

LumaUSB DLL API for the LumaScope 600

Introduction

This help file describes the API (application programming interface) for the 'LumaUSB.DLL' for interfacing with the LumaScope 600. This DLL enables the software developer to create a program to interface with the LumaScope 600 without needing LumaView 600. The LumaUSB DLL is written with C# but the user can interface to the DLL with other language.

Example Application, 'ReadISO.exe', to Help Get Started

The C# project provides a good example how to interface to LumaUSB.DLL. You can examine the code to see how the application interacts with the DLL. A compiled version of 'ReadISO.exe' is included in the development kit, which you can immediately run and see work. Before running 'ReadISO.exe', you must install LumaView 600 as the installation installs the USB drivers, USB is the communication path between the computer and the LumaScope. Ensure that 'ReadISO.exe', 'LumaUSB.dll' and 'Lumascope600.hex' are all in a folder before running 'ReadISO.exe'. 'Lumascope600.hex' is a code-containing file, that gets loaded into the LumaScope at startup of the application if the LumaScope was previously powered off. We recommend making sure that you can run 'ReadISO.exe' successfully before attempting your application development. Also, 'ReadISO.exe' provides a sanity-check when developing, showing that interface to the LumaScope is working.

Initialization of the 'LumaUSB' Object

The C# project provides a good example how to interface to LumaUSB.DLL. You can examine the code to see how the application interacts with the DLL. All the functionality that you will need to interact with the LumaScope is contained in the 'LumaUSB' class in the DLL. In your application, you will need to create an instance of it. Here is how you do it in C#:

```
this.usb = new LumaUSB( LumaUSB.VID_CYPRESS, LumaUSB.PID_LSCOPE,  
videoParams );
```

Note on Initialization

Just after the LumaScope is powered up, it contains no custom Etaluma code for on-board control of the microscope. The custom Etaluma code is contained in "Lumascope600.hex" and gets loaded into the LumaScope when ReadISO.exe starts up. Here is what happens (start ReadISO.exe and then plug in the USB to the LumaScope): ReadISO.exe receives a 'WM_USER_HOT_ARRIVAL' message. ReadISO.exe sees that the USB PID (product ID) is 0x8613 (34323 base 10) and knows, based on the PID, that it needs to load the contents of "Lumascope600.hex" into the LumaScope. After loading the contents of "Lumascope600.hex", the LumaScope simulates a USB reboot and thus getting a 'WM_USER_HOT_REMOVAL' message followed by a 'WM_USER_HOT_ARRIVAL' message. After all of this, the LumaScope gives itself a new PID of 0x1004 (4100 base 10).

Here are the trace statements you would see in the Visual Studio "Output" window during 'ReadISO.exe' startup when the code in 'Lumascope600.hex' has not already been loaded into the LumaScope 600:

```
Selected 23,142,873, index = 7  
WM_USER_HOT_ARRIVAL  
Instantiated 'LumaUSB' with width = 1200, height = 1200.  
DeviceAdded()
```

```

HexFileDownload()
HexFileDownload().....
Lumascope600.hex
HexFileDownload(): 8051 placed in reset.
HexFileDownload(): Released 8051 from reset.
WM_USER_HOT_REMOVAL
WM_USER_HOT_ARRIVAL
Instantiated 'LumaUSB' with width = 1200, height = 1200.
DeviceAdded()

```

Here are the trace statements you would see in the Visual Studio “Output” window during 'ReadISO.exe' startup when the code in 'Lumascope600.hex' has already been loaded into the LumaScope 600:

```

Selected 23,142,873, index = 7
WM_USER_HOT_ARRIVAL
Instantiated 'LumaUSB' with width = 1200, height = 1200.
DeviceAdded()

```

API Listing

Below follows documented API calls. All the calls are members of the 'LumaUSB' class. Note that your application code must use the namespace, “using libusbK”.

LumaUSB.LumaUSB(int vid, int pid, int frameWidthPixels, int frameHeightPixels)

Desc: Constructor, initializes the USB and sets the PID and VID (USB product and vendor IDs).

Param "vid": USB vendor ID.

Param "pid": USB product ID.

Param "frameWidthPixels": Specifies the width of the image, in pixels.

Param "frameHeightPixels": Specifies the height of the image, in pixels.

LumaUSB.LumaUSB(int vid, int pid, VideoParameters vp)

Desc: Constructor, initializes the USB and sets the PID, VID (USB product and vendor IDs) and video parameters.

Param "vid": USB vendor ID.

Param "pid": USB product ID.

Param "vp": Video parameters. NOTE: 'vp.width' and 'vp.height' both must be multiples of four.

bool LumaUSB.GetLatest24bppBuffer(out byte[] buffer)

Desc: Gets image data that was received from the LumaScope. The data is 24 bits per pixel.

Param "buffer": The buffer is filled with image data if there is data available.

Return: True if there was data available and the parameter was filled.

bool LumaUSB.ISOStreamStart()

Desc: The user must call this to start the USB image data streaming.

Param "buffer": The buffer is filled with image data if there is data available.

Return: True is successfully started, else false.

void LumaUSB.ISOStreamStop()

Desc: The counter-call to 'ISOStreamStart()', call this function to stop the image data streaming.

bool LumaUSB.GetNumBytesReceived(ref ulong numBytesReceived)

Desc: Gets the cumulative number of image data bytes received from the LumaScope since starting or calling 'ResetNumBytesReceived()'.
Param "numBytesReceived": This parameter gets set to the number of received bytes.

Return: True if the function is successfully able to get the data byte count.

void LumaUSB.ResetNumBytesReceived()

Desc: Sets the cumulative image data byte count back to zero. Note: get the cumulative byte count by calling 'GetNumBytesReceived()'.

List<string> LumaUSB.GetDeviceDescriptionList()

Desc: Gets a list of the connected LumaScope USB devices.

Return: A textual list connected USB devices.

void LumaUSB.DeviceAdded(KLST_DEVINFO_HANDLE deviceInfo)

Desc: The application using this DLL calls this function when the USB device is connected to the LumaScope.

Param "deviceInfo": Structure containing information about the USB device of interest.

bool LumaUSB.LedControllerWrite(Byte ledID, Byte brightness)

Desc: Sets the brightness level of the specified LED.

Param "ledId": Selects the LED: LED-A = 0x41 = 'A', LED-B = 0x42 = 'B', LED-C = 0x43 = 'C', LED-D = 0x44 = 'D'.

Param "brightness": Specifies the brightness of the LED (range: 0 to 255).

Return: True if successful write.

static string LumaUSB.GetPixelClockDescription(Int32 speed)

Desc: This returns a textual frequency of the image sensor pixel clock corresponding to the index passed as the parameter.

Param "speed": This is the index that specifies the frequency. The range of this parameter is from zero to one less than what 'GetPixelClockDescriptionCount()' returns.

Return: Returns the frequency in text form.

static int LumaUSB.GetPixelClockDescriptionCount()

Desc: Returns the number of the possible frequencies to which we may set the image sensor image sensor pixel clock.

Return: Number of the possible frequencies.

bool LumaUSB.SetImageSensorPixelClockFrequency(Int32 speed)

Desc: Sets the pixel clock frequency on the image sensor chip on the LumaScope, which is the same as selecting the image scanning frequency.

Param "speed": This is the zero-based index corresponding to what is returned from 'GetPixelClockDescription()'. The range of the 'speed' is from zero to one less that what 'GetPixelClockDescriptionCount()' returns.

Return: true if successfully set.

bool LumaUSB.ImageSensorRegisterWrite(UInt16 registerId, UInt16 value)

Desc: Sets an image sensor register to the parameter-specified value. Normally, you will not need to call this function; other functions in the 'LumaUSB' class handle most functions that users need. **Param "registerId":** Specifies the register to set.

Param "value": Contains the value to write to the register.

Return: True if the value was successfully set in the register.

bool LumaUSB.ImageSensorRegisterRead(UInt16 registerId, UInt16 value)

Desc: Reads an image sensor register from the parameter-specified value. Normally, you will not need to call this function; other functions in the 'LumaUSB' class handle most functions that users need. **Param "registerId":** Specifies the register to read.

Param "value": Contains the value to read from the register.

Return: True if the value was successfully set in the register.

bool LumaUSB.SetGlobalGain(UInt16 value)

Desc: This sets the gain of the image sensor. The larger the parameter value, the higher the gain.

Param "value": This can range from 0 to 222 (MAX_GLOBAL_GAIN_PARAMETER_VALUE). The larger the value, the larger the gain.

Return: 'true' if the gain was successfully set, else 'false'.

void LumaUSB.InitImageSensor()

Desc: This initializes the image sensor to reasonable, default settings.

bool LumaUSB.SetWindowSize(int width, int height)

Desc: This sets to the image sensor window to the size, in pixels, as specified by the parameters.

Param "width": Image window width, in pixels.

Param "height": Image window height, in pixels.

Return: 'true' if the image window size successfully set, false if not.

bool LumaUSB.SetWindowSize(int pixelCountSide)

Desc: This sets to the image sensor window to a square size, in pixels, as specified by the parameter.

Param "pixelCountSide": Image window width and height, in pixels.

Return: 'true' if the image window size successfully set, false if not.

bool LumaUSB.StartStreaming()

Desc: Starts the image data streaming on the LumaScope.

Return: True if the LumaScope was successfully commanded.

bool LumaUSB.StopStreaming()

Desc: Stops the image data streaming on the LumaScope.

Return: True if the LumaScope was successfully commanded.

bool LumaUSB.GetFx2FirmwareVersion(out Int16 firmwareVersionWord)

Desc: Reads the FX2 8051 firmware version from the LumaScope.

Param "pixelCountSide": Image window width and height, in pixels.

Return: True if the LumaScope was successfully commanded.

bool LumaUSB.InitializeGPIF()

Desc: Initializes/resets the GPIF LumaScope. It is necessary to reset the GPIF on the FX2 USB chip on the LumaScope after changing the pixel clock.

Return: True if the LumaScope was successfully commanded.

string LumaUSB.HexPath (Property)

Desc: Gets and sets the full file path of the location of the HEX files that gets loaded into the 8051 of the Cypress FX2 USB chip on the LumaScope.

UInt16 LumaUSB.ProductID (Property)

Desc: Gets the USB product ID of the initialized (HEX file already downloaded) LumaScope 600.

UInt16 LumaUSB.VendorID (Property)

Desc: Gets the USB vendor ID of the uninitialized/initialized (same for both) LumaScope 600.

string LumaUSB.ProductName (Property)

Desc: Gets a string containing the name of the product, 'LS600'.

UInt16 LumaUSB.PID_LSCOPE (Constant)

Desc: This is the USB 'Product ID' of the LumaScope 600.

UInt16 LumaUSB.VID_CYPRESS (Constant)

Desc: This is the USB 'Vendor ID' of the Cypress FX2 chip in the LumaScope 600.

UInt16 LumaUSB.PID_FX2_DEV (Constant)

Desc: This is the USB 'Product ID' of the Cypress FX2 chip in the LumaScope 600.

byte LumaUSB.IMAGE_SENSOR_SHUTTER_WIDTH_LOWER (Constant)

Desc: This is a register of the image sensor, 'Shutter Width Lower'.

byte LumaUSB.IMAGE_SENSOR_RESET (Constant)

Desc: This is a register of the image sensor, 'Reset'.

byte LumaUSB.IMAGE_SENSOR_GLOBAL_GAIN (Constant)

Desc: This is a register of the image sensor, 'Global Gain'.

byte LumaUSB.IMAGE_SENSOR_GREEN1_GAIN (Constant)

Desc: This is a register of the image sensor, 'Green Gain1'.

byte LumaUSB.IMAGE_SENSOR_BLUE_GAIN (Constant)

Desc: This is a register of the image sensor, 'Blue Gain'.

byte LumaUSB.IMAGE_SENSOR_RED_GAIN (Constant)

Desc: This is a register of the image sensor, 'Red Gain'.

byte LumaUSB.IMAGE_SENSOR_GREEN2_GAIN (Constant)

Desc: This is a register of the image sensor, 'Green Gain2'.

int LumaUSB.MAX_IMAGE_SENSOR_EXPOSURE (Constant)

Desc: This is the "Shutter Width Lower" register used for exposure. 1943 is the default value.

int LumaUSB.MAX_GLOBAL_GAIN_PARAMETER_VALUE (Constant)

Desc: This specifies the maximum value for the gain parameter in the 'SetGlobalGain()'.

int LumaUSB.RECOMMENDED_MIN_GLOBAL_GAIN_PARAMETER_VALUE (Constant)

Desc: This specifies the minimum recommended value (7) for the gain parameter in the 'SetGlobalGain()'. If the gain goes below this it empirically observed that the image sensor cannot saturate no matter the intensity of the light source.

LumaUSB v14.10.1