



EVALUATION KIT (EVK)

CUSTOM CAMERAS FOR MASS PRODUCTION APPLICATIONS



**APPLICATION PROGRAM
INTERFACE**

©Copyright©Imaging Diagnostics 2009

This manual is copyrighted. All rights are reserved and no part of this publication may be reproduced or transmitted in any form or by any means without prior written consent.

Disclaimer

The information in this manual was accurate and reliable at the time of its release. However, we reserve the right to change the specifications of the product described in this manual without notice at any time.

Registered Trademarks

All other proprietary names mentioned in this manual are the trademarks of their respective owners.

Print Version C-03

December 2009

CONTENTS

| | |
|--|------------|
| CHAPTER 1. PREFACE..... | III |
| About this Document | iii |
| Target Audience..... | iii |
| About this Application | iii |
| Important Information | iii |
| How to Contact Us | iii |
| CHAPTER 2. INTRODUCTION | 1 |
| CHAPTER 3. API COMMAND SUMMARY TABLE | 2 |
| CHAPTER 4. UPGRADING THE TO THE LATEST VERSION | 7 |
| CHAPTER 5. INITIALIZING, ACCESSING AND RUNNING THE CAMERA | 8 |
| FillCameraList | 8 |
| InitCamera | 8 |
| RunCamera..... | 9 |
| ShutDownCamera..... | 9 |
| StopCamera..... | 9 |
| StopDriver..... | 9 |
| CloseCamera..... | 10 |
| OnPause..... | 10 |
| OnRun | 10 |
| SetViewWindow..... | 11 |
| CHAPTER 6. ACCESSING THE CAMERA AND PARAMETERS | 12 |
| GetSensorType..... | 12 |
| GetCameraVersionInfo | 13 |
| GetCamCaps | 13 |
| GetFrameNum..... | 13 |
| GetFPS..... | 14 |
| GetFPSrate | 14 |
| GetFlipHorizontal | 14 |
| SetFlipHorizontal | 14 |
| GetFlipVertical..... | 15 |
| SetFlipVertical | 15 |
| SetRotation..... | 15 |
| GetGreyScale | 15 |
| SetGreyScale | 16 |
| CaptureRawImage | 16 |
| CaptureBMP | 16 |
| Setting and Getting the Gain values..... | 16 |
| GetGainType | 17 |
| SetGainType | 17 |
| Exposure Time | 17 |
| GetShutterDelay | 18 |
| SetShutterDelay | 18 |
| GetShutterWidth | 18 |
| SetShutterWidth..... | 18 |
| Understanding the Camera Resolution Types | 19 |
| GetResolution | 19 |
| SetResolution..... | 19 |
| GetCaptureResolution | 20 |
| SetCaptureResolution..... | 20 |
| Setting ROI (Region of Interest) and FOV (Frame of View) | 20 |

| | |
|--|-----------|
| GetRegion | 20 |
| SetRegion | 21 |
| SetBlankRows | 21 |
| GetView | 21 |
| SetView | 22 |
| Binning | 22 |
| SetBinning | 22 |
| SetCapBinning | 22 |
| Snapshot Mode | 23 |
| EnterSnapShotMode | 23 |
| GetSnapShot | 23 |
| EnterPauseMode | 23 |
| CHAPTER 7. ADVANCED API FUNCTIONS | 24 |
| UpdateFW | 24 |
| UploadLUT | 25 |
| SetTestData | 26 |
| SetGetRawFullData | 27 |
| Using the Callback Function | 28 |
| SetDataCBPtr | 28 |
| SetRawFrameDataCB | 28 |
| SetRawFullFrameCB | 29 |
| SetRGBFrameDataCB | 29 |
| SetPIIRate | 29 |
| SetPwmTimer | 30 |
| GetGPIO | 30 |
| SetGPIO | 31 |
| EnableSensorLight | 31 |
| SetMonochrome | 31 |
| AdjustFlicker | 32 |
| SetExposureTime | 32 |
| GetExposureTime | 32 |
| CheckLeds | 32 |
| SetLedStates | 33 |
| GetLogs | 33 |
| GetRegVal | 33 |
| SetRegVal | 34 |
| GetRegValues | 34 |
| SetRegValues | 34 |
| CHAPTER 8. USING DEBUGPRINT | 35 |
| DebugPrint | 35 |

About this Document

This document is the Camelot Camera Evaluation Kit (EVK) API.

Target Audience

This document is meant for application programmers to integrate the Camelot series cameras into 3rd party applications.

About this Application

The Camelot Sample Application was written to demonstrate to programmers on how to build applications using the **CamelotFilter.ax** or **CamelotDII.dll** functional API. **This API gives the programmer full control over the Camelot camera series.**

The sample applications were compiled using **Microsoft Visual Studio 2005**.

All applications should include the following files:

- **hostCamApi.h** – a file containing definitions used by the camera, the API and the sample applications
- **CamelotDII.h** – a list of API functions in the **CamelotDII.dll** and **CamelotFilter.ax** that can be accessed by your application.

The final RGB images (BMP) from the DirectShow sample (CamelotView_DS) are in 24-bit format, while those using the CamelotDII (CamelotView) are 32-bit.

If you want to use the DirectShow version, you must first compile the Baseclasses (Debug and Release) on your working computer –

"\Program Files\Microsoft SDKs\Windows\v6.1\Samples\Multimedia\DirectShow\BaseClasses". See DirectShow documentation for more details.

Important Information

NOTE

Important information is shown as notes.

How to Contact Us

Website

<http://www.imagine2d.com/>

Support

support@imagine2d.com

Sales

sales@imagine2d.com

INTRODUCTION

Camelot is a family of digital cameras for machine vision applications. Using a fast USB2 connection and an embedded digital signal processor, Camelot cameras are capable of performing advanced image processing algorithms in the camera and buffering images internally; this decreases the camera/host bandwidth requirement significantly. The cameras are intended for medical and industrial applications requiring superior image quality and high performance.

IMPORTANT NOTE

You can download the latest version of the software from our website. See [Upgrading the to the Latest Version](#) on page 7.

Downloads Contain:

- **CamelotFilter.ax** – a DirectShow filter containing functions that access and communicate with the camera - this file must be placed in the **X:\Windows\System32** folder. Where: **X:** is your system disk drive.
- **CamelotView_DS.exe** – the **SampleApplication** executable based on DirectShow which accesses functions in **CamelotFilter.ax**
- **CamelotView.zip** – contains the latest updated sources of the Sample applications.
- **CamelotDll.dll** – a library of functions that access and run the camera without using DirectShow.
- **CamelotView.exe** – executable that uses the **CamelotDll.dll** and when running must be in the same directory.
- **Camelot_XXX.ldr** – is the loader file with the new code for the camera. This file must be loaded using the example application [UpdateFW](#) or by calling the API function and supplying the path to the file.

API COMMAND SUMMARY TABLE

| Command | Description |
|---|--|
| <u>AdjustFlicker</u> | Adjusts the Shutter Width in order to avoid the 50Hz or 60Hz flicker experienced when using indoor lighting. |
| <u>CaptureBMP</u> | Captures the next frame from the camera and saves it as a BMP image. DirectShow images are saved in RGB24 format. CamelotDll images are saved in RGB32 format. |
| <u>CaptureRawImage</u> | Captures the next frame from the camera and saves it as a RAW, GBRG Bayer image. |
| <u>CheckLeds</u> | Checks whether the three external LEDs (optionally provided with the camera) are functioning. To utilize this function a separate control board is required . |
| <u>CloseCamera</u> | Closes all connections to the selected camera and releases all buffers used for video capture. |
| <u>DebugPrint</u> | Prints a message to a debug window (if open) while in DEBUG mode. |
| <u>EnableSensorLight</u> | Turns on LEDs on the Sensor board. These can be optionally supplied with the camera. If Flash option is enabled when Snapshot is chosen, the LEDs light whenever a snapshot is taken. |
| <u>EnterPauseMode</u> | Enters or leaves pause mode. When in PauseMode the camera stops capturing and streaming video. All control messages are still sent. This message must be sent to the camera to notify it that video streaming is paused. |
| <u>EnterSnapShotMode</u> | Enters or leaves Snapshot mode. When in SnapshotMode the camera stops capturing and streaming video until a trigger is set. |
| <u>FillCameraList</u> | Searches for USB based Camelot cameras connected on your PC. DirectShow - finds Video Capture devices in registry that are called ID Camelot. CamelotDll - accesses the ID Camelot USB devices using an exported function. |
| <u>GetCamCaps</u> | Returns a structure containing some of the camera's capabilities. When this function is called, the camera is initialized and connection is established with the PC. |
| <u>GetCameraVersionInfo</u> | Gets the camera's version numbers – HW, FW and others. |
| <u>GetCaptureResolution</u> | Returns the current resolution of the camera's output. |
| <u>GetExposureTime</u> | Gets the actual exposure time (in milliseconds) for each frame according to the ShutterWidth and ShutterDelay and capture resolution. |
| <u>GetFlipHorizontal</u> | Returns whether the picture is flipped horizontally. |
| <u>GetFlipVertical</u> | Returns whether the picture is flipped vertically. |

| Command | Description |
|--|--|
| <u>GetFPS</u> | Outputs the FPS (frames per second) of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning. |
| <u>GetFPSrate</u> | Outputs the Bit Rate of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning. |
| <u>GetFrameNum</u> | Outputs the current frame number being captured from the camera. This is a sequential number that starts at 0 (zero) every time the camera is reinitialized. |
| <u>GetGainType</u> | Returns the sensor gain according to the type queried. See <u>Setting and Getting the Gain values</u> on page 16 for explanation of values. |
| <u>GetGPIO</u> | Gets the current state and values of the 4 GPIO (general purpose input/output) pins that are reserved for your use. These pins can either be an input or an output, which means they either read a value from the camera, or set a value into the camera. Cameras can be optionally wired so that these GPIO pins can turn on/off a circuit or transmit info from the camera. To utilize this function a separate control board is required. |
| <u>GetGreyScale</u> | Returns whether the picture is shown in greyscale. |
| <u>GetLogs</u> | Gets log messages saved in the camera for debugging purposes. These messages are sent to a separate <u>DebugPrint</u> screen (if opened) and can be used to debug applications. See Using DebugPrint <u>on page 35</u> . |
| <u>GetRegion</u> | Gets the coordinates of the region (ROI – Region of Interest) programmed in the registers of the camera. |
| <u>GetRegVal</u> | Returns value of specified register. |
| <u>GetRegValues</u> | Outputs all defined register values to a REG_LIST_API structure as defined in hostCamApi.h . |
| <u>GetResolution</u> | Returns the current resolution of the Preview screen. |
| <u>GetSensorType</u> | Returns the type of Micron sensor present in the camera. The type may be 1.3, 3, 5 Megapixel or WVGA sensor according to Table 1 below. |
| <u>GetShutterDelay</u> | Returns the sensor shutter delay. This value is used for controlling Exposure time and to avoid any 50Hz or 60Hz flicker. |
| <u>GetShutterWidth</u> | Sets the sensor shutter width. This value controls the time taken between each new frame capture. If in <u>SetMonochrome</u> mode the value supplied is adjusted to the next highest value to avoid any 50/60Hz flicker. |
| <u>GetSnapShot</u> | Imitates the optional external trigger and instructs the camera to capture one frame and send it to the PC. |
| <u>GetView</u> | Gets the coordinates of the FOV (Frame of View) displayed on the PC. [This option is currently not supported] |

| Command | Description |
|---|---|
| <u>InitCamera</u> | <p>DirectShow - initializes the DirectShow filters needed to run the SampleApplication. Within the function, once the pFilter is enabled, the camera is initialized and ready for commands.</p> <p>CamelotDll - opens the camera and initializes it for use using AccessCamera(), an exported function from the dll.</p> <p>Commands are sent to the camera using the piBDR interface.</p> <p>All of the commands in this API are invoked using the following syntax: piBDR[camNum] ->Command(parameters)</p> <p>The application enables initializing up to 10 cameras. Each camera has a number according to its location in the List Control displayed on the Main screen.</p> |
| <u>OnPause</u> | <p>Pauses the selected camera that is running. OnRun() restarts it.</p> <p>DirectShow - uses the MediaControl filter.</p> <p>CamelotDll - calls the API function - Pause().</p> |
| <u>OnRun</u> | <p>Starts the selected camera that has been either paused or stopped. If the camera wasn't initialized, it is initialized and then started.</p> <p>DirectShow: uses the MediaControl filter.</p> <p>CamelotDll - calls <code>SetViewWindow(HWND hWnd, int a_camNum)</code> API function in order to provide HANDLE to preview window and then calls API function, Run().</p> |
| <u>RunCamera</u> | <p>Starts a camera that has already been initialized. When called, video is streamed from the camera and displayed in a separate Preview window.</p> |
| <u>SetBinning</u> | <p>Sets whether binning should be used for preview (if HALF or QUARTER resolution is selected).</p> <p>[This option is currently not supported]</p> |
| <u>SetBlankRows</u> | <p>For a simple, custom ROI - 2 horizontal bars are output from the camera. One near the top of the screen and one near the bottom. The width of the bars is determined by the amount of blank rows between the two upper and lower video bars.</p> |
| <u>SetCapBinning</u> | <p>Sets whether binning should be used by the sensor for video output (if HALF or QUARTER resolutions is selected).</p> |
| <u>SetCaptureResolution</u> | <p>Sets the resolution used for the camera's output.</p> |
| <u>SetDataCBPtr</u> | <p>Sets the callback function that is called every time a new frame is received, if specified.</p> |
| <u>SetExposureTime</u> | <p>Sets the exposure time (in milliseconds) for each frame. If the value given is lower than the camera can operate at, the camera operates at its minimum exposure time.</p> |
| <u>SetFlipHorizontal</u> | <p>Sets whether the picture should be flipped horizontally.</p> |
| <u>SetFlipVertical</u> | <p>Sets whether the picture should be flipped vertically.</p> |
| <u>SetGainType</u> | <p>Sets sensor's gain according to the type specified. See <u>Setting and Getting the Gain values</u> on page 16.</p> |
| <u>SetGetRawFullData</u> | <p>Sets mode for getting RAW frames. For 5Mp camera only: If a 12-bit RAW frame is required, puts the camera into 48MHz mode (using the <u>SetPIIRate</u> function) before requesting the RAW frame.</p> |

| Command | Description |
|--|---|
| <u>SetGPIO</u> | Sets the current state and values of the 4 GPIO (general purpose input/output) pins that are reserved for the user. These pins can either be an input or an output – which means they either read a value from the camera, or set a value. Cameras can be optionally wired so that these GPIO pins can turn on/off a circuit or transmit info from the camera. To utilize this function a separate control board is required. |
| <u>SetGreyScale</u> | Sets whether the picture should be shown using greyscale. |
| <u>SetLedStates</u> | Sets how many different LED states have been programmed into the camera. Changing between states is done by triggering a GPIO. The optional LEDs can be controlled differently in each LED state. To utilize this function a separate control board is required. |
| <u>SetMonochrome</u> | Sets whether sensor outputs monochrome (greyscale) data. This information is used by the Display thread when converting the image from 8-bit Bayer format to RGB (24-bit or 32-bit). |
| <u>SetPIIRate</u> | Sets the camera's PLL (internal clock) rate. Default: 96MHz. This option is not supported by the 1.3Mp and the 3Mp models. Before uploading and using a LUT with 12-bit streaming, the 5Mp camera must be set to 48MHz. Also, if a RAW frame is to be captured with 12-bit data, change the camera PLL rate to 48MHz before frame capture. |
| <u>SetPwmTimer</u> | Sets three PWM (pulse width modulation) timers which you can attach to peripherals (LEDs, etc.) that need PWM controlled current source. The period is the overall cycle, while the width is the duration of the period (duty cycle) where the signal is low. The width must be less than the period for each timer. The timers (TMR1, TMR2 and TMR3) are started simultaneously and therefore synchronized. To utilize this function a separate control board is required. |
| <u>SetRawFrameDataCB</u> | Sets whether a callback function should be called for each raw frame received. |
| <u>SetRawFullFrameCB</u> | Sets whether a callback function should be called for each raw Full frame received (as a result of the <u>SetGetRawFullData</u> function on page 27). |
| <u>SetRegion</u> | Sets the region (ROI - Region of Interest) to be output by the camera. |
| <u>SetRegVal</u> | Sets specified register value. |
| <u>SetRegValues</u> | Sets register values to list in file. [This option is not currently supported] |
| <u>SetResolution</u> | Sets the resolution used for the Preview screen. The FULL parameter (0) gives the maximum sensor output from the camera. All other resolutions are subsets of this FULL output. |
| <u>SetRGBFrameDataCB</u> | Sets whether a callback function should be called after each raw frame has been processed into an RGB frame. |
| <u>SetRotation</u> | Sets the angle in degrees (0, 90, 180 and 270) for rotating the image. |

| Command | Description |
|--|---|
| <u>SetShutterDelay</u> | Sets the sensor shutter delay This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker. |
| <u>SetShutterWidth</u> | Sets the sensor shutter width This value controls the sensor delay time between each new frame capture. |
| <u>SetTestData</u> | Specifies whether the camera should output set Test Data or regular video images. This is an option provided for developers to check whether the data received is actually the data sent and other sensor independent features. 1.3Mp and 3Mp cameras have only 1 option – to choose a value which is output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED. For 5Mp cameras this is type=4. The 5Mp camera has other optionsfor creating patterns. |
| <u>SetView</u> | Sets the FOV (Field of View) to be displayed on the PC. [This option is currently not supported] |
| <u>SetViewWindow</u> | Only with CamelotDll – to provide the dll with a HANDLE to the preview window which is created by the CamelotView sample. This window is used by the DisplayThread for viewing the video. |
| <u>ShutDownCamera</u> | DirectShow - stops the camera and tears down the DirectShow filter graph. CamelotDll - stops the camera and frees all memory associated with the camera using FreeCamera() an exported dll function. Calls API functions <u>StopDriver</u> and <u>CloseCamera</u> |
| <u>StopCamera</u> | Stops a running camera but keeps it initialized. Calls API function <u>StopDriver</u> . CamelotDll - uses StopCameraOutput an exported function in the dll. |
| <u>StopDriver</u> | Stops the selected camera’s driver. Camera stops outputting video and enters a waiting for command mode. |
| <u>UpdateFW</u> | Updates the camera Firmware. |
| <u>UploadLUT</u> | Uploads a look-up table (LUT) for translating raw pixel values in the camera before being outputted. The LUT can translate from 8-bit, 10-bit or 12-bit data to 8-bit, 10-bit or 12-bit output. In this version only translations to 8-bit data are supported, which is transmitted in the output. Once a LUT is loaded, the camera uses it to translate all outgoing frames. After the LUT is successfully loaded, the SampleApplication must reinitialize the camera and run it. |

UPGRADING THE TO THE LATEST VERSION

After you have installed the software from the Installation CD you must visit our website <http://www.imagine2d.com/> and download and install the latest version of both the software and camera firmware currently available.

◆ **To install the latest version of the camera software and firmware:**

1. Download the latest version of the software from <http://www.imagine2d.com/>
2. Click the **Support** tab.
3. From the menu, click **Downloads**.
4. Download the latest version of the Camelot software to a temporary location.
5. Upload the new firmware (**Camelot_XXX.Idr**) to the camera using the current/old working **SampleApplication** (**NOT** the file you just downloaded).
6. When the firmware upload is successfully completed, unplug the USB cable and close the **CamelotSample** application
7. Copy the **Camelotfilter.ax** file into the folder **C:\Windows\system32**
Where: **C:** is the system disk.

Reconnect the camera and copy and then run **NEW SampleApplication** file.

INITIALIZING, ACCESSING AND RUNNING THE CAMERA

This section describes the commands for initializing and running the camera.

All the commands are provided in the Sample applications. There are two sets of commands, one for the **DirectShow** functions and the other uses the functions in the **CamelotDll.dll**. Only when you actually look at the actual code are you able to fully understand the usage and syntax of each command.

Main Dialog module function calls – for Camelot_SampleCode.cpp.

These functions are used to access, initialize, run, pause, stop and close the camera.

FillCameraList

| | | | |
|----------------------|--|--------------------------|----------------------------|
| Description | Searches for USB based Camelot cameras connected on your PC. DirectShow - finds Video Capture devices in registry that are called ID Camelot. CamelotDll - accesses the ID Camelot USB devices using an exported function. | | |
| Syntax | <code>InitCamera (int a_nCamNum, bool *CameraFound)</code> | | |
| Parameters | | | |
| | Input | <code>a_nCamNum</code> | Camera number |
| | Output | <code>CameraFound</code> | Whether a camera was found |
| Return Values | None | | Success |

InitCamera

| | | | |
|----------------------|--|--------------------------|----------------------------|
| Description | DirectShow - initializes the DirectShow filters needed to run the SampleApplication . Within the function, once the pFilter is enabled, the camera is initialized and ready for commands. CamelotDll - opens the camera and initializes it for use using AccessCamera() , an exported function from the dll. Commands are sent to the camera using the piBDR interface. All of the commands in this API are invoked using the following syntax: piBDR[camNum] ->Command(parameters) The application enables initializing up to 10 cameras. Each camera has a number according to its location in the List Control displayed on the Main screen. | | |
| Syntax | <code>InitCamera (int a_nCamNum, bool *CameraFound)</code> | | |
| Parameters | | | |
| | Input | <code>a_nCamNum</code> | Camera number |
| | Output | <code>CameraFound</code> | Whether a camera was found |
| Return Values | None | | Success |

RunCamera

| | | | |
|----------------------|--|------------------|---------------|
| Description | Starts a camera that has already been initialized. When called, video is streamed from the camera and displayed in a separate Preview window. | | |
| Syntax | RunCamera (int a_nCamNum) | | |
| Parameters | | | |
| | Input | a_nCamNum | Camera number |
| | Output | None | |
| Return Values | true | succeeded | |
| | false | Failed | |

ShutDownCamera

| | | | |
|----------------------|--|------------------|---------------|
| Description | DirectShow - stops the camera and tears down the DirectShow filter graph. CamelotDll - stops the camera and frees all memory associated with the camera using FreeCamera() an exported dll function. Calls API functions StopDriver and CloseCamera . | | |
| Syntax | ShutDownCamera (int a_nCamNum) | | |
| Parameters | | | |
| | Input | a_nCamNum | Camera number |
| | Output | None | |
| Return Values | None | | |

StopCamera

| | | | |
|----------------------|---|------------------|---------------|
| Description | Stops a running camera but keeps it initialized. Calls API function StopDriver . CamelotDll - uses StopCameraOutput an exported function in the dll. | | |
| Syntax | StopCamera (int a_nCamNum) | | |
| Parameters | | | |
| | Input | a_nCamNum | Camera number |
| | Output | None | |
| Return Values | None | | |

StopDriver

| | | | |
|--------------------|---|-------------|--|
| Description | Stops the selected camera's driver. Camera stops outputting video and enters a waiting for command mode. | | |
| Syntax | StopDriver () | | |
| Parameters | | | |
| | Input | None | |
| | Output | None | |

CloseCamera

| | | | |
|--------------------|--|-------------|--|
| Description | Closes all connections to the selected camera and releases all buffers used for video capture. | | |
| Syntax | CloseCamera () | | |
| Parameters | | | |
| | Input | None | |
| | Output | None | |

OnPause

| | | | |
|----------------------|---|-------------|--|
| Description | Pauses the selected camera that is running. OnRun() restarts it. DirectShow - uses the MediaControl filter. CamelotDll - calls the API function - Pause() . | | |
| Syntax | OnPause () | | |
| Parameters | | | |
| | Input | None | |
| | Output | None | |
| Return Values | None | | |

OnRun

| | | | |
|----------------------|---|-------------|--|
| Description | Starts the selected camera that has been either paused or stopped. If the camera wasn't initialized, it is initialized and then started. DirectShow: uses the MediaControl filter. CamelotDll - calls SetViewWindow(HWND hWnd, int a_camNum) API function in order to provide HANDLE to preview window and then calls API function, Run() . | | |
| Syntax | OnRun () | | |
| Parameters | | | |
| | Input | None | |
| | Output | None | |
| Return Values | None | | |

SetViewWindow

| | | | |
|----------------------|--|-----------------|---|
| Description | Only with CamelotDll – to provide the dll with a HANDLE to the preview window which is created by the CamelotView sample. This window is used by the DisplayThread for viewing the video. | | |
| Syntax | <code>SetViewWindow(HWND hWnd, int a_camNum)</code> | | |
| Parameters | | | |
| | Input | hWnd | HWND handle to CFrameWnd's m_hWnd – which is used for preview |
| | | a_camNum | Camera number |
| | Output | None | |
| Return Values | None | | |

Accessing the Camera and Parameters

All of the commands in this API are invoked with the following syntax
`piBDR[camNum] ->Command(parameters)`

Most of these function calls are used in the Sample Application.

GetSensorType

| | | | |
|--------------------|---|--|---|
| Description | Returns the type of Micron sensor present in the camera. The type may be 1.3, 3, 5 Megapixel or WVGA sensor according to Table 1 below. | | |
| Syntax | <code>GetSensorType (int *pnSensorType)</code> | | |
| Parameters | | | |
| Input | None | | |
| Output | pnSensorType | SENSOR_UNKNOWN 0 SENSOR_WVGA 3 SENSOR_1300 13 SENSOR_3000 30 SENSOR_5000 50 | WVGA 1.3 Megapixel 3 Megapixel 5 Megapixel |

Table 1: Types of Cameras and their Resolutions

| Resolution | Width | Height | Total Raw Pixels | Pixels for RGB24 | Y Row Start | X Column Start |
|---------------|-------|--------|------------------|------------------|-------------|----------------|
| 1.3 Mp | | | | | 12 | 20 |
| Full | 1280 | 1024 | 1,310,720 | 3,932,160 | | |
| Half | 640 | 512 | 327,680 | 983,040 | | |
| Quarter | 320 | 256 | 81,920 | 245,760 | | |
| 3 Mp | | | | | 20 | 32 |
| Full | 2048 | 1536 | 3,145,728 | 9,437,184 | | |
| Half | 1024 | 768 | 786,432 | 2,359,296 | | |
| Quarter | 512 | 384 | 196,608 | 589,824 | | |
| 5 Mp | | | | | 54 | 16 |
| Full | 2592 | 1944 | 5,038,848 | 15,116,544 | | |
| Half | 1296 | 972 | 1,259,712 | 377,9136 | | |
| Quarter | 648 | 486 | 314,928 | 944,784 | | |
| WVGA | | | | | 4 | 1 |
| Full | 752 | 480 | 360,960 | 1,082,880 | | |
| Half | 376 | 240 | 90,240 | 270,720 | | |
| Quarter | 188 | 120 | 22,560 | 67,680 | | |

GetCameraVersionInfo

| | | | |
|----------------------|--|--------------------|---|
| Description | Gets the camera's version numbers – HW, FW and others. | | |
| Syntax | <code>GetCameraVersionInfo(char *versionInfo)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | versionInfo | A string of up to 495 characters describing camera HW version, FW version and dates as well as other details. |
| Return Values | None | | |

GetCamCaps

| | | | |
|--------------------|--|------------------|---|
| Description | Returns a structure containing some of the camera's capabilities. When this function is called, the camera is initialized and connection is established with the PC. | | |
| Syntax | Gets the camera's version numbers – HW, FW and others. | | |
| Parameters | | | |
| | Input | None | |
| | Output | CameraCap | <p>CAMERA_CAP_API struct which is defined as:</p> <pre> unsigned int SensorType; // see GetSensorType char Desc[SENSOR_DESC_LEN]; float FirmwareVersion; // float HardwareVersion; // unsigned int Width; // maximum width unsigned int Height; // maximum height unsigned int ActiveStartX; // x column start unsigned int ActiveStartY; // y row start char CameraID[CAM_ID_LEN]; // serial # unsigned char *pCam; // pointer to camera </pre> |

GetFrameNum

| | | | |
|--------------------|--|---------------------|--|
| Description | Outputs the current frame number being captured from the camera. This is a sequential number that starts at 0 (zero) every time the camera is reinitialized. | | |
| Syntax | <code>unsigned long GetFrameNum(void)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | Frame number | Current frame number captured. Frame numbers are given to each frame captured by the camera. |

GetFPS

| | | | |
|--------------------|---|-------------------|---|
| Description | Outputs the FPS (frames per second) of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning. | | |
| Syntax | <code>GetFPS(float *pFPS, float *pSkipPS, float *pDisplayPS)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pFPS | How many FPS captured by DirectShow filter |
| | | pSkipPS | How many FPS were skipped by filter (couldn't be buffered) Total FPS output of camera = pSkipPS + pFPS |
| | | pDisplayPS | How many FPS displayed on PC Preview screen |

GetFPSrate

| | | | |
|--------------------|--|-------------------|--|
| Description | Outputs the Bit Rate of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning. | | |
| Syntax | <code>GetFPSrate(float *pFPS, float *pDisplayPS)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pFPS | How many MegaBits/sec captured by DirectShow filter |
| | | pDisplayPS | How many MegaBits/sec displayed on PC Preview screen |

GetFlipHorizontal

| | | | |
|--------------------|--|-------------------------|--|
| Description | Returns whether the picture is flipped horizontally. | | |
| Syntax | <code>GetFlipHorizontal(bool *pbFlipHorizontal)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pbFlipHorizontal | true = picture is flipped horizontally false = picture isn't flipped horizontally |

SetFlipHorizontal

| | | | |
|--------------------|--|------------------------|---|
| Description | Sets whether the picture should be flipped horizontally. | | |
| Syntax | <code>SetFlipHorizontal(bool bFlipHorizontal)</code> | | |
| Parameters | | | |
| | Input | bFlipHorizontal | true = flip the picture horizontally false = don't flip the picture horizontally |
| | Output | None | |

GetFlipVertical

| | | | |
|--------------------|---|-----------------------|--|
| Description | Returns whether the picture is flipped vertically. | | |
| Syntax | <code>GetFlipVertical (bool *pbFlipVertical)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pbFlipVertical | true = picture is flipped vertically false = picture isn't flipped vertically |

SetFlipVertical

| | | | |
|--------------------|--|----------------------|---|
| Description | Sets whether the picture should be flipped vertically. | | |
| Syntax | <code>SetFlipVertical (bool bFlipVertical)</code> | | |
| Parameters | | | |
| | Input | bFlipVertical | true = flip the picture vertically false = don't flip the picture vertically |
| | Output | None | |

SetRotation

| | | | |
|----------------------|--|------------------|--|
| Description | Sets the angle in degrees (0, 90, 180 and 270) for rotating the image. | | |
| Syntax | <code>SetRotation (int nRotation)</code> | | |
| Parameters | | | |
| | Input | nRotation | DON'T ROTATE = 0, 90 //currently unsupported 180, 270 //currently unsupported |
| | Output | None | |
| Return Values | None | | |

GetGreyScale

| | | | |
|--------------------|--|--------------------|---|
| Description | Returns whether the picture is shown in greyscale. | | |
| Syntax | <code>GetGreyScale (bool *pbGreyScale)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pbGreyScale | true = picture is in greyscale false = picture is shown in color |

SetGreyScale

| | | | |
|--------------------|---|--------------|-----------------------------|
| Description | Sets whether the picture should be shown using greyscale. | | |
| Syntax | SetGreyScale (bool bGreyScale) | | |
| Parameters | | | |
| Input | bGreyScale | true | = show picture in greyscale |
| | | false | = show picture in color |
| Output | None | | |

CaptureRawImage

| | | | |
|--------------------|--|--|--|
| Description | Captures the next frame from the camera and saves it as a RAW, GBRG Bayer image. | | |
| Syntax | CaptureRawImage (char* a_sFileName) | | |
| Parameters | | | |
| Input | a_sFileName | Valid file name – does not have to exist. If exists, is overwritten. | |
| Output | None | | |

CaptureBMP

| | | | |
|--------------------|--|--|--|
| Description | Captures the next frame from the camera and saves it as a BMP image. DirectShow images are saved in RGB24 format. CamelotDII images are saved in RGB32 format. | | |
| Syntax | CaptureBMP (char* a_sFileName) | | |
| Parameters | | | |
| Input | a_sFileName | Valid file name – does not have to exist. If exists, is overwritten. | |
| Output | None | | |

Setting and Getting the Gain values

The gain values have been normalized to the range 0-1024.

All values from 0-64 relate to the analog gain, where:

- 0 - 32 are "real" gains of 0.0 – 4.0
- 33 - 64 are "real" gains of 4.25 – 8.00

Values from 65 – 1024 translate as follows;

- in 1.3Mp cameras – 65-120 are analog gains of 9-15 (values over 120 are ignored)

For other cameras, 65 – 1024 are digital gains.

GetGainType

| | | | |
|--------------------|--|---------------------------------|------------------|
| Description | Returns the sensor gain according to the type queried. See Setting and Getting the Gain values on page 16 for explanation of values. | | |
| Syntax | <code>GetGainType(int type, int *pnGain)</code> | | |
| Parameters | | | |
| Input | Type | RED | 0 |
| | | GREEN | 1 |
| | | BLUE | 2 |
| | | GLOBAL | 3 // not defined |
| Output | pnGain | Returned gain in range 0 - 1024 | |

SetGainType

| | | | |
|--------------------|---|--|-----------------------------|
| Description | Sets sensor's gain according to the type specified. See Setting and Getting the Gain values on page 16. | | |
| Syntax | <code>SetGainType(int type, int nGain)</code> | | |
| Parameters | | | |
| Input | Type | RED | 0 |
| | | GREEN | 1 |
| | | BLUE | 2 |
| | | GLOBAL | 3 // sets R, G, and B gains |
| | nGain | Value of gain – range 0 – 1024. See above for short explanation of analog/digital gains. | |
| Output | None | | |

Exposure Time

The Exposure time is the reset time of each pixel row subtracted from the sample time. It is the amount of time required until a new row is available. Exposure time is a function of the camera **PIXCLK** (pixel clock), shutter width, shutter delay, frame width and binning. Exposure time is calculated per row, and is displayed on the **Main** screen. Except for the WVGA sensor (described below) all the Camelot series cameras use a rolling shutter. Rolling shutters cannot freeze moving objects as well as a global shutter can.

For more information refer to the datasheets of each specific sensor.

WVGA (Wide VGA Sensor - 752 x 480 pixels)

The global shutter feature of the WVGA image sensor is able to freeze moving objects as all pixels are exposed simultaneously. When using a global shutter all pixels start being exposed (integrating charge) simultaneously and stop being exposed simultaneously and a new exposure only begins after the readout of all of the pixels is completed.

GetShutterDelay

| | | | |
|--------------------|---|-----------------------|---|
| Description | Returns the sensor shutter delay. This value is used for controlling Exposure time and to avoid any 50Hz or 60Hz flicker. | | |
| Syntax | <code>GetShutterDelay (int *pnShutterDelay)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pnShutterDelay | Returned shutter delay – values are in range from 0 – 2,047 |

SetShutterDelay

| | | | |
|--------------------|---|----------------------|-----------------------------|
| Description | Sets the sensor shutter delay This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker. | | |
| Syntax | <code>SetShutterDelay (int nShutterDelay)</code> | | |
| Parameters | | | |
| | Input | nShutterDelay | Valid values from 0 – 2,047 |
| | Output | None | |

GetShutterWidth

| | | | |
|--------------------|---|-----------------------|--|
| Description | Sets the sensor shutter width. This value controls the time taken between each new frame capture. If in SetMonochrome mode the value supplied is adjusted to the next highest value to avoid any 50/60Hz flicker. | | |
| Syntax | <code>GetShutterWidth (int *pnShutterWidth)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pnShutterWidth | Returned shutter width – values are in range from 1 – 10,000 |

SetShutterWidth

| | | | |
|--------------------|---|----------------------|----------------------------|
| Description | Sets the sensor shutter width This value controls the sensor delay time between each new frame capture. | | |
| Syntax | <code>SetShutterWidth (int nShutterWidth)</code> | | |
| Parameters | | | |
| | Input | nShutterWidth | Valid values from 1-10,000 |
| | Output | None | |

Understanding the Camera Resolution Types

In this application we relate to two resolution types:

1. Resolution of the camera output or capture resolution.
 - a. **FULL** - the camera outputs frames at maximum width and height.
 - b. **HALF** - the camera outputs frames at HALF width and HALF height of the FULL sized frame. That means that HALF frames are really 1/4 the size (in bytes) of FULL frames.
 - c. **QUARTER** - the camera is outputting frames at 1/16 the size of a FULL frame, since every three rows and every 3 columns are skipped.
2. Resolution of the **Preview** screen on the PC.
 - a. **FULL** - what is received from the camera is displayed at the same dimensions.
 - b. **HALF** - in preview means that only 1/4 of the pixels coming from the camera are displayed, since every alternate row and column is skipped.
 - c. **QUARTER** - only 1/16 of the pixels received from the camera are displayed.

GetResolution

| | | | |
|--------------------|---|---------------------|--|
| Description | Returns the current resolution of the Preview screen. | | |
| Syntax | <code>GetResolution(int *pnResolution)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pnResolution | <pre> FULL = 0, HALF = 1, QUARTER = 2, EIGHTH = 3, // not yet supported </pre> |

SetResolution

| | | | |
|--------------------|---|--------------------|--|
| Description | Sets the resolution used for the Preview screen. The FULL parameter (0) gives the maximum sensor output from the camera. All other resolutions are subsets of this FULL output. | | |
| Syntax | <code>SetResolution(int nResolution)</code> | | |
| Parameters | | | |
| | Input | nResolution | <pre> FULL = 0, HALF = 1, QUARTER = 2, EIGHTH = 3, // not yet supported </pre> |
| | Output | None | |

GetCaptureResolution

| | | | |
|--------------------|--|---------------------|---|
| Description | Returns the current resolution of the camera's output. | | |
| Syntax | <code>GetCaptureResolution(int *pnResolution)</code> | | |
| Parameters | | | |
| | Input | None | |
| | Output | pnResolution | <pre> FULL = 0, HALF = 1, QUARTER = 2, EIGHTH = 3, // not yet supported.</pre> |

SetCaptureResolution

| | | | |
|--------------------|--|--------------------|---|
| Description | Sets the resolution used for the camera's output. | | |
| Syntax | <code>SetCaptureResolution(int nResolution)</code> | | |
| Parameters | | | |
| | Input | nResolution | <pre> FULL = 0, HALF = 1, QUARTER = 2, EIGHTH = 3, // not yet supported.</pre> |
| | Output | None | |

Setting ROI (Region of Interest) and FOV (Frame of View)

You can request a specific area of the frame as an output from the camera. The picture remains in the same resolution, but a smaller subset of the entire frame is sent to the PC. The SetRegion and GetRegion commands control the ROI requested from the camera. From the captured frames sent to the PC, you may only want to display a partial section – SetView and GetView control the specific FOV to be displayed on the PC.

GetRegion

| | | | |
|--------------------|--|----------------|---|
| Description | Gets the coordinates of the region (ROI – Region of Interest) programmed in the registers of the camera. | | |
| Syntax | <code>GetRegion(int *startX, int *startY, int *width, int *height)</code> | | |
| Parameters | | | |
| | Input | *startX | X coordinate value (column number) for region's left side |
| | | *startY | Y coordinate (row number) for region's top row |
| | | *width | Width in pixels from startX |
| | | *height | Height in pixels from startY |
| | Output | None | |

SetRegion

| | | | |
|--------------------|--|--|---|
| Description | Sets the region (ROI - Region of Interest) to be output by the camera. | | |
| Syntax | <code>SetRegion(int startX, int startY, int width, int height)</code> | | |
| Parameters | | | |
| Input | startX | | X coordinate value (column number) for region's left side |
| | startY | | Y coordinate (row number) for region's top row |
| | width | | Width in pixels from startX |
| | height | | Height in pixels from startY |
| Output | None | | |

SetBlankRows

| | | | |
|--------------------|--|--|---|
| Description | For a simple, custom ROI – 2 horizontal bars are output from the camera. One near the top of the screen and one near the bottom. The width of the bars is determined by the amount of blank rows between the two upper and lower video bars. | | |
| Syntax | <code>SetBlankRows(int numRows)</code> | | |
| Parameters | | | |
| Input | numRows | | How many blank rows should be output between the top bar and bottom bar of video. |
| Output | None | | |

GetView

| | | | |
|--------------------|---|--|--|
| Description | Gets the coordinates of the FOV (Frame of View) displayed on the PC. [This option is currently not supported] | | |
| Syntax | <code>GetView(int *startX, int *startY, int *width, int *height)</code> | | |
| Parameters | | | |
| Input | *startX | | Pointer to X coordinate value (column number) for view's left side |
| | *startY | | Pointer to Y coordinate (row number) for view's top row |
| | *width | | Pointer to width in pixels from startX |
| | *height | | Pointer to height in pixels from startY |
| Output | None | | |

SetView

| | | | |
|--------------------|--|---------------|---|
| Description | Sets the FOV (Field of View) to be displayed on the PC. [This option is currently not supported] | | |
| Syntax | <code>SetView(int startX, int startY, int width, int height)</code> | | |
| Parameters | | | |
| | Input | startX | X coordinate value (column number) for view's left side |
| | | startY | Y coordinate (row number) for view's top row |
| | | width | Width in pixels from startX |
| | | height | Height in pixels from startY |
| | Output | None | |

Binning

Binning takes place when rows and columns are skipped (in HALF and QUARTER resolutions). The rows/columns displayed are average with the skipped rows/columns. The picture should be a bit smoother but the FPS is similar to that of a FULL a frame (which means a lower FPS value).

SetBinning

| | | | |
|--------------------|---|-----------------|---|
| Description | Sets whether binning should be used for preview (if HALF or QUARTER resolution is selected). [This option is currently not supported] | | |
| Syntax | <code>SetBinning(bool bBinning)</code> | | |
| Parameters | | | |
| | Input | bBinning | true = use binning for preview false = don't use binning (default) |
| | Output | None | |

SetCapBinning

| | | | |
|--------------------|--|-----------------|--|
| Description | Sets whether binning should be used by the sensor for video output (if HALF or QUARTER resolutions is selected). | | |
| Syntax | <code>SetCapBinning(bool bBinning)</code> | | |
| Parameters | | | |
| | Input | bBinning | true = use binning for video output false = don't use binning (default) |
| | Output | None | |

Snapshot Mode

When the camera is in Snapshot mode it stops outputting frames until an external trigger is set. Each time the trigger is tripped a snapshot is performed (displayed on SampleApplication video screen). You can order Camelot cameras with an optional external trigger.

EnterSnapshotMode

| | | | |
|--------------------|---|---------------------|--|
| Description | Enters or leaves Snapshot mode. When in SnapshotMode the camera stops capturing and streaming video until a trigger is set. | | |
| Syntax | <code>EnterSnapshotMode (bool snapshotMode)</code> | | |
| Parameters | | | |
| | Input | snapshotMode | true = Snapshot mode is ON false = Snapshot mode is OFF |
| | Output | None | |

GetSnapshot

| | | | |
|--------------------|---|--------------|---|
| Description | Imitates the optional external trigger and instructs the camera to capture one frame and send it to the PC. | | |
| Syntax | <code>GetSnapshot (bool flash)</code> | | |
| Parameters | | | |
| | Input | flash | true = use the external flash (optionally supplied with camera) when snapshot is triggered false = don't use flash |
| | Output | None | |

EnterPauseMode

| | | | |
|--------------------|---|------------------|--|
| Description | Enters or leaves pause mode. When in PauseMode the camera stops capturing and streaming video. All control messages are still sent. This message must be sent to the camera to notify it that video streaming is paused. | | |
| Syntax | <code>EnterPauseMode (bool pauseMode)</code> | | |
| Parameters | | | |
| | Input | pauseMode | true = Pause mode is ON false = Pause mode is OFF |
| | Output | None | |

Advanced API Functions

IMPORTANT NOTES

BEWARE the UpdateFW function writes control data to the camera - an operation that could render it non-functional if an invalid custom LDR file is being used.

You can use the UpdateFW function to update to newer, validated versions of the firmware available on our website.

In order to check a new *.ldr file that has not yet been validated by us at ID, please contact us so we can run it for you to assure the camera will boot after loading your file.

UpdateFW

| | | | |
|----------------------|------------------------------|--|---|
| Description | Updates the camera Firmware. | | |
| Syntax | UpdateFW(char* a_sFileName) | | |
| Parameters | | | |
| | Input | a_sFileName | Valid file name of type *.ldr- must exist and must be in correct analog loader file format. |
| | Output | None | |
| Return Values | S_OK | 0 - updated successfully | |
| | S_FALSE | 1 - camera FW was not updated successfully | |

UploadLUT

| | | | |
|----------------------|---|---------------------------------------|--|
| Description | <p>Uploads a look-up table (LUT) for translating raw pixel values in the camera before being outputted. The LUT can translate from 8-bit, 10-bit or 12-bit data to 8-bit, 10-bit or 12-bit output. In this version only translations to 8-bit data are supported, which is transmitted in the output. Once a LUT is loaded, the camera uses it to translate all outgoing frames.</p> <p>After the LUT is successfully loaded, the SampleApplication must reinitialize the camera and run it.</p> | | |
| Syntax | UploadLut(char* a_sFileName, int a_numBits, bool a_transform) | | |
| Parameters | | | |
| | Input | a_sFileName | Valid file name – must exist. Each value should be on a separate line. |
| | | a_numBits | 8, 10, or 12 For 8 bits, 256 values are expected in the file. For 10 bits, 1024 values are expected in the file. For 12 bits, 4096 values are expected in the file. |
| | | a_transform | Whether values should be transformed since 8 MSB are in lower byte. Should be true . |
| | Output | None | |
| Return Values | S_OK | 0 – uploaded successfully | |
| | S_FALSE | 1 – LUT was not uploaded successfully | |

SetTestData

| | | | |
|--------------------|---|--------------------|--|
| Description | <p>Specifies whether the camera should output set Test Data or regular video images. This is an option provided for developers to check whether the data received is actually the data sent and other sensor independent features.</p> <p>1.3Mp and 3Mp cameras have only 1 option – to choose a value which is output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED.</p> <p>For 5Mp cameras this is type=4. The 5Mp camera has other options for creating patterns.</p> | | |
| Syntax | <pre>SetTestData(bool useTestData, int type, int redData, int greenData, int blueData, int barWidth)</pre> | | |
| Parameters | | | |
| | Input | useTestData | <p>true – use Test Data</p> <p>false – don't use Test Data</p> |
| | | type | <p>Only applicable for 5Mp sensor.</p> <p>Values 0-8.</p> <p>0 - Color Field (Normal Operation – like 1.3Mp and 3Mp)</p> <p>1 - Horizontal Gradient</p> <p>2 - Vertical Gradient</p> <p>3 - Diagonal Gradient</p> <p>4 - Classic Test Pattern</p> <p>5 - Marching ones</p> <p>6 - Monochrome Horizontal Bars</p> <p>7 - Monochrome Vertical Bars</p> <p>8 - Vertical Color Bars</p> <p>For detailed description, see Micron's data sheets.</p> |
| | | redData | <p>Value for R pixel (in 5Mp).</p> <p>Value for Test Data in 1.3Mp and 3Mp – is output and then its complement in a pattern.</p> |
| | | greenData | Value for G pixel (in 5Mp). |
| | | blueData | Value for B pixel (in 5Mp). |
| | | barWidth | Width of bars if type 6, 7, or 8 should be an odd number |
| | Output | None | |

SetGetRawFullData

| | | | |
|--------------------|--|-----------------------|--|
| Description | Sets mode for getting RAW frames. For 5Mp camera only: If a 12-bit RAW frame is required, puts the camera into 48MHz mode (using the SetPIIRate function) before requesting the RAW frame. | | |
| Syntax | <pre>SetGetRawFullData (bool getRawFullData, int a_numBits, bool a_Resolution, bool withLUT, int a_numFrames)</pre> | | |
| Parameters | | | |
| | Input | getRawFullData | true – set RawFull data mode (FULL capture resolution regardless of preview resolution) false – revert back to preview resolution |
| | | a_numBits | Number of RAW data bits used internally for camera capture |
| | | a_Resolution | RAW frame Resolution. Default is FULL - Other resolutions not yet supported. <pre>FULL = 0, HALF = 1, QUARTER = 2, EIGHTH = 3, // not yet supported.</pre> |
| | | withLUT | Use a LUT before data is sent to PC. This option is only available if a LUT has been loaded to the camera. |
| | | a_numFrames | Number of frames captured. More than 1 frame is not currently supported. |
| | Output | None | |

Using the Callback Function

Callback functions are functions that are accessed whenever an event or interrupt occurs. In the Camelot API, the callback function is accessed, if so defined, whenever a new frame is received. There is one callback function that receives a parameter to identify the type of frame (Raw, or RGB, etc.) received and then proceeds to deal with it.

You can program the Sample Application to use a callback function whenever a raw (Bayer 8bit), or a RAW-FULL frame is received. This also includes when a frame is converted to RGB (RGB24 or RGB32). Also, any parameters or information needed by the function can be sent using the `pUserData` pointer.

At the moment one Edge filter function is used on RGB (24 and 32) frames, while all RAW FULL frames (8, 10, 12-bit) are saved using the Callback function.

SetDataCBPtr

| | | | |
|--------------------|---|----------------|--|
| Description | Sets the callback function that is called every time a new frame is received, if specified. | | |
| Syntax | <code>SetDataCBPtr(funcDataPtr funcPtr)</code> | | |
| Parameters | | | |
| | Input | funcPtr | <p>A pointer to an existing function of type</p> <pre>void func(unsigned char *rawDataPtr, int size, FRAME_TYPE frameType, void *pUserData)</pre> <p>In the Camelot implementation, FRAME_TYPE is one of the following:</p> <pre>BAYER_8_BIT // regular raw file BAYER_RAW // 8, 10 or 12 bit raw, Full data RGB_24 // BMP using RGB24 RGB_32 // BMP using RGB32</pre> <p>In the CamelotView sample, the Callback function is</p> <pre>DataCallback(unsigned char *dataPtr, int size, FRAME_TYPE type, void *pUserData),</pre> <p>which can be found in <code>Cbfunctions.cpp</code>. The user may edit or replace this function as appropriate.</p> <p>WARNING: This function must be performed quickly so as not to interfere with the frame rate.</p> |
| | Output | None | |

SetRawFrameDataCB

| | | | |
|--------------------|--|------------------|--|
| Description | Sets whether a callback function should be called for each raw frame received. | | |
| Syntax | <code>SetRawFrameDataCB (bool useCB, void *pUserdata)</code> | | |
| Parameters | | | |
| | Input | useCb | Whether this callback function should be set. |
| | | pUserData | Pointer to user data for the Callback function |
| | Output | None | |

SetRawFullFrameCB

| | | | |
|--------------------|--|------------------------|--|
| Description | Sets whether a callback function should be called for each raw Full frame received (as a result of the SetGetRawFullData function on page 27). | | |
| Syntax | SetRawFullFrameCB (<code>bool useCB, void *pUserdata</code>) | | |
| Parameters | | | |
| | Input | <code>useCb</code> | Whether this callback function should be set. |
| | | <code>pUserData</code> | Pointer to user data for the Callback function |
| | Output | <code>None</code> | |

SetRGBFrameDataCB

| | | | |
|--------------------|--|------------------------|--|
| Description | Sets whether a callback function should be called after each raw frame has been processed into an RGB frame. | | |
| Syntax | SetRGBFrameDataCB (<code>bool useCB, void *pUserdata</code>) | | |
| Parameters | | | |
| | Input | <code>useCb</code> | Whether this callback function should be set. |
| | | <code>pUserData</code> | Pointer to user data for the Callback function |
| | Output | <code>None</code> | |

SetPllRate

| | | | |
|--------------------|--|----------------------|---|
| Description | Sets the camera's PLL (internal clock) rate. Default: 96MHz. This option is not supported by the 1.3Mp and the 3Mp models. Before uploading and using a LUT with 12-bit streaming, the 5Mp camera must be set to 48MHz. Also, if a RAW frame is to be captured with 12-bit data, change the camera PLL rate to 48MHz before frame capture. | | |
| Syntax | SetPllRate (<code>int pllRate</code>) | | |
| Parameters | | | |
| | Input | <code>pllRate</code> | Sets the internal clock in MHz. Values must be in the range of 1 – 96. In this release only 48 and 96 MHz are supported. |
| | Output | <code>None</code> | |

SetPwmTimer

| | | | |
|--------------------|---|--|--|
| Description | Sets three PWM (pulse width modulation) timers which you can attach to peripherals (LEDs, etc.) that need PWM controlled current source. The period is the overall cycle, while the width is the duration of the period (duty cycle) where the signal is low. The width must be less than the period for each timer. The timers (TMR1, TMR2 and TMR3) are started simultaneously and therefore synchronized. To utilize this function a separate control board is required. | | |
| Syntax | <code>SetPwmTimer (PWM_API pwmTimer)</code> | | |
| Parameters | | | |
| Input | PWM_API | PWM_API struct which is defined as: <pre> unsigned int Timer1_period; // TMR1 period unsigned int Timer1_width; // TMR1 width unsigned int Timer2_period; // TMR2 period unsigned int Timer2_width; // TMR2 width unsigned int Timer3_period; // TMR3 period unsigned int Timer3_width; // TMR3 width </pre> | |
| Output | None | | |

GetGPIO

| | | | |
|--------------------|--|--|--|
| Description | Gets the current state and values of the 4 GPIO (general purpose input/output) pins that are reserved for your use. These pins can either be an input or an output, which means they either read a value from the camera, or set a value into the camera. Cameras can be optionally wired so that these GPIO pins can turn on/off a circuit or transmit info from the camera. To utilize this function a separate control board is required. | | |
| Syntax | <code>GetGPIO (GPIO_API *gpioApi)</code> | | |
| Parameters | | | |
| Input | *gpioApi | A pointer to a GPIO_API struct which is defined as: <pre> int GPIO_0_state; //0 = Input(read), 1 = Output(set) int GPIO_0_value; //if input - read, if output - set int GPIO_1_state; // 0 - Input (read), 1 - Output (set) int GPIO_1_value; // if input - read, if output - set int GPIO_2_state; // 0 - Input (read), 1 - Output (set) int GPIO_2_value; // if input - read, if output - set int GPIO_3_state; // 0 - Input (read), 1 - Output (set) int GPIO_3_value; // if input - read, if output - set </pre> | |
| Output | None | | |

SetGPIO

| | | | |
|--------------------|----------------------|--|--|
| Description | | Sets the current state and values of the 4 GPIO (general purpose input/output) pins that are reserved for the user. These pins can either be an input or an output – which means they either read a value from the camera, or set a value. Cameras can be optionally wired so that these GPIO pins can turn on/off a circuit or transmit info from the camera. To utilize this function a separate control board is required. | |
| Syntax | | <code>SetGPIO(GPIO_API gpioApi)</code> | |
| Parameters | | | |
| Input | <code>gpioApi</code> | A GPIO_API struct which is defined as: <pre>int GPIO_0_state; //0 = Input(read),1 = Output(set) int GPIO_0_value; //if input - read,if output - set int GPIO_1_state; // 0 - Input (read), 1 - Output (set) int GPIO_1_value; // if input - read, if output - set int GPIO_2_state; // 0 - Input (read), 1 - Output (set) int GPIO_2_value; // if input - read, if output - set int GPIO_3_state; // 0 - Input (read), 1 - Output (set) int GPIO_3_value; // if input - read, if output - set</pre> | |
| Output | None | | |

EnableSensorLight

| | | | |
|--------------------|----------------------|---|--|
| Description | | Turns on LEDs on the Sensor board. These can be optionally supplied with the camera. If Flash option is enabled when Snapshot is chosen, the LEDs light whenever a snapshot is taken. | |
| Syntax | | <code>EnableSensorLight(bool lightOn)</code> | |
| Parameters | | | |
| Input | <code>lightOn</code> | true - lights are turned on false - lights are off | |
| Output | None | | |

SetMonochrome

| | | | |
|--------------------|-------------------|--|--|
| Description | | Sets whether sensor outputs monochrome (greyscale) data. This information is used by the Display thread when converting the image from 8-bit Bayer format to RGB (24-bit or 32-bit). | |
| Syntax | | <code>SetMonochrome(bool mono)</code> | |
| Parameters | | | |
| Input | <code>mono</code> | true - sensor is monochrome false - sensor is color | |
| Output | None | | |

AdjustFlicker

| | | | |
|--------------------|--|---|--|
| Description | Adjusts the Shutter Width in order to avoid the 50Hz or 60Hz flicker experienced when using indoor lighting. | | |
| Syntax | AdjustFlicker(bool adjust, int freqHz) | | |
| Parameters | | | |
| Input | adjust | true - turns on this feature - whenever Shutter Width is modified, it will be changed to adjust the flicker false - no adjustment made for flicker | |
| | freqHz | For which frequency should the flicker be adjusted? Values: 50, 60 | |
| Output | None | | |

SetExposureTime

| | | | |
|--------------------|--|--|--|
| Description | Sets the exposure time (in milliseconds) for each frame. If the value given is lower than the camera can operate at, the camera operates at its minimum exposure time. | | |
| Syntax | SetExposureTime(int expTime) | | |
| Parameters | | | |
| Input | expTime | number of milliseconds for exposing each frame | |
| Output | none | | |

GetExposureTime

| | | | |
|--------------------|---|--|--|
| Description | Gets the actual exposure time (in milliseconds) for each frame according to the ShutterWidth and ShutterDelay and capture resolution. | | |
| Syntax | GetExposureTime(int *expTime) | | |
| Parameters | | | |
| Input | expTime | number of milliseconds each frame is exposed | |
| Output | none | | |

CheckLeds

| | | | |
|--------------------|--|--|--|
| Description | Checks whether the three external LEDs (optionally provided with the camera) are functioning. To utilize this function a separate control board is required. | | |
| Syntax | int CheckLeds() | | |
| Parameters | | | |
| Input | none | | |
| Output | int | An OR-ed value representing which LED is malfunctioning. A value of 0 means all LEDs are working properly. 1 - LED 1 is not working 2 - LED 2 is not working 4 - LED 3 is not working | |

SetLedStates

| | | | |
|--------------------|--|-------------|--|
| Description | Sets how many different LED states have been programmed into the camera. Changing between states is done by triggering a GPIO. The optional LEDs can be controlled differently in each LED state. To utilize this function a separate control board is required. | | |
| Syntax | <code>int CheckLeds ()</code> | | |
| Parameters | | | |
| | Input | none | |
| | Output | int | An OR-ed value representing which LED is malfunctioning. A value of 0 means all LEDs are working properly. 1 - LED 1 is not working 2 - LED 2 is not working 4 - LED 3 is not working |

GetLogs

| | | | |
|--------------------|---|-------------|--|
| Description | Gets log messages saved in the camera for debugging purposes. These messages are sent to a separate DebugPrint screen (if opened) and can be used to debug applications. See Using DebugPrint on page 35. | | |
| Syntax | <code>GetLogs ()</code> | | |
| Parameters | | | |
| | Input | none | |
| | Output | None | |

GetRegVal

| | | | |
|--------------------|--|-----------------|-----------------|
| Description | Returns value of specified register. | | |
| Syntax | <code>GetRegVal (int nRegAddr, int *pnRegVal)</code> | | |
| Parameters | | | |
| | Input | nRegAddr | Register number |
| | Output | pnRegVal | value |

SetRegVal

WARNING

This function can change register values to undefined or inadvisable values. This could cause the camera to malfunction or hang. Refer to Micron Data sheets for a full description of all registers and values.

| | | | |
|--------------------|---------------------------------------|-----------------|--|
| Description | Sets specified register value. | | |
| Syntax | SetRegVal (int nRegAddr, int nRegVal) | | |
| Parameters | | | |
| | Input | nRegAddr | Register number |
| | | nRegVal | Value to be set. Micron registers are 16-bit (2 bytes) so values range from 0-65535. Not all values are valid for each register – please read sensor data sheet before changing register values. |
| | Output | None | |

GetRegValues

| | | | |
|--------------------|--|---------------|---|
| Description | Outputs all defined register values to a REG_LIST_API structure as defined in hostCamApi.h . | | |
| Syntax | GetRegValues (REG_LIST_API *regApi) | | |
| Parameters | | | |
| | Input | none | |
| | Output | regApi | Structure with up to MAX_REG_NUM registers and their values |

SetRegValues

WARNING

This function can change register values to undefined or inadvisable values. This could cause the camera to malfunction or hang. Refer to Micron Data sheets for a full description of all registers and values.

| | | | |
|--------------------|--|---------------|---|
| Description | Sets register values to list in file. [This option is not currently supported] | | |
| Syntax | SetRegValues (REG_LIST_API regApi) | | |
| Parameters | | | |
| | Input | regApi | Structure with up to MAX_REG_NUM registers and their values |
| | Output | None | |

Using DebugPrint

This section describes how to use a separate window for sending debug messages while the application is running.

◆ **To use a separate window for sending debug messages:**

1. Include **BDRdebug.h** in your project.
2. Open the **BDR_debugger** window by running **BDR_debugger.exe**.

BDR_debugger.exe is included on the Installation CD.

3. In the code, call the **DebugPrint(msg)** function with an ASCII message as a parameter.

The message is displayed in the **BDR_debugger** window and logged into a file by time and date.

DebugPrint

| | | | |
|--------------------|---|-------------|---|
| Description | Prints a message to a debug window (if open) while in DEBUG mode. | | |
| Syntax | <code>DebugPrint(char * Str)</code> | | |
| Parameters | | | |
| | Input | *Str | Char string – ASCII message to be printed |
| | Output | none | |

| | | | |
|-------------------|---|-------------|---|
| Syntax | <code>DebugPrint(char * Str, int Val1)</code> | | |
| Parameters | | | |
| | Input | *Str | Char string – ASCII message to be printed |
| | | Val1 | Number to be printed |
| | Output | none | |

| | | | |
|-------------------|---|-------------|---|
| Syntax | <code>DebugPrint(char * Str, int Val1, int Val2)</code> | | |
| Parameters | | | |
| | Input | *Str | Char string – ASCII message to be printed |
| | | Val1 | Number to be printed |
| | | Val2 | Number to be printed |
| | Output | none | |

Website

<http://www.imagine2d.com/>

Support

We would appreciate any feedback you have about the Camelot EVK.

support@imagine2d.com

Sales

Please contact us if we can assist you in building your own custom applications.

sales@imagine2d.com



Kane Computing Ltd
7 Theatre Court, London Road,
Northwich, Cheshire, CW9 5HB, UK.
Tel: +44(0)1606 351006
Fax: +44(0)1606 351007/8
Email: sales@kanecomputing.com
Web: www.kanecomputing.co.uk



a member of the