Camelot

EVALUATION KIT (EVK)

CUSTOM CAMERAS FOR MASS PRODUCTION APPLICATIONS

API REFERENCE MANUAL

Print Version B10

October 2009

CONTENTS

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

This application is based on DirectShow filters - compiled and run using Microsoft Visual Studio 2005.

The BDR Sample Application was written to demonstrate to programmers how to build applications using the **BECFilter.ax** functional API.

This API gives the programmer full control over the Camelot series.

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

Chapter 1

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

We periodically update this software application with new commands. You can download the latest version of the software from our website. See *Upgrading the to the Latest Version* on page 6.

Downloads Contain:

- **BECFilter.ax –** a DirectShow filter containing functions that access and communicate with the camera - this file must be placed in the **C:\Windows\System32** folder
  Where: **C:\** is your system disk drive
- **CamelotSample.exe –** the **SampleApplication** executable
- **CamelotSampleCode.zip –** contains the latest update sources
- **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

Referenced Documents

This document references the data-sheets for all the digital image sensors listed below. The PDF version can be downloaded from the Micron website:

Data Sheet Name	Digital Image Sensor Size
Micron MT9M001C12STM	1.3 Megapixels
Micron MT9T03P12STC	3-Megapixels
Micron MT9P031i12STC	5-Megapixels
Micron MT9N001i12STC	9-Megapixels
MICRON MT9Vo24iA7XTC	Wide VGA

Links to Micron Data Sheets

Sensor	Link
1.3	http://www.aptina.com/products/image_sensors/mt9m001c12stm/
3	http://www.aptina.com/products/image_sensors/mt9t031p12stc/#overview
5	http://www.aptina.com/products/image_sensors/mt9p031i12stc/#overview
9	http://www.aptina.com/products/image_sensors/mt9n001i12stc/
WVGA	http://www.aptina.com/products/image_sensors/mt9v024ia7xtc/#overview

Chapter 2

API COMMAND SUMMARY TABLE

Command	Description
AdjustFlicker	Adjusts the Shutter Width in order to avoid the 50Hz or 60Hz flicker experienced when using indoor lighting.
CaptureBMP	Captures the next frame from the camera and saves it as a BMP image (RGB24).
CaptureRawImage	Captures the next frame from the camera and saves it as a RAW, GBRG Bayer image.
CheckLeds	Checks whether the three external LEDs, Red, Green and Blue (optionally provided with the camera) are functioning.
CloseCamera	Closes all connections to the selected camera and releases all buffers used for video capture.
DebugPrint	Prints a message to a debug window (if open) while in DEBUG mode.
EnableSensorLight	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.
EnterSnapShotMode	Enters or leaves Snapshot mode. When in SnapshotMode the camera stops capturing and streaming video until a trigger is set.
GetCamCaps	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.
GetCameraVersionInfo	Gets the camera's version numbers – HW, FW and others.
GetCaptureResolution	Returns the current resolution of the camera's output.
GetFlipHorizontal	Returns whether the picture is flipped horizontally.
GetFlipVertical	Returns whether the picture is flipped vertically.
GetFPS	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.
GetFPSrate	Outputs the Bit Rate of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning.
GetFrameNum	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.
GetGainType	Returns the sensor gain according to the type queried. See *Setting and Getting the Gain values* on page 14 for explanation of values.

Command	Description
GetGPIO	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.
GetGreyScale	Returns whether the picture is shown in greyscale.
GetLogs	Gets log messages saved in the camera and used for debugging purposes. These messages are sent to a separate *DebugPrint* screen that can optionally be used to debug applications.
GetRegion	Gets the coordinates of the region (ROI – Region of Interest) programmed in the registers of the camera.
GetRegVal	Returns value of specified register.
GetResolution	Returns the current resolution of the Preview screen.
GetSensorType	Returns the type of Micron sensor present in the camera. The type may be 1.3, 3, 5 or 9 Megapixel sensor according to the table below.
GetShutterDelay	Returns the sensor shutter delay. This value is used for controlling Exposure time and to avoid any 50Hz or 60Hz flicker.
GetShutterWidth	Sets the sensor shutter width. This value controls the time taken between each new frame capture. If in *AdjustFlicker* mode the value supplied is adjusted to the next highest value to avoid any 50/60Hz flicker.
GetSnapShot	Imitates the optional external trigger and instructs the camera to capture one frame and send it to the PC.
GetView	Gets the coordinates of the FOV (Frame of View) displayed on the PC.
InitCamera	Initializes the DirectShow filters needed to run **the** SampleApplication. Within the function, **once** the pFilter is instantiated, the camera is initialized and ready for commands. Commands are sent **using** the **piBDR** interface.
OnPause	Pauses the selected camera that is running. **OnRun()** restarts it.
OnRun	Starts the selected camera that has been either paused or stopped. If the camera wasn't initialized, it is initialized and then started.
RunCamera	Starts a camera that has already been initialized. When called, video is streamed from the camera and displayed in a separate **Preview** window.
SetBinning	Sets whether binning should be used for preview (if HALF or QUARTER resolution is selected). **[This option is not yet supported]**
SetCapBinning	Sets whether binning should be used by the sensor for video output (if HALF or QUARTER resolutions is selected).
SetCaptureResolution	Sets the resolution used for the camera's output.
SetDataCBPtr	Sets the callback function that is called every time a new frame is received, if specified.
SetExposureTime	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.

Command	Description
SetFlipHorizontal	Sets whether the picture should be flipped horizontally.
SetFlipVertical	Sets whether the picture should be flipped vertically.
SetGainType	Sets sensor's gain according to the type specified. See Setting and Getting the Gain values on page 14.
SetGetRawFullData	Sets mode for getting RAW frames. For 5Mp camera only: If a 12-bit RAW frame is required, put the camera into 48MHz mode (using the SetPllRate function) before requesting the RAW frame.
SetGPIO	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.
SetGreyScale	Sets whether the picture should be shown using greyscale.
SetPllRate	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.
SetPwmTimer	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.
SetRawFrameDataCB	Sets whether a callback function should be called for each raw frame is received.
SetRawFullFrameCB	Sets whether a callback function should be called for each raw Full frame received (as a result of the SetGetRawFullData function on page 26).
SetRegion	Sets the region (ROI - Region of Interest) to be output by the camera.
SetRegVal	Sets specified register value.
SetResolution	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.
SetRGBFrameDataCB	Sets whether a callback function should be called after each raw frame has been processed into an RGB frame.
SetRotation	Sets the angle in degrees (0, 90, 180 and 270) for rotating the image.
SetShutterDelay	Sets the sensor shutter delay This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker.
SetShutterWidth	Sets the sensor shutter width This value controls the sensor delay time between each new frame capture.

Command	Description
SetTestData	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 will be output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED. For 5Mp cameras this is type=4 and you can create patterns and change the RGB values.
SetView	Sets the FOV (Field of View) to be displayed on the PC
ShutDownCamera	Stops camera and tears down the DirectShow filter graph. Calls API functions *StopDriver* and *CloseCamera.*
StopCamera	Stops a camera that is running, but keeps it initialized. Calls API function *StopDriver.*
StopDriver	Stops the selected camera's driver. Camera stops outputting video and enters a **waiting for command** mode.
UpdateFW	Updates the camera Firmware.
UploadLUT	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.

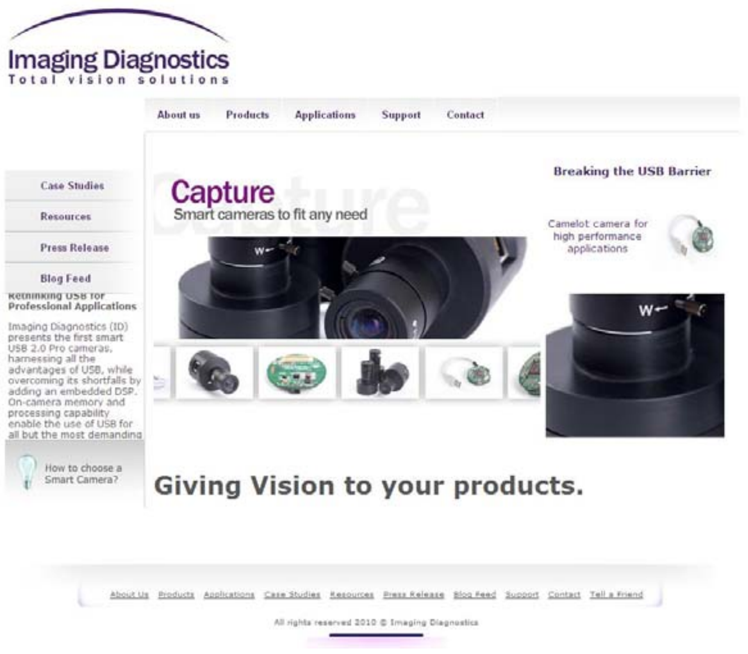
Chapter 3

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:**

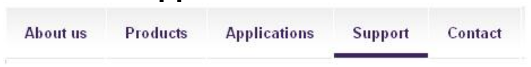
1. Download the latest version of the software from **http://www.imagine2d.com/**


Imaging Diagnostics
Total vision solutions
About us
Products
Applications
Support
Contact
Case Studies
Resources
Press Release
Blog Feed
Capture
Smart cameras to fit any need
Breaking the USB Barrier
Camelot camera for high performance applications
How to choose a Smart Camera?
Giving Vision to your products.
About Us Products Applications Case Studies Resources Press Release Blog Feed Support Contact Tell a Friend
All rights reserved 2010 © Imaging Diagnostics

Rethinking USB for Professional Applications

Imaging Diagnostics (ID) presents the first smart USB 2.0 Pro cameras, harnessing all the advantages of USB, while overcoming its shortfalls by adding an embedded DSP. On-camera memory and processing capability enable the use of USB for all but the most demanding

2. Click the **Support** tab.



3. From the menu, click **Downloads**.
4. Download the latest version of the Camelot software and install it.
5. Upload the new firmware to the camera using the current/old working **SampleApplication** (**NOT** the file you just downloaded).
6. When the firmware upload is successfully completed, remove the USN cable.
7. Copy the **BECfilter.ax** file into the folder **C:\Windows\system32**
   Where: **C:\** is the system disk.
8. Reconnect the camera and run the **NEW SampleApplication** file.

Chapter 4

INITIALIZING AND RUNNING THE CAMERA

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

Main Dialog module function calls – for **BDR_SampleCodeDlg.cpp**.

These functions, based on DirectShow filter graphs, are used to run, pause and stop the camera.

InitCamera

Description		Initializes the DirectShow filters needed to run the **SampleApplication**. Within the function, once the **pFilter** is instantiated, the camera is initialized and ready for commands. Commands are sent using the **piBDR** interface. All of the commands in this API 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 place in the **List Control** displayed on the **Main** screen.	
Syntax		`InitCamera(int a_nCamNum, bool *CameraFound)`	
Parameters			
	Input	**a_nCamNum**	Camera number
	Output	**CameraFound**	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		Stops camera and tears down the DirectShow filter graph. 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 camera that is running, but keeps it initialized. Calls API function *StopDriver*.	
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.	
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.	
Syntax		`OnRun()`	
Parameters			
	Input	**None**	
	Output	**None**	
Return Values		**None**	

Chapter 5

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 or 9 Megapixel sensor according to the table below.			
Syntax		`GetSensorType (int *pnSensorType)`			
Parameters					
	Input	**None**			
	Output	`pnSensorType`	SENSOR_UNKNOWN	0	
			SENSOR_WVGA	3	1.3 Megapixel
			SENSOR_1300	13	3 Megapixel
			SENSOR_3000	30	5 Megapixel
			SENSOR_5000	50	9 Megapixel
			SENSOR_9000	90	

GetCameraVersionInfo

Description		Gets the camera's version numbers – HW, FW and others.	
Syntax		`GetCameraVersionInfo(char *versionInfo)`	
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`	`CAMERA_CAP_API struct which is defined as:`
			`   unsigned int SensorType; // see GetSensorType`
			`   char         Desc[SENSOR_DESC_LEN];`
			`   unsigned int CameraID;         // serial number`
			`   unsigned int FirmwareVersion; // n.a.`
			`   unsigned int HardwareVersion; // n.a.`
			`   unsigned int Width;            // maximum width`
			`   unsigned int Height;           // maximum height`
			`   unsigned int ActiveStartX;     // x column start`
			`  unsigned int ActiveStartY;      // y row start`

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		`unsigned long GetFrameNum(void)`	
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		`GetFPS(float *pFPS, float *pSkipPS, float *pDisplayPS)`	
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		`GetFPSrate(float *pFPS, float *pDisplayPS)`	
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		`GetFlipHorizontal(bool *pbFlipHorizontal)`	
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		`SetFlipHorizontal(bool bFlipHorizontal)`	
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		`GetFlipVertical(bool *pbFlipVertical)`	
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	`SetFlipVertical(bool bFlipVertical)`		
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	`SetRotation(int nRotation)`		
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	`GetGreyScale(bool *pbGreyScale)`		
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 (RGB24).	
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 14 for explanation of values.	
Syntax		`GetGainType(int type, int *pnGain)`	
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 14.	
Syntax		`SetGainType(int type, int nGain)`	
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. See *Referenced Documents* on page 1.

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		`GetShutterDelay(int *pnShutterDelay)`	
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		`SetShutterDelay(int nShutterDelay)`	
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 *AdjustFlicker* mode the value supplied is adjusted to the next highest value to avoid any 50/60Hz flicker.	
Syntax		`GetShutterWidth(int *pnShutterWidth)`	
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		`SetShutterWidth(int nShutterWidth)`	
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	`GetResolution(int *pnResolution)`		
Parameters			
	Input	**None**	

	Output	`pnResolution`	`FULL    = 0,`
			`HALF    = 1,`
			`QUARTER = 2,`
			`EIGHTH  = 3,   // not yet supported`

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	`SetResolution(int nResolution)`		
Parameters			

	Input	`nResolution`	`FULL    = 0,`
			`HALF    = 1,`
			`QUARTER = 2,`
			`EIGHTH  = 3,   // not yet supported`

	Output	**None**	

GetCaptureResolution

Description	Returns the current resolution of the camera's output.		
Syntax	`GetCaptureResolution(int *pnResolution)`		
Parameters			
	Input	**None**	

	Output	`pnResolution`	`FULL    = 0,`
			`HALF    = 1,`
			`QUARTER = 2,`
			`EIGHTH  = 3,   // not yet supported.`

SetCaptureResolution

Description		Sets the resolution used for the camera's output.	
Syntax		`SetCaptureResolution(int nResolution)`	
Parameters			

	Input	nResolution	`FULL = 0,`
			`HALF = 1,`
			`QUARTER = 2,`
			`EIGHTH = 3, // not yet supported.`

	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		`GetRegion(int *startX, int *startY, int *width, int *height)`	
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		`SetRegion(int startX, int startY, int width, int height)`	
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	

GetView

Description		Gets the coordinates of the FOV (Frame of View) displayed on the PC.	
Syntax		`GetView(int *startX, int *startY, int *width, int *height)`	
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.	
Syntax		`SetView(int startX, int startY, int width, int height)`	
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 not yet supported]**	
Syntax		`SetBinning(bool bBinning)`	
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		`SetCapBinning(bool bBinning)`	
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		`EnterSnapShotMode(bool snapshotMode)`	
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		`GetSnapShot(bool flash)`	
Parameters			
	Input	`flash`	**true** = use the external flash (optionally supplied with camera) when snapshot is triggered **false** = don't use flash
	Output	**None**	

Chapter 6

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		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.	
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		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 will be output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED. For 5Mp cameras this is type=4 and you can create patterns and change the RGB values.	
Syntax		`SetTestData(bool useTestData, int type, int redData, int greenData, int blueData, int barWidth)`	
Parameters			
	Input	**useTestData**	**true** – use Test Data **false** – don't use Test Data
		type	Only applicable for 5Mp sensor. Values 0-8. 0 - Color Field (Normal Operation – like 1.3Mp and 3Mp) 1 - Horizontal Gradient 2 - Vertical Gradient 3 - Diagonal Gradient 4 - Classic Test Pattern 5 - Marching ones 6 - Monochrome Horizontal Bars 7 - Monochrome Vertical Bars 8 - Vertical Color Bars For detailed description, see Micron's See *Links to Micron Data Sheets* on page 1
		redData	Value for **R** pixel (in 5Mp). Value for Test Data in 1.3Mp and 3Mp – is output and then its compliment in a pattern.
		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**	

SetDataCBPtr

Description	Sets the callback function that is called every time a new frame is received, if specified.		
Syntax	`SetDataCBPtr(funcDataPtr funcPtr)`		
Parameters			

	Input	`funcPtr`	A pointer to an existing function of type
			void func(unsigned char `*rawDataPtr`, int size, FRAME_TYPE frameType)
			
			In the Camelot implementation, FRAME_TYPE is one of the following:
			`BAYER_8_BIT = 10, // regular raw file`
			`BAYER_RAW = 11 // 8, 10 or 12 bit raw, Full data`
			`RGB_24 = 13 // BMP using RGB24`
			
			In the CameloSample, this function is implemented with the placeholder function `DataCallback( unsigned char *dataPtr, int size, FRAME_TYPE type)`, which can be found in `Cbfunctions.cpp`. The user may edit or replace this do-nothing placeholder function as appropriate.
			WARNING: This function must be performed quickly so as not to interfere with the frame rate.

	Output	**None**	

SetRawFrameDataCB

Description	Sets whether a callback function should be called for each raw frame is received.		
Syntax	`SetRawFrameDataCB (bool useCB)`		
Parameters			
	Input	**useCb**	Whether this callback function should be set.
	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 26).		
Syntax	`SetRawFullFrameCB (bool useCB)`		
Parameters			
	Input	**useCb**	Whether this callback function should be set.
	Output	**None**	

SetRGBFrameDataCB

Description		Sets whether a callback function should be called after each raw frame has been processed into an RGB frame.	
Syntax		`SetRGBFrameDataCB (bool useCB)`	
Parameters			
	Input	`useCb`	Whether this callback function should be set.
	Output	**None**	

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(int pllRate)`	
Parameters			
	Input	`pllRate`	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	**None**	

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.	
Syntax		`SetPwmTimer (PWM_API pwmTimer)`	
Parameters			

	Input	**PWM_API**	`PWM_API struct which is defined as:`
			`   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`

	Output	**None**	

SetGetRawFullData

Description		Sets mode for getting RAW frames. For 5Mp camera only: If a 12-bit RAW frame is required, put the camera into 48MHz mode (using the *SetPllRate* function) before requesting the RAW frame.	
Syntax		`SetGetRawFullData(bool getRawFullData, int a_numBits, bool a_Resolution, bool withLUT, int a_numFrames)`	
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.
			`FULL    = 0,`
			`HALF    = 1,`
			`QUARTER = 2,`
			`EIGHTH  = 3,   // not yet supported.`

		`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**	

Syntax as printed:

```
SetGetRawFullData(bool getRawFullData,
                  int a_numBits,
                  bool a_Resolution,
                  bool withLUT,
                  int a_numFrames)
```

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.		
Syntax	`GetGPIO(GPIO_API *gpioApi)`		
Parameters			

	Input	*gpioApi	**A pointer to a GPIO_API struct which is defined as:**
			`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`

	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.		
Syntax	`SetGPIO(GPIO_API gpioApi)`		
Parameters			

	Input	gpioApi	**A GPIO_API struct which is defined as:**
			`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`

	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		`EnableSensorLight(bool lightOn)`	
Parameters			
	Input	`lightOn`	**`true`** – lights are turned on **`false`** – lights are off
	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	

CheckLeds

Description	Checks whether the three external LEDs, Red, Green and Blue (optionally provided with the camera) are functioning.		
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

GetLogs

Description	Gets log messages saved in the camera and used for debugging purposes. These messages are sent to a separate *DebugPrint* screen that can optionally be used to debug applications. See also *Using DebugPrint* on page 31.		
Syntax	`GetLogs()`		
Parameters			
	Input	none	
	Output	None	

GetRegVal

Description	Returns value of specified register.		
Syntax	`GetRegVal(int nRegAddr, int *pnRegVal)`		
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. See *Referenced Documents* on page 1.

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**	

Chapter 7

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. Run the application in DEBUG mode.
3. Open the **BDR_debugger** window by running **BDR_debugger.exe**.

   BDR_debugger.exe is included on the Installation CD.
4. 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		`DebugPrint(char * Str)`	
Parameters			
	Input	***Str**	Char string – ASCII message to be printed
	Output	**none**	

Imaging Diagnostics
Total vision solutions

a member of the BDR GROUP