e.g. invalid device handle or the frame grabbing is NOT running)
 1 if the call succeeds.

Important: This function can only be invoked while the frame grabbing is started, user might want to get a frame of image in memory, instead of in a file. This function will put the current frame into Buffer, either in Raw data format, or in 24bit bitmap format, note that it's caller's responsibility to have big enough buffer to hold the image data, the buffer size should be:

In Raw Data format: At least (Row x Column) Bytes,

In Bitmap Data Format: At least 3 x (Row x Column) Bytes.

Note that this buffer is from current frame flow, so it may be corrected with user's setting of Gamma, contrast, bright and sharp level, if user wants to get un-adjusted image data, user might invoke **MTUSB_SetGammaValue()** function with Gamma set to 10 (mean 1.0), contrast and bright set to 50 (mean 50%) and SharpLevel set to 0 (mean Normal).

SDK_API MTUSB_SaveFramesToFiles(DEV_HANDLE DevHandle, PImageCtl SettingPtr, char *FilePath, char *FileName);

User may call this function to save one or more frames to files.

Argument: DevHandle – the device handle returned by either MTUSB_ShowOpenDeviceDialog() or MTUSB_OpenDevice()
 SettingPtr – the Pointer to variable of TImageControl structure.
 FilePath – the directory to store the saved files.
 FileName – the Filename used for file saving, note that the actual file name will be FileName_x.bmp (or FileName_x.jpg).

Return: -1 If the function fails (e.g. invalid device handle or the frame grabbing is NOT running)
 1 if the call succeeds.

Important: Note that This function can only be invoked while the frame grabbing is started, user might want to get one or more frames and save them into a specified location, the "CatchFrames", "IsAverageFrame", "IsCatchRAW", "IsRawGraph", "IsCatchJPEG" and "CatchIgnoreSkip" in the data structure pointed by SettingPtr will be used for number of frames, frame format (Raw, Bmp or Jpeg) and ignore skip mode options.

Note:

IsAverageFrame gives user an option to get only ONE frame but it's the average of all the captured frames.

IsCatchRAW gives user an option to save the files as CMOS sensor raw data, currently, we generate a Text file for user's ease of observation

IsRawGraph gives user an option to save the graphic file (jpg or bmp) with the data which are NOT corrected with the current Gamma, contrast, bright or sharp algorithm.