



# **EVALUATION KIT (EVK)**

CUSTOM CAMERAS FOR MASS PRODUCTION APPLICATIONS

## **APPLICATION PROGRAM INTERFACE**

**©Copyright Imaging Diagnostics 2009-2011**

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.

**Revision 1.7**

**Software Version 2.1.1423**

**Print Version 03**

**August 2011**

# CONTENTS

<b>PREFACE</b> .....	<b>VII</b>
<b>ABOUT THIS DOCUMENT</b> .....	<b>VII</b>
<b>TARGET AUDIENCE</b> .....	<b>VII</b>
<b>ABOUT THIS APPLICATION</b> .....	<b>VII</b>
<b>IMPORTANT INFORMATION</b> .....	<b>VII</b>
<b>HOW TO CONTACT US</b> .....	<b>VIII</b>
<b>CHAPTER 1. INTRODUCTION</b> .....	<b>1</b>
<b>WHAT'S NEW IN VERSION 2.1.1290</b> .....	<b>2</b>
List of Updated Commands.....	2
List of New Commands .....	3
<b>CHAPTER 2. API COMMAND SUMMARY TABLE</b> .....	<b>4</b>
<b>CHAPTER 3. INITIALIZING AND ACCESSING THE CAMERA</b> .....	<b>10</b>
<i>CamEnumCams</i> .....	11
<i>CamEnumCamsEx</i> .....	11
<i>FillCameraList</i> .....	12
<i>FillCameraListEx</i> .....	12
<b>CHAPTER 4. CONTROLLING THE CAMERA</b> .....	<b>13</b>
<b>GETTING CAMERA PARAMETERS</b> .....	<b>13</b>
<i>GetSensorType / CamGetSensorType</i> .....	13
<i>GetCameraVersionInfo / CamGetCameraVersionInfo</i> .....	14
<i>GetCamCaps / CamGetCamCaps</i> .....	15
<i>OpenCamera / CamOpenCamera</i> .....	15
<i>GetState / CamGetState</i> .....	15
<i>CamGetMaxRegion</i> .....	16
<b>CONTROLLING THE VIDEO STREAMING</b> .....	<b>16</b>
<i>StartVideo / CamStartVideo</i> .....	16
<i>StartVideoStreaming / CamStartVideoStreaming</i> .....	17
<i>SetViewWindow / CamSetViewWindow</i> .....	18
<i>GetViewWindow / CamGetViewWindow</i> .....	19
<i>SetPixelBits / CamSetPixelBits</i> .....	19
<i>GetPixelBits / CamGetPixelBits</i> .....	19
<i>Run / CamRun</i> .....	19
<i>Pause / CamPause</i> .....	20
<i>StopCamera / CamStopCamera</i> .....	20
<i>CloseCamera / CamCloseCamera</i> .....	21
<i>ResetCamera / CamResetCamera</i> .....	21
<i>GetFrameNum / CamGetFrameNum</i> .....	21

<i>DisplayFrameNumTime / CamDisplayFrameNumTime</i> .....	22
<i>GetFPS / CamGetFPS</i> .....	22
<i>GetFPSrate / CamGetFPSrate</i> .....	22
<b>SETTING AND GETTING THE GAIN VALUES</b> .....	<b>23</b>
<i>SetGainType / CamSetGainType</i> .....	23
<i>GetGainType / CamGetGainType</i> .....	23
<i>SetGainProportions / CamSetGainProportions</i> .....	24
<b>EXPOSURE TIME AND SHUTTER CONTROL</b> .....	<b>25</b>
<i>GetShutterWidth / CamGetShutterWidth</i> .....	25
<i>SetShutterWidth / CamSetShutterWidth</i> .....	25
<i>GetShutterWidthGains / CamGetShutterWidthGains</i> .....	26
<i>GetShutterDelay / CamGetShutterDelay</i> .....	26
<i>SetShutterDelay / CamSetShutterDelay</i> .....	26
<i>AdjustFlicker / CamAdjustFlicker</i> .....	27
<i>SetExposureTime / CamSetExposureTime</i> .....	27
<i>SetFrameRate / CamSetFrameRate</i> .....	27
<i>GetExposureTime / CamGetExposureTime</i> .....	28
<i>CalcShutWidthForExpTime / CamCalcShutWidthForExpTime</i> .....	28
<b>UNDERSTANDING THE CAMERA RESOLUTION TYPES</b> .....	<b>28</b>
<i>GetResolution / CamGetResolution</i> .....	29
<i>SetResolution / CamSetResolution</i> .....	29
<i>GetCaptureResolution / CamGetCaptureResolution</i> .....	29
<i>SetCaptureResolution / CamSetCaptureResolution</i> .....	30
<b>SETTING A REGION OF INTEREST (ROI)/FRAME OF VIEW (FOV)</b> .....	<b>30</b>
<i>GetRegion / CamGetRegion</i> .....	30
<i>SetRegion / CamSetRegion</i> .....	31
<b>BINNING</b> .....	<b>31</b>
<i>SetCapBinning / CamSetCapBinning</i> .....	31
<i>GetCapBinning / CamGetCapBinning</i> .....	32
<b>SNAPSHOT MODE</b> .....	<b>32</b>
<i>EnterSnapShotMode / CamEnterSnapShotMode</i> .....	32
<i>GetSnapShotMode / CamGetSnapShotMode</i> .....	32
<i>GetSnapShot / CamGetSnapShot</i> .....	33
<b>CAPTURING FRAMES TO FILE</b> .....	<b>33</b>
<i>CaptureFrame / CamCaptureFrame</i> .....	33
<i>CaptureFrameDone / CamCaptureFrameDone</i> .....	34
<i>CaptureFrameDoneEx / CamCaptureFrameDoneEx</i> .....	34
<i>CaptureFrameFileBuf / CamCaptureFrameFileBuf</i> .....	35

	<i>GetFrameWithFlash / CamGetFrameWithFlash</i> .....	35
<b>CHAPTER 5. PREVIEW CONTROLS</b> .....		<b>36</b>
	<i>GetFlipHorizontal / CamGetFlipHorizontal</i> .....	36
	<i>SetFlipHorizontal / CamSetFlipHorizontal</i> .....	36
	<i>GetFlipVertical / CamGetFlipVertical</i> .....	36
	<i>SetFlipVertical / CamSetFlipVertical</i> .....	37
	<i>SetRotation / CamSetRotation</i> .....	37
	<i>GetGreyScale / CamGetGreyScale</i> .....	37
	<i>SetGreyScale / CamSetGreyScale</i> .....	38
	<i>SetMonochrome / CamSetMonochrome</i> .....	38
	<i>GetMonochrome / CamGetMonochrome</i> .....	38
	<i>SetFixBadPixels / CamSetFixBadPixels</i> .....	39
<b>CHAPTER 6. ADVANCED API COMMANDS</b> .....		<b>40</b>
	<i>SetTestData / CamSetTestData</i> .....	40
	<i>SetLowPowerMode / CamSetLowPowerMode</i> .....	41
	<i>SetBadFrameDetection / CamSetBadFrameDetection</i> .....	41
	<i>SetGetRawFullData / CamSetGetRawFullData</i> .....	41
	<b>DESCRIPTION OF THE PLL RATE FOR 5MP CAMERAS</b> .....	<b>42</b>
	<i>SetPIIRate / CamSetPIIRate</i> .....	42
	<i>GetPIIRate / CamGetPIIRate</i> .....	42
	<b>COMMANDS USING THE GPIO CONNECTOR ON THE CAMERA</b> .....	<b>43</b>
	<i>SetPwmTimer / CamSetPwmTimer</i> .....	43
	Using General Purpose Input/Output (GPIO) .....	44
	<i>GetGPIO / CamGetGPIO</i> .....	44
	<i>SetGPIO / CamSetGPIO</i> .....	44
	<i>GetGPIO_Pins / CamGetGPIO_Pins</i> .....	45
	<i>SetGPIO_Pins / CamSetGPIO_Pins</i> .....	45
	<i>CheckLeds / CamCheckLeds</i> .....	45
	<b>SETTING/GETTING SENSOR REGISTER VALUES</b> .....	<b>46</b>
	<i>GetRegVal / CamGetRegVal</i> .....	46
	<i>SetRegVal / CamSetRegVal</i> .....	46
	<i>GetRegValues / CamGetRegValues</i> .....	47
	<i>SetRegValues / CamSetRegValues</i> .....	47
	<b>LOOK-UP TABLE (LUT)</b> .....	<b>48</b>
	<i>UploadLUT / CamUploadLUT</i> .....	48
	<i>SetLUT / CamSetLUT</i> .....	48
<b>CHAPTER 7. IMAGE PROCESSING COMMANDS</b> .....		<b>49</b>
	<b>COLOR CORRECTION OPTIONS – PROCESSING ON PC</b> .....	<b>49</b>
	Histogram Stretching .....	49

Histogram Equalization .....	49
Color Correction (CCM).....	49
<i>SetColorCorr / CamSetColorCorr.....</i>	<i>50</i>
<b>AWB (AUTOMATIC WHITE BALANCE) .....</b>	<b>50</b>
<b>AGC (AUTOMATIC GAIN CONTROL) .....</b>	<b>50</b>
<i>SetAGC_AWB / CamSetAGC_AWB.....</i>	<i>52</i>
<i>SingleFrameAWB / CamSingleFrameAWB.....</i>	<i>52</i>
<i>PC_AWB / CamPC_AWB.....</i>	<i>52</i>
<b>CHAPTER 8. CONFIGURING THE CAMELOT IP CAMERAS.....</b>	<b>53</b>
<i>SetIpConfig / CamSetIpConfig.....</i>	<i>53</i>
<i>GetIpConfig / CamGetIpConfig .....</i>	<i>54</i>
<i>SetIpDefaultConfig / CamSetIpDefaultConfig.....</i>	<i>54</i>
<i>GetIpDefaultConfig / CamGetIpDefaultConfig .....</i>	<i>54</i>
<i>SetPortRange / CamSetPortRange .....</i>	<i>55</i>
<b>CHAPTER 9. USING THE CALLBACK COMMANDS .....</b>	<b>56</b>
<i>SetDataCBPtr / CamSetDataCBPtr .....</i>	<i>57</i>
<i>SetRawFrameDataCB / CamSetRawFrameDataCB .....</i>	<i>58</i>
<i>SetRawFullFrameCB / CamSetRawFullFrameCB .....</i>	<i>58</i>
<i>SetRGBFrameDataCB / CamSetRGBFrameDataCB.....</i>	<i>58</i>
<i>SaveRawFrameToFileBuf / CamSaveRawFrameToFileBuf.....</i>	<i>59</i>
<i>SetMsgCBPtr.....</i>	<i>59</i>
<b>CHAPTER 10. USING DEBUGPRINT .....</b>	<b>60</b>
<i>DebugPrint .....</i>	<i>60</i>
<i>GetLogs / CamGetLogs.....</i>	<i>61</i>
<b>CHAPTER 11. UPGRADING TO THE LATEST SOFTWARE VERSION.....</b>	<b>62</b>
<i>UpdateFW / CamUpdateFW.....</i>	<i>63</i>
<b>APPENDIX A MOLEX CONNECTOR TABLE.....</b>	<b>64</b>
<b>MOLEX CONNECTOR .....</b>	<b>64</b>

## About this Document

This document is the Camelot and IDM Camera Series' Evaluation Kit (EVK) API.

## Target Audience

This document is meant for application programmers who wish to integrate Camelot and IDM-500 series cameras into 3<sup>rd</sup> party applications.

## About this Application

The CamelotView Sample Application was written to demonstrate to programmers how to build applications using the **Camelot.dll** functional API. **This API gives the programmer full control over Camelot and IDM-500 Series Cameras.**

All C/C++ applications must include the following files:

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

All C# applications must include the following files:

- **hostCamApi.cs** – a file containing definitions used by the camera, the API and the sample applications
- **CamelotDll.cs** – a list of API commands in the **Camelot.dll** that can be accessed by your application.

The final RGB images (BMP) from the CamelotDll (CamelotView) are in 32-bit format.

**If you want to use the DirectShow version, you must first compile the Baseclasses (Debug and Release) on your working computer and have CamelotFilter.ax in your \$(System) directory.**

See **DirectShow** documentation for more details.

## Important Information

### NOTE

Important information is shown as notes.

---

### WARNING

Important Safety information is displayed in this format

---

## How to Contact Us

**Imaging Diagnostics** invites comments and feedback on this and all of our products and documentation. Please contact us at one of the email addresses below.

### Website

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

### Support

[support@imagine2d.com](mailto:support@imagine2d.com)

### Sales

[sales@imagine2d.com](mailto: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 capability decreases the camera/host bandwidth requirement significantly. These 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 to the Latest Software](#) Version on page 62.

---

### Downloads Contain:

- **CamelotView.zip** – Contains the latest, updated sources of the Sample applications.
- **Camelot.dll** – A library of commands that access and operate the camera
- **CamelotView.exe** – An executable that uses the **Camelot.dll** and when running must be in the same directory.
- **CamelotFilter.ax** – a DirectShow filter containing commands 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.
- **Camelot\_FW.ldr** – This is the loader file with the new/updated code for the camera. This file must be loaded using the Sample Application or by calling the API command **UpdateFW** and supplying the path to the file.

See [Upgrading to the Latest Software](#) Version on page 62.

## What's New in Version 2.1.1290

Please read the section [About this Application](#) in the **Preface** above for instructions about writing C#/.NET applications.

- Product Hardware and Firmware versions are updated and stored as strings
- Single Frame AWB algorithm and PC\_AWB added.
- 10Mp resolution changes coded into dll, snapshot mode
- Better StartCamera/StopCamera/StartCamera sequence – no memory loss and more efficient
- If frame resolution or ROI would mean an odd value for either Frame Width or Frame Height, odd value is changed to an even value
- Added Get... commands to additional Set... commands.
- If streaming data that is > 8bit - simple gamma conversion is performed before display
- Snapshot serialization - sets of pictures are captured
- High quality BMP captured correctly while streaming display is regular BMP quality
- 2 new FRAME\_TYPES – ACK\_REQUEST and BAYER\_RAW\_CMP – 2 pixels of 12-bit data compressed into 3 bytes
- new SAVE\_FRAME\_TYPE - BMP\_NI - RGB24 - no header and non-color-interleaved (RR..R, GG..G, BB..B)
- Added commands for writing and reading to flash
- FindIpCameras – searches over multiple gateways

### List of Updated Commands

Updated Commands	Changes	Page
<a href="#">CaptureFrameFileBuf</a>	For this composite command to work, the following structures must be defined. All the parameters needed for saving the video are included.	35
<a href="#">CamEnumCamsEx</a>	<p>Same as CamEnumCams (above) with additional parameters:</p> <ul style="list-style-type: none"> <li>● specify whether USB or IP cameras are being searched</li> <li>● whether a camera with a specific serial number is requested</li> </ul> <p>For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</p>	12
<a href="#">FillCameraListEx</a>	<p>Same as FillCameraList (above) with additional parameters:</p> <ul style="list-style-type: none"> <li>● specify whether USB or IP cameras are being searched</li> <li>● whether a camera with a specific serial number is requested</li> </ul> <p>For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</p>	12

## List of New Commands

New Commands	Description	Page
<a href="#"><u>CaptureFrameDoneEx</u></a>	Returns TRUE when the amount of frames specified has been captured.	34
<a href="#"><u>GetIpDefaultConfig</u></a>	Get IP default configurations – similar to GetIPConfig – but refers to default configurations.	54
<a href="#"><u>SetIpDefaultConfig</u></a>	Set IP default configurations – similar to SetIPConfig – but refers to default configurations.	54
<a href="#"><u>SetPortRange</u></a>	Set range of ports to be used by IP application.	55
<a href="#"><u>SetGainProportions</u></a>	<p>The three R G B parameters are set as ratios between the gains – not by absolute values. When the R gain is changed, G and B are changed proportionately.</p> <p>Until a subsequent call, the gains will have the ratio <math>1024:((G*1024)/R):((B*1024)/R)</math>. Integer math is used in the scaling.</p>	24

## API COMMAND SUMMARY TABLE

---

**NOTE**

All of the commands are listed below in alphabetical order.

---

Command	Description
<a href="#"><u>AdjustFlicker</u></a>	Adjusts the Shutter Width to avoid the 50Hz or 60Hz flicker experienced when using indoor lighting.
<a href="#"><u>CalcShutWidthForExpTime</u></a>	Returns the shutter width that should be programmed in order to get the exposure time (in microseconds) indicated.
<a href="#"><u>CamEnumCams</u></a>	Searches for all Camelot and IDM-500 cameras connected to your PC USB port or on the LAN. After calling this command the cameras found can be accessed using the Cam... version of the API commands. The camera number is also provided, zero-based, as the first parameter.

Command	Description
<a href="#"><u>CamEnumCamsEx</u></a>	<p>Same as CamEnumCams (above) with additional parameters:</p> <ul style="list-style-type: none"> <li>• specify whether USB or IP cameras are being searched</li> <li>• whether a camera with a specific serial number is requested</li> </ul> <p>For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</p>
<a href="#"><u>CamGetMaxRegion</u></a>	<p>Gets the maximum coordinates of the region of interest, ROI, that the camera is capable of capturing. When accessing these commands through interface and camera's pointer, these values are in the CamCaps (CAMERA_CAP_API) struct.</p>
<a href="#"><u>CaptureFrame</u></a>	<p>Captures the next frame from the camera and saves it as a RAW, BMP or JPEG image. If in Snapshot mode, will capture the next frame after the GetSnapShot() command is called.</p>
<a href="#"><u>CaptureFrameDone</u></a>	<p>Used to poll if frame was captured. After calling CaptureFrame. This function will return whether or not the frame was saved.</p>
<a href="#"><u>CaptureFrameDoneEx</u></a>	<p>Returns TRUE when the amount of frames specified has been captured.</p>
<a href="#"><u>CaptureFrameFileBuf</u></a>	<p>For this composite command to work, the following structures must be defined. All the parameters needed for saving the video are included.</p>

Command	Description
<a href="#"><u>CheckLeds</u></a>	<p>Checks whether the three external LEDs (optionally provided with the camera) are functioning.</p> <p><b>To utilize this command a separate control board is required.</b></p> <p>This command is not supported in IDM-500 cameras.</p>
<a href="#"><u>CloseCamera</u></a>	<p>Closes all connections to the selected camera and releases all buffers used for video capture.</p>
<a href="#"><u>DebugPrint</u></a>	<p>Prints a message to a debug window, if open, while in DEBUG mode.</p>
<a href="#"><u>DisplayFrameNumTime</u></a>	<p>Displays the current frame and time on the frames output from camera.</p> <p>This command is not supported in IDM-500 cameras.</p>
<a href="#"><u>EnterSnapShotMode</u></a>	<p>Enters or leaves Snapshot Mode. When in Snapshot Mode the camera stops capturing and streaming video until a trigger is set.</p> <p>When entering or leaving Snapshot Mode, the FPS calculations start from 0.0 Fps.</p>
<a href="#"><u>FillCameraList</u></a>	<p>Searches for Camelot cameras connected to your PC USB port or on the LAN.</p> <p>After calling this command, use the pointer to the camera (pCam in the CAMERA_CAP_API structure) to access the API commands.</p>

Command	Description
<a href="#"><u>FillCameraListEx</u></a>	<p>Same as FillCameraList (above) with additional parameters:</p> <ul style="list-style-type: none"> <li>• specify whether USB or IP cameras are being searched</li> <li>• whether a camera with a specific serial number is requested</li> </ul> <p>For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</p>
<a href="#"><u>GetCamCaps</u></a>	Returns a structure containing some of the camera's capabilities.
<a href="#"><u>GetCameraVersionInfo</u></a>	Gets the camera's version numbers – HW, FW see also <b>Output</b> parameters.
<a href="#"><u>GetCapBinning</u></a>	Returns whether binning has been selected.
<a href="#"><u>GetCaptureResolution</u></a>	Returns the current camera output resolution.
<a href="#"><u>GetExposureTime</u></a>	Gets the actual exposure time (in microseconds) for each frame according to the <b>ShutterWidth</b> and <b>ShutterDelay</b> and capture resolution.
<a href="#"><u>GetFlipHorizontal</u></a>	Returns whether the picture is flipped horizontally.
<a href="#"><u>GetFlipVertical</u></a>	Returns whether the picture is flipped vertically.
<a href="#"><u>GetFPS</u></a>	Outputs the frames per second, FPS, of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning.

Command	Description
<a href="#"><u>GetFPSrate</u></a>	Outputs the Bit Rate of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning.
<a href="#"><u>GetFrameNum</u></a>	Outputs the current frame number being captured from the camera. This is a sequential number that starts at zero (0) every time streaming starts
<a href="#"><u>GetFrameWithFlash</u></a>	While streaming video, two frames are captured with flash (in the first the flash may be weaker) and streaming continues. The flash must be defined using the PWM controls, and EnableTimers set to 0 (flash for frame) See SetPwmTimer.
<a href="#"><u>GetGainType</u></a>	Returns with the value of the gain according to the camera type.
<a href="#"><u>GetGPIO</u></a>	Gets the current state and values of up to 9 GPIO (general purpose input/output) pins that are reserved for end user use.
<a href="#"><u>GetGPIO Pins</u></a>	Gets the current values of the up to 20 GPIO, general purpose input/output, pins that are reserved for application use. Each bit represents a pin and can either be high (1) or low (0).

Command		Description		
<a href="#"><u>SetCrosshair</u></a> / CamSetCrosshair		Returns whether the picture is shown in greyscale.		
<b>Description</b>	Puts a green crosshair in the middle of the screen. This is centering camera on specific points of an object.			
<b>Syntax</b>	SetCrosshair( <b>bool</b> crosshair) CamSetCrosshair( <b>int</b> nCamNum, <b>bool</b> crosshair)			
<b>Parameters</b>				
<b>Input</b>	nCamNum			Camera number
	crosshair			<b>True</b> – displays green crosshair <b>False</b> – no crosshair in display
<b>Output</b>	<b>None</b>			
GetGreyScale				
<a href="#"><u>GetIpConfig</u></a>		Gets IP configuration according to the NET_CONFIG_INFO structure above.		
<a href="#"><u>GetIpDefaultConfig</u></a>		Get IP default configurations – similar to GetIPConfig – but refers to default configurations.		
<a href="#"><u>GetLogs</u></a>		Gets log messages saved in the camera for debugging purposes. These messages are sent to a separate <a href="#"><u>DebugPrint</u></a> screen (if opened) and can be used to debug applications. See Using DebugPrint <a href="#"><u>on page 60</u></a> . <b>This command is not supported in IDM-500 cameras.</b>		
<a href="#"><u>GetMonochrome</u></a>		Returns whether sensor outputs monochrome (greyscale) data.		
<a href="#"><u>GetPixelBits</u></a>		Returns how many bits per pixel are being captured by camera.		
<a href="#"><u>GetPIIRate</u></a>		Gets the camera's PLL (internal clock) rate. <b>This option is only supported by the 5Mp models.</b>		
<a href="#"><u>GetRegion</u></a>		Gets the coordinates of the region of interest, ROI, programmed in the registers of the camera.		

Command	Description
<a href="#"><u>GetRegVal</u></a>	Returns value of specified register. Register addresses and values are in HEX notation. If fails returns -1.
<a href="#"><u>GetRegValues</u></a>	Outputs register values to a <b>REG_LIST_API</b> structure as defined in <b>hostCamApi.h</b> .
<a href="#"><u>GetResolution</u></a>	Returns the current resolution of the Preview Screen.
<a href="#"><u>GetSensorType</u></a>	Returns the type of sensor present in the camera. The type may be 1.3, 3, 5, 10 Megapixel, VGA or WVGA sensor according to Table 1 below.
<a href="#"><u>GetShutterDelay</u></a>	Returns the sensor shutter delay. This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker.
<a href="#"><u>GetShutterWidth</u></a>	Returns the sensor's shutter width. This value controls the time of exposure for each new frame captured
<a href="#"><u>GetShutterWidthGains</u></a>	Gets the sensor's current shutter width and color gain values.
<a href="#"><u>GetSnapShot</u></a>	Imitates the external trigger and instructs the camera to capture one frame and send it to the PC.
<a href="#"><u>GetSnapShotMode</u></a>	Returns whether camera is in Snapshot Mode.
<a href="#"><u>GetState</u></a>	Returns the current state of the camera
<a href="#"><u>GetViewWindow</u></a>	Returns the HANDLE of the preview window which was set in the Cam/SetViewWindow command.
<a href="#"><u>OpenCamera</u></a>	Camera streaming is stopped, the sensor is reset and the default parameters are loaded. The camera sensor is reset to default values as if powered up.

Command	Description
<a href="#"><u>Pause</u></a>	Pause the preview of the camera output. Streaming continues, and callback commands, if set, are accessible.
<a href="#"><u>PC_AWB</u></a>	Request the camera to perform the Automatic White Balance, AWB, algorithm on the frame on the PC. The new, calculated gain values will be sent to the camera.
<a href="#"><u>ResetCamera</u></a>	Closes all connections to the selected camera, releases all buffers used for video capture and resets the camera.
<a href="#"><u>Run</u></a>	When called, video is streamed from the camera. If a window has been defined (in the SetVideoWindow command), the video will be displayed.
<a href="#"><u>SaveRawFrameToFileBuf</u></a>	Saves a captured Raw image to a file in RAW, BMP or JPG format. This function can only be used within the RawFrame or RawFullFrame callbacks.
<a href="#"><u>SetAGC_AWB</u></a>	Sets if a camera should perform Automatic Gain Control, AGC, and/or Automatic White Balance, AWB, on data. Each of these commands dynamically changes gain values and shutter width. When AGC and AWB are disabled, the last, calculated, shutter width and gain values remain. <b>This command is not supported in IDM-500 cameras.</b>
<a href="#"><u>SetBadFrameDetection</u></a>	Specifies whether the camera should detect and correct Bad Frames.

Command	Description
<a href="#"><u>SetCapBinning</u></a>	Sets whether binning will be used by the sensor for video output. Only available if HALF or QUARTER resolution is selected.
<a href="#"><u>SetCaptureResolution</u></a>	Sets the camera output resolution. Video streaming is stopped before resolution is set.
<a href="#"><u>SetColorCorr</u></a>	Sets the type of color correction to be implemented on the PC for all incoming frames. If using Color Correction, default values can be found in the cc.ini file.
<a href="#"><u>SetDataCBPtr</u></a>	Sets the callback function that is called every time a new frame is received, if specified.
<a href="#"><u>SetExposureTime</u></a>	Sets the exposure time, in microseconds, for each frame. If the value entered is lower than the minimum value for this camera, the camera operates at the minimum value.
<a href="#"><u>SetFixBadPixels</u></a>	<p>Sets whether "bad" pixels should be fixed. "Bad pixels" are defined by one or two wrong colored pixels in an area of a known hue/color. The "bad pixels" originate from the sensor and are fixed on the PC.</p> <p>Only if width and height are at least 4 can this be performed. The "bad pixels" are fixed in stages, so it takes a few seconds to fix all of them on frames of 5Mp.</p>
<a href="#"><u>SetFlipHorizontal</u></a>	Sets whether the picture should be flipped horizontally.
<a href="#"><u>SetFlipVertical</u></a>	Sets whether the picture should be flipped vertically.

Command	Description
<a href="#"><u>SetFrameRate</u></a>	Sets the frame rate of the sensor output. The vertical blanking register is set so that the frame rate is controlled. If the frame rate given is higher than the sensor can deliver the sensor output is limited to the maximum output possible. If the requested frame rate is lower than the vertical blanking can control, then the frame rate will be as low as the highest vertical blanking value allows.
<a href="#"><u>SetGainProportions</u></a>	<p>The three R G B parameters are set as ratios between the gains – not by absolute values. When the R gain is changed, G and B are changed proportionately.</p> <p>Until a subsequent call, the gains will have the ratio  <math>1024:((G*1024)/R):((B*1024)/R)</math>. Integer math is used in the scaling.</p>
<a href="#"><u>SetGainType</u></a>	Sets sensor's gain according to the type specified.

Command	Description
<a href="#"><u>SetGetRawFullData</u></a>	<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 <a href="#"><u>SetPlIRate</u></a> command) before requesting the RAW frame.</p> <p><b>This command is not supported by the IDM-500 cameras. Explanation of the PLL Rate for 5Mp cameras</b></p> <p>We recommend using as low a PLL rate as possible for the the frame rate required. The fewer frames the sensor captures, the more time the DSP has for processing them. Since the USB bandwidth doesn't allow for more than 8.5 fps, there is no reason for the sensor to output 14fps.</p>
<a href="#"><u>SetGPIO</u></a>	Sets the current state and values of up to 9 GPIO (general purpose input/output) pins that are reserved for end user use.
<a href="#"><u>SetGPIO Pins</u></a>	Sets the current values of the up to 20 GPIO, general purpose input/output, pins that are reserved for application use. Each bit represents a pin and can either be high (1) or low(0).
<a href="#"><u>SetGreyScale</u></a>	Sets whether the picture should be shown using greyscale.
<a href="#"><u>SetIpConfig</u></a>	Sets IP configuration according to the NET_CONFIG_INFO structure above.
<a href="#"><u>SetIpDefaultConfig</u></a>	Set IP default configurations – similar to SetIPConfig – but refers to default configurations.

Command	Description
<a href="#"><u>SetLowPowerMode</u></a>	Specifies whether the camera should enter Low Power mode. If video is now streaming the camera will enter Low Power mode when streaming stops. <b>This command is not supported in any of the Camelot models or IDM-500 cameras.</b>
<a href="#"><u>SetLUT</u></a>	Can be set only after a LUT is successfully uploaded. Indicates to camera that all data should be translated with the uploaded LUT (Look up Table). <b>This command is not supported in IDM-500 cameras.</b>
<a href="#"><u>SetMonochrome</u></a>	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).
<a href="#"><u>SetMsgCBPtr</u></a>	Sets pointer to an error message callback function. The dll functions FillCameraList, and AccessCamera also send this pointer.
<a href="#"><u>SetPixelBits</u></a>	Defines how many bits per pixel will be captured by camera. This must be set when camera is initialized but not running. Default if not set is 8 bits per pixel.
<a href="#"><u>SetPIIRate</u></a>	Sets the camera's PLL (internal clock) rate. Video streaming is stopped before PLL is set. <b>Default: 60MHz.</b> <b>This option is only supported by the 5Mp models.</b>
<a href="#"><u>SetPortRange</u></a>	Set range of ports to be used by IP application.

Command	Description
<a href="#"><u>SetPwmTimer</u></a>	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 high. The width must be less or equal to the period for each timer. The timers (TMR1, TMR2 and TMR3) are started simultaneously and therefore synchronized. <b>This command is not supported in IDM-500 cameras.</b>
<a href="#"><u>SetRawFrameDataCB</u></a>	Sets whether a callback function should be called for each raw frame received.
<a href="#"><u>SetRawFullFrameCB</u></a>	Sets whether a callback function should be called for each raw Full frame received (as a result of the <a href="#"><u>SetGetRawFullData</u></a> function on page 41).
<a href="#"><u>SetRegion</u></a>	Sets the region of interest, ROI, to be output by the camera. The width and height must be even numbers. Video streaming is stopped before region is set.
<a href="#"><u>SetRegVal</u></a>	Sets specified register value.
<a href="#"><u>SetRegValues</u></a>	Sets register values according to list supplied in structure. Register addresses and values are in HEX notation.
<a href="#"><u>SetResolution</u></a>	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.

Command	Description
<a href="#"><u>SetRGBFrameDataCB</u></a>	Sets whether a callback function should be called after each raw frame has been processed into an RGB frame.
<a href="#"><u>SetRotation</u></a>	Sets the angle in degrees (0, 90, 180 and 270) for rotating the image.
<a href="#"><u>SetShutterDelay</u></a>	Sets the sensor shutter delay This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker.
<a href="#"><u>SetShutterWidth</u></a>	Sets the sensor shutter width. This value controls the time of exposure for each new frame captured
<a href="#"><u>SetTestData</u></a>	Specifies whether the camera should output Test Data or captured video images. This option is provided for developers to check if the data received is actually the data sent and other sensor independent features. WVGA, 1.3Mp and 3Mp cameras have only 1 option – to choose a value that is output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED. The 5Mp and 10Mp cameras have other options for creating patterns.
<a href="#"><u>SetViewWindow</u></a>	Provides the dll with a HANDLE to a window to be used for previewing the video. If video preview is required, this command must be called before Run/CamRun.

Command	Description
<a href="#"><u>SingleFrameAWB</u></a>	Request the camera to perform the Automatic White Balance, AWB, algorithm on a single frame. AWB will be performed a specified number of times and then cease. This command is not supported in IDM-500 cameras.
<a href="#"><u>StartVideo</u></a>	Starts a camera that has already been initialized. When called, video is streamed from the camera and displayed in a separate <b>Preview</b> window. Initial camera settings can be sent as parameters.
<a href="#"><u>StartVideoStreaming</u></a>	Starts a video streaming from a camera that has already been initialized. When called, video is streamed from the camera and displayed (if requested) in a separate Preview Window. Many capture and preview settings can be sent as parameters using structs.
<a href="#"><u>StopCamera</u></a>	Stops a running camera but keeps it initialized.
<a href="#"><u>UpdateFW</u></a>	Updates the camera Firmware. Video streaming is stopped before firmware is updated.

Command	Description
<u>UploadLUT</u>	<p>Uploads a Look-up Table, LUT, for translating raw pixel values in the camera before being output. 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. After the LUT is successfully loaded, SetLUT may be called in order to start using the LUT for new data.</p> <p><b>Not all of the Camelot models support this command. This command is not supported in the IDM-500 series.</b></p>

## INITIALIZING AND ACCESSING THE CAMERA

This chapter describes the commands for initializing and accessing the camera. Examples of all the commands are provided in the sample application. The files **CamelotDll.h** and **hostCamApi.h** for C++, or **CamelotDll.cs** and **hostCamApi.cs** for C#, must be included in each project.

To use these commands, the dll library must be loaded and each command assigned a pointer for access. The code below is written in C++ but is similar for other languages.

◆ **To initialize and access the camera:**

- a. Load the dll library.

```
HINSTANCE hDll; // handle to Dll with commands for Camelot cameras
hDll = LoadLibrary(TEXT("Camelot.dll"));
```

- b. Get address for the **CamEnumCameras** command:

```
typedef int (__stdcall * FARPROC_0_PARAM)(void);
FARPROC_0_PARAM pfnEnumCameras;
pfnEnumCameras =
(FARPROC_0_PARAM)GetProcAddress(hDll, "CamEnumCameras");
int numCamFound = lpfnEnumCameras();
```

- c. After calling **CamEnumCameras()** (by calling its pointer `lpfnEnumCameras()`), the number of cameras found is returned and the dll maintains a list of cameras. Each camera can now be accessed by calling one of the Cam... commands with the camera number (zero based) as the first parameter. Pointers to each of the Cam... commands must be individually accessed (by calling `GetProcAddress`).

**OR**

- a. Get address for the **FillCameraList** command and use the pointer passed back in the **CAMERA\_CAP\_API** for accessing the camera's interface commands:

```
typedef int(__cdecl * FARPROC_2_PTR)(CAMERA_CAP_API *camArray,
funcMsgPtr funcMsgCb);

// pointer to exported command
FARPROC_2_PTR pfnFillCameraList;
pfnFillCameraList = FARPROC_2_PTR)GetProcAddress(hDll, "FillCameraList");
```

- b. Both **funcMsgPtr** **funcMsgCb** and **CAMERA\_CAP\_API** are defined in the **hostCamApi.h** file.

**funcMsgPtr** **funcMsgCb** – a pointer to a command that will be called on error. It can contain information and recovery code.

**CAMERA\_CAP\_API** is a structure for each camera with specific identifying information and a pointer to the camera (`pCam`). The pointer can be used to access the camera interface commands (see next chapter):

```
#define SENSOR_DESC_LEN 30
#define CAM_ID_LEN 20
```

```

typedef struct
{
    unsigned int SensorType;
    char Desc[SENSOR_DESC_LEN];
    unsigned int CommType; // USB or TCP - COMM_TYPE

    int fTransportStarted;
    int reserved;
    int Width; // maximum width
    int Height; // maximum height
    int ActiveStartX; // X column start
    int ActiveStartY; // Y row start
    char CameraID[CAM_ID_LEN]; // camera's unique serial number
    unsigned int pCam; // pointer to IBDRCamera interface
    char Color; // 1-color, 0-monochrome
    char ProductID[CAM_ID_LEN]; // ID product number
    char FirmwareVersion[CAM_ID_LEN]; // FW version burned when shipped
    char HardwareVersion[CAM_ID_LEN]; // HW version
} CAMERA_CAP_API;

```

## CamEnumCams

<b>Description</b>	Searches for all Camelot and IDM-500 cameras connected to your PC USB port or on the LAN. After calling this command the cameras found can be accessed using the Cam... version of the API commands. The camera number is also provided, zero-based, as the first parameter.		
<b>Syntax</b>	<code>int CamEnumCams (void)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>None</b>	
<b>Return Values</b>	<b>int</b>		number of cameras found

## CamEnumCamsEx

<b>Description</b>	Same as CamEnumCams (above) with additional parameters: <ul style="list-style-type: none"> <li>• specify whether USB or IP cameras are being searched</li> <li>• whether a camera with a specific serial number is requested</li> <li>• For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</li> </ul>		
<b>Syntax</b>	<code>int CamEnumCamerasEx (TRANSPORT_TYPE TransportType, char *SerialNumber, char *szBroadcastIP, char *szGateway, int CamerasPerIP,)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>TransportType</b>	TT_USB or TT_IP
		<b>SerialNumber</b>	Specify serial number of camera to be found
		<b>szBroadcastIP</b>	For IP cameras, address for Broadcast message used for searching cameras on LAN
		<b>szGateway</b>	For IP cameras, gateway used for searching cameras on LAN
		<b>CamerasPerIP</b>	How many cameras are allowed with the same IP address (should be 1, unless special proprietary board)

		is used)
<b>Return Values</b>	<b>int</b>	number of cameras found

## FillCameraList

<b>Description</b>	Searches for Camelot cameras connected to your PC USB port or on the LAN. After calling this command, use the pointer to the camera (pCam in the CAMERA_CAP_API structure) to access the API commands.		
<b>Syntax</b>	<code>int FillCameraList(CAMERA_CAP_API *camArray, funcMsgPtr funcMsgCb)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>camArray</b>	pointer to an empty array of MAX_CAM_NUM elements. See above for build of CAMERA_CAP_API structure. Array elements will be filled in for each camera found.
		<b>funcMsgCb</b>	pointer to a Callback command with Error messages that will be called from the dll if an error occurs. Recovery code can be inserted if needed.
	<b>Output</b>		
<b>Return Values</b>	<b>int</b>		number of cameras found

## FillCameraListEx

<b>Description</b>	Same as FillCameraList (above) with additional parameters: <ul style="list-style-type: none"> <li>• specify whether USB or IP cameras are being searched</li> <li>• whether a camera with a specific serial number is requested</li> <li>• For IP cameras, the IP address from which the IP broadcast message is sent can be specified. If not specified, Broadcast message is sent from INADDR_BROADCAST (255.255.255.255).</li> </ul>		
<b>Syntax</b>	<code>int FillCameraListEx(TRANSPORT_TYPE TransportType, char *SerialNumber, char *szBroadcastIP, char *szGateway, CAMERA_CAP_API *camArray, int CamerasPerIP, funcMsgPtr funcMsgCb)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>TransportType</b>	TT_USB or TT_IP
		<b>SerialNumber</b>	Specify serial number of camera to be found
		<b>szBroadcastIP</b>	For IP cameras, address for Broadcast message used for searching cameras on LAN
		<b>szGateway</b>	For IP cameras, specific Gateway used for searching cameras on LAN
		<b>camArray</b>	pointer to an empty array of MAX_CAM_NUM elements. See above for build of CAMERA_CAP_API structure. Array elements will be filled in for each camera found.
		<b>CamerasPerIP</b>	How many cameras are allowed with the same IP address (should be 1, unless special proprietary board is used)
		<b>funcMsgCb</b>	pointer to a Callback command with Error messages

			that will be called from the dll if an error occurs. Recovery code can be inserted if needed.
	<b>Output</b>		
<b>Return Values</b>	<b>int</b>		number of cameras found

Chapter 4

## Controlling the Camera

There are two ways of accessing a camera and sending it commands:

1. Calling **CamEnumCams**, described in the previous chapter. Getting the number of cameras and then accessing each of the Cam... commands individually using the camera number (zero based) as the first parameter. In order to do this, all the pointers to the dll commands should be accessed using the `GetProcAddress` command. This method is demonstrated in the SimpleCamelot sample application.
2. Calling **FillCameraList** (described in previous chapter) with a pointer to an empty array of `MAX_CAM_NUM` cameras the default of 30 is defined in `hostCamApi.h`.

The command will return as many cameras found and the array updated with all camera information as well as a pointer to each camera. The pointer should be shown as described below:

```
int numCamFound = lpfnFillCameraList(g_CameraCap,
MessageCallback);
piBDR[camNum] = (IBDRCamera *)g_CameraCap[camNum].pCam;
```

Interface commands should be accessed by:

```
piBDR[camNum]->Command(parameters);
```

**Allmost all of these function calls are demonstrated in the CamelotView Sample Application.**

## Getting Camera Parameters

### GetSensorType / CamGetSensorType

<b>Description</b>	Returns the type of sensor present in the camera. The type may be 1, 3, 5, 10 Megapixel, VGA or WVGA sensor according to Table 1 below.											
<b>Syntax</b>	<pre>GetSensorType (int *pnSensorType) CamGetSensorType (int nCamNum, int *pnSensorType)</pre>											
<b>Parameters</b>												
	<b>Input</b>	<code>nCamNum</code>	Number of camera									
	<b>Output</b>	<code>pnSensorType</code>	<table border="0"> <tr> <td>SENSOR_UNKNOWN</td> <td>0</td> <td></td> </tr> <tr> <td>SENSOR_WVGA</td> <td>3</td> <td>WVGA</td> </tr> <tr> <td>SENSOR_OVT_VGA</td> <td>4</td> <td>Omni VGA sensor (IDM-500)</td> </tr> </table>	SENSOR_UNKNOWN	0		SENSOR_WVGA	3	WVGA	SENSOR_OVT_VGA	4	Omni VGA sensor (IDM-500)
SENSOR_UNKNOWN	0											
SENSOR_WVGA	3	WVGA										
SENSOR_OVT_VGA	4	Omni VGA sensor (IDM-500)										

			SENSOR_1300	13	1.3 Megapixel
			SENSOR_3000	30	3 Megapixel
			SENSOR_5000	50	5 Megapixel
			SENSOR_10000	100	10 Megapixel

**Table 1: Types of Cameras and their Resolutions**

Resolution	Width	Height	Total Raw Pixels	Y Row Start	X Column Start
<b>VGA</b>					
Full	640	480	307,200		
Half	320	240	76,800		
Quarter	160	120	19,200		
<b>WVGA</b>					
Full	752	480	360,960	<b>4</b>	<b>1</b>
Half	376	240	90,240		
Quarter	188	120	22,560		
<b>1.3 Mp</b>					
Full	1280	1024	1,310,720	<b>12</b>	<b>20</b>
Half	640	512	327,680		
Quarter	320	256	81,920		
<b>3 Mp</b>					
Full	2048	1536	3,145,728	<b>20</b>	<b>32</b>
Half	1024	768	786,432		
Quarter	512	384	196,608		
<b>5 Mp</b>					
Full	2592	1944	5,038,848	<b>54</b>	<b>16</b>
Half	1296	972	1,259,712		
Quarter	648	486	314,928		
<b>10 Mp</b>					
Full	3664	2748	10,068,672	<b>8</b>	<b>112</b>
Half	1832	1374	2,517,168		
Quarter	916	687	629,292		

## GetCameraVersionInfo / CamGetCameraVersionInfo

<b>Description</b>	Gets the camera's version numbers – HW, FW see also <b>Output</b> parameters.		
<b>Syntax</b>	<pre>GetCameraVersionInfo(HW_FW_VERSION_API *versionInfo) CamGetCameraVersionInfo(int nCamNum, HW_FW_VERSION_API *versionInfo);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>versionInfo</b>	An HW_FW_VERSION_API structure defined as: unsigned int sensorVersion; unsigned short len; char versionInfo[MAX_VERSION_LEN];

			unsigned char ExtClk; // 24 Mhz - 96 Mhz
<b>Return Values</b>	<b>None</b>		

## GetCamCaps / CamGetCamCaps

<b>Description</b>	Returns a structure containing some of the camera's capabilities.		
<b>Syntax</b>	<pre>GetCamCaps(CAMERA_CAP_API *CameraCap) CamGetCamCaps(int nCamNum, CAMERA_CAP_API *CameraCap);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>CameraCap</b>	<p>CAMERA_CAP_API struct which is defined as:</p> <pre>unsigned int SensorType; // see GetSensorType char Desc[SENSOR_DESC_LEN]; unsigned int CommType; // USB or TCP - COMM_TYPE float FirmwareVersion; float HardwareVersion; int Width; // maximum width int Height; // maximum height int ActiveStartX; // X column start int ActiveStartY; // Y row start char CameraID[CAM_ID_LEN]; // serial number unsigned int pCam; // pointer to camera interface char Color;</pre>

## OpenCamera / CamOpenCamera

<b>Description</b>	Camera streaming is stopped, the sensor is reset and the default parameters are loaded. The camera sensor is reset to default values as if powered up.		
<b>Syntax</b>	<pre>OpenCamera(void); CamOpenCamera(int nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

## GetState / CamGetState

<b>Description</b>	Returns the current state of the camera.		
<b>Syntax</b>	<pre>int GetState(void); int CamGetState(int nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>int</b>	State of camera:

			NO_INIT	0	// camera was closed, can't access
			INIT	1	// camera is found and ready
			RUNNING	2	// video is streaming
			PAUSE	3	// display is paused

## CamGetMaxRegion

<b>Description</b>	Gets the maximum coordinates of the region of interest, ROI, that the camera is capable of capturing. When accessing these commands through interface and camera's pointer, these values are in the CamCaps (CAMERA_CAP_API) struct.		
<b>Syntax</b>	<code>CamGetMaxRegion(int nCamNum, int * startX, int * startY, int *maxWidth, int *maxHeight)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*startX</b>	X coordinate value (column number) for region's left side
		<b>*startY</b>	Y coordinate (row number) for region's top row
		<b>*maxWidth</b>	Width in pixels from startX
		<b>*maxHeight</b>	Height in pixels from startY
	<b>Output</b>	<b>None</b>	

## Controlling the Video Streaming

### StartVideo / CamStartVideo

<b>Description</b>	Starts a camera that has already been initialized. When called, video is streamed from the camera and displayed in a separate <b>Preview</b> window. Initial camera settings can be sent as parameters.		
<b>Syntax</b>	<pre>StartVideo(int nCamNum, int nShutterWidth, int nGain,            int nResolution, int a_nX, int a_nY,            int a_nWidth, int a_nHeight, HWND hWindow); CamStartVideo(int nCamNum, int nShutterWidth, int nGain,               int nResolution, int a_nX, int a_nY,               int a_nWidth, int a_nHeight, HWND hWindow);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nShutterWidth</b>	Camera's shutter width
		<b>nGain</b>	Global gain
		<b>a_nX</b>	ROI - Start X
		<b>a_nY</b>	ROI - Start Y
		<b>a_nWidth</b>	ROI - Width of video
		<b>a_nHeight</b>	ROI - Height of Video
		<b>hWindow</b>	HANDLE to video window. If NULL, will create new window for video.
	<b>Output</b>	<b>None</b>	

<b>Return Values</b>	<b>true</b>	Succeeded
	<b>false</b>	Failed

## StartVideoStreaming / CamStartVideoStreaming

For this composite function to work, the following structures must be defined. All the parameters needed for viewing the video are included. Capture Info structure:

```
typedef struct
{
    RESOLUTION_TYPE Resolution; // Capture resolution - FULL, HALF,
                                QUARTER
    FOV_API ROI; // Capture ROI - x,y,width,height
    unsigned int Exposure; // Exposure time (in microseconds) for
                            capture
    unsigned int ShutterWidth; // Shutter width- if Exposure time = 0
    unsigned int Gain; // Gain for capture
    unsigned int NumFrames; // number of frames to capture,
                            0 = streaming
    unsigned char WithFlash; // should flash be on for capture
    unsigned char UseLUT; // should frame be translated using
                            LUT that was loaded
    unsigned char UseAgc; // should Automatic Gain Control be
                            used
    unsigned char UseAwb; // should Automatic White Balance be
                            used
    unsigned char AdjustFlicker; // 0 - don't adjust, 50 - for 50Hz, 60
                                - for 60Hz
    unsigned char PixelBits; // Bits per pixel - 8, 10, 12
    unsigned char PllRate ; // for 5Mp with 24Mhz crystal - values
                            between 24-96
    unsigned char UseTestData; // Use Test Data - default data for
                                each sensor
    double FrameRate; // rate in FPS
} tCaptureInfo;
```

Preview info structure:

```
typedef struct
{
    RESOLUTION_TYPE Resolution; // Capture resolution - FULL,
                                HALF, QUARTER
    unsigned char FlipHorizontal; // should data be flipped
                                horizontally?
    unsigned char FlipVertical; // should data be flipped
                                vertically?
    unsigned char DegreesRotate; // values of 0, 90, 180, 270 -
                                n.a.
    unsigned char DisableBayer; // RAW data is displayed as is -
                                good for monochrome sensors
    unsigned char Greyscale; // should frames be displayed in
                                greyscale?
    unsigned char UseHistogramEqualization; // use Histogram
                                equalization
}
```

```

    unsigned char    UseHistogramStretching;    // use Histogram
                                                    Stretching
    unsigned char    UseColorCorrectionMatrix; // use Color
                                                    Correction Matrix - uploaded or default
    HWND             hParentWindow;           // can be NULL
    HWND             hPreviewWindow;         // HANDLE to child window with
                                                    Preview. NULL for new
                                                    window to be created
    int              Width;                  // dimensions of window to open
    int              Height;                 // dimensions of window to open
    RECT             Rect;                   // where to place window on
                                                    screen
    unsigned char    FitToScreen;            // if hPreviewWindow is NULL -
                                                    0 - scroll bars, 1 - video
                                                    sized to screen
} tPreviewInfo;

```

<b>Description</b>	Starts a video streaming from a camera that has already been initialized. When called, video is streamed from the camera and displayed (if requested) in a separate Preview Window. Many capture and preview settings can be sent as parameters using structs.		
<b>Syntax</b>	<pre> StartVideoStreaming(tCaptureInfo *CaptureInfo,                    tPreviewInfo *PreviewInfo); CamStartVideoStreaming(int nCamNum, tCaptureInfo *CaptureInfo,                        tPreviewInfo *PreviewInfo); </pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>CaptureInfo</b>	Pointer to structure with capture information
		<b>PreviewInfo</b>	Pointer to structure with preview information
	<b>Output</b>	<b>None</b>	
<b>Return Values</b>	<b>true</b>	Succeeded	
	<b>false</b>	Failed	

## SetViewWindow / CamSetViewWindow

<b>Description</b>	Provides the dll with a HANDLE to a window to be used for previewing the video. If video preview is required, this command must be called before Run/CamRun.		
<b>Syntax</b>	<pre> SetViewWindow(HWND hWnd, int a_camNum, int width,               int height) CamSetViewWindow(int nCamNum, HWND hWnd, int a_nWidth,                  int a_nHeight) </pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>hWindow</b>	HWND handle to the window to be used for preview
		<b>nCamNum</b>	Camera number
		<b>width</b>	Initial width of video to be displayed. Should be calculated according to capture and preview resolutions.
		<b>height</b>	Initial height of video to be displayed. Should be calculated according to capture and preview resolutions.

	<b>Output</b>	<b>None</b>	
--	---------------	-------------	--

## GetViewWindow / CamGetViewWindow

<b>Description</b>	Returns the HANDLE of the preview window which was set in the Cam/SetViewWindow command.		
<b>Syntax</b>	<pre>int GetViewWindow(void); int CamGetViewWindow(void);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>none</b>	
	<b>Output</b>	<b>int</b>	Handle to the preview window in which video is displayed

## SetPixelBits / CamSetPixelBits

<b>Description</b>	Defines how many bits per pixel will be captured by camera. This must be set when camera is initialized but not running. The default is 8 bits per pixel.		
<b>Syntax</b>	<pre>SetPixelBits(int BitsPerPixel) CamSetPixelBits(int nCamNum, int BitsPerPixel)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>BitsPerPixel</b>	8 is default. 10 or 12 (for 5Mp and 10Mp) are also options.
	<b>Output</b>	<b>None</b>	

## GetPixelBits / CamGetPixelBits

<b>Description</b>	Returns how many bits per pixel are being captured by camera.		
<b>Syntax</b>	<pre>SetPixelBits(int *BitsPerPixel) CamSetPixelBits(int nCamNum, int *BitsPerPixel)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>BitsPerPixel</b>	8, 10, or 12

## Run / CamRun

<b>Description</b>	When called, video is streamed from the camera. If a window has been defined (in the <a href="#">SetVideoWindow</a> command), the video will be displayed.		
<b>Syntax</b>	<pre>Run(void) CamRun(int a_nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

<b>Return Values</b>	<b>true</b>	Succeeded
	<b>false</b>	Failed

## Pause / CamPause

<b>Description</b>	Pause the preview of the camera output. Streaming continues, and callback commands, if set, are accessible.		
<b>Syntax</b>	<pre>Pause () CamPause (int a_nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>a_nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	
<b>Return Values</b>	<b>None</b>		

## StopCamera / CamStopCamera

<b>Description</b>	Stops a running camera but keeps it initialized.		
<b>Syntax</b>	<pre>StopCamera () CamStopCamera (int a_nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

## CloseCamera / CamCloseCamera

<b>Description</b>	Closes all connections to the selected camera and releases all buffers used for video capture.		
<b>Syntax</b>	<code>CloseCamera ()</code> <code>CamCloseCamera (int nCamNum)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

## ResetCamera / CamResetCamera

<b>Description</b>	Closes all connections to the selected camera, releases all buffers used for video capture and resets the camera.		
<b>Syntax</b>	<code>ResetCamera ()</code> <code>CamResetCamera (int nCamNum)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

## GetFrameNum / CamGetFrameNum

<b>Description</b>	Outputs the current frame number being captured from the camera. This is a sequential number that starts at zero (0) every time streaming starts.		
<b>Syntax</b>	<code>GetFrameNum (void)</code> <code>CamGetFrameNum (int nCamNum)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>Frame number</b>	Current frame number captured. Frame numbers are given to each frame when captured by the camera. Not all frames are transmitted to the PC (bandwidth limits), so Frame numbers may not be consecutive. These Frame numbers may be used to calculate the camera's FPS.

## DisplayFrameNumTime / CamDisplayFrameNumTime

<b>Description</b>	Displays the current frame and time on the frames output from camera. This command is not supported in IDM-500 cameras.		
<b>Syntax</b>	<pre>DisplayFrameNumTime (void) CamDisplayFrameNumTime (int nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	None	

## GetFPS / CamGetFPS

<b>Description</b>	Outputs the frames per second, FPS, of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning.		
<b>Syntax</b>	<pre>GetFPS (double *pFPS, double *pSkipPS, double *pDisplayPS) CamGetFPS (int nCamNum, double *pFPS, double *pSkipPS, double *pDisplayPS)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pFPS</b>	How many FPS captured by PC.
		<b>pSkipPS</b>	n.a. (should be NULL)
		<b>pDisplayPS</b>	How many FPS displayed on PC Preview screen

## GetFPSrate / CamGetFPSrate

<b>Description</b>	Outputs the Bit Rate of the selected camera. This is a dynamically changing statistic and depends on camera resolution, shutter width, gain and binning.		
<b>Syntax</b>	<pre>GetFPSrate (double *pFPS, double *pDisplayPS) CamGetFPSrate (int nCamNum, double *pFPS, double *pDisplayPS)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pFPS</b>	How many MegaBits/sec captured by DirectShow filter
		<b>pDisplayPS</b>	How many MegaBits/sec displayed on PC Preview screen

## 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.**

### SetGainType / CamSetGainType

<b>Description</b>	Sets sensor's gain according to the type specified.		
<b>Syntax</b>	<pre>SetGainType(int type, int nGain) CamSetGainType(int nCamNum, int type, int nGain)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>Type</b>	RED 0 GREEN 1 BLUE 2 GLOBAL 3 // sets R, G, and B gains to same value PROPORTIONAL 4 // gain is for RED, sets others proportionally
		<b>nGain</b>	Value of gain – range 0 – 1024. See above for short explanation of analog/digital gains.
	<b>Output</b>	<b>None</b>	

### GetGainType / CamGetGainType

<b>Description</b>	Returns with the value of the gain according to the camera type.		
<b>Syntax</b>	<pre>GetGainType(int type, int *pnGain) CamGetGainType(int nCamNum, int type, int *pnGain)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>Type</b>	RED 0 GREEN 1 BLUE 2
	<b>Output</b>	<b>pnGain</b>	Returned gain in range 0 - 1024

## SetGainProportions / CamSetGainProportions

<b>Description</b>	<p>The three R G B parameters are set as ratios between the gains – not by absolute values. When the R gain is changed, G and B are changed proportionately.</p> <p>Until a subsequent call, the gains will have the ratio <math>1024:((G*1024)/R):((B*1024)/R)</math>. Integer math is used in the scaling.</p>	
<b>Syntax</b>	<pre>SetGainProportions (int R, int G, int B) SetGainProportions (int nCamNum, int R, int G, int B)</pre>	
<b>Parameters</b>		
<b>Input</b>	nCamNum	Camera number
	R	A number representing weight of RED gain
	G	A number representing weight of GREEN gain
	B	A number representing weight of BLUE gain
<b>Output</b>	pnGain	Returned gain in range 0 - 1024

## Exposure Time and Shutter Control

### Exposure Time

The Exposure time is the reset time of each pixel row subtracted from the sample time. This is the amount of time required before a new row is available. Exposure time is a function of the camera's external **PIXCLK** (pixel clock), PLL rate, shutter width, shutter delay, frame width and binning. Exposure time is calculated per frame, and is displayed on the **Main** screen. Except for the WVGA sensor, described below, all the Camelot and IDM-500 series cameras use a rolling shutter. Rolling shutters cannot freeze moving objects as well as global shutters can.

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

### Shutter Control (WVGA -Wide VGA Sensor - 752 x 480 pixels)

The global shutter feature of the WVGA image sensor is able to freeze moving objects because all pixels are exposed simultaneously. When using a global shutter all pixels begin exposure, integrating a charge, simultaneously and end exposure simultaneously. A new sequence only begins after the readout of all pixels is completed.

## GetShutterWidth / CamGetShutterWidth

<b>Description</b>	Returns the sensor's shutter width. This value controls the time of exposure for each new frame captured		
<b>Syntax</b>	<pre>GetShutterWidth (int *pnShutterWidth) CamGetShutterWidth (int nCamNum, int *nShutterWidth)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
<b>Output</b>	<b>pnShutterWidth</b>	Returned shutter width – values are in range from 1 – 65,500	

## SetShutterWidth / CamSetShutterWidth

<b>Description</b>	Sets the sensor shutter width. This value controls the time of exposure for each new frame captured		
<b>Syntax</b>	<pre>SetShutterWidth (int nShutterWidth) CamSetShutterWidth (int nCamNum, int nShutterWidth)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>nShutterWidth</b>	Valid values from 1-65,500	
<b>Output</b>	<b>None</b>		

## GetShutterWidthGains / CamGetShutterWidthGains

<b>Description</b>	Gets the sensor's current shutter width and color gain values.		
<b>Syntax</b>	<pre>GetShutterWidthGains (SHUTTER_GAIN_API *pnShutterGains) CamGetShutterWidthGains (int nCamNum, SHUTTER_GAIN_API *pnShutterGains)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pnShutterGains</b>	SHUTTER_GAIN_API struct which is defined as: <pre>int shutterWidth; int exposureTime; // in microseconds int redGain; int blueGain; int greenGain; int globalGain;</pre>

## GetShutterDelay / CamGetShutterDelay

<b>Description</b>	Returns the sensor shutter delay. This value is used for controlling exposure time and to avoid any 50Hz or 60Hz flicker.		
<b>Syntax</b>	<pre>GetShutterDelay (int *pnShutterDelay) CamGetShutterDelay (int nCamNum, int *pnShutterDelay)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pnShutterDelay</b>	Returned shutter delay – values are in range from 0 – 2,047

## SetShutterDelay / CamSetShutterDelay

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

## AdjustFlicker / CamAdjustFlicker

<b>Description</b>	Adjusts the Shutter Width to avoid the 50Hz or 60Hz flicker experienced when using indoor lighting.		
<b>Syntax</b>	<code>AdjustFlicker (bool adjust, int freqHz)</code> <code>CamAdjustFlicker (int nCamNum, bool adjust, int freqHz)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>adjust</b>	<b>True</b> – turns on this feature – whenever Shutter Width is modified, it will be changed to adjust the flicker <b>False</b> – no adjustment made for flicker
		<b>freqHz</b>	For what frequency should the flicker be adjusted? Values: 50, 60
	<b>Output</b>	<b>None</b>	

## SetExposureTime / CamSetExposureTime

<b>Description</b>	Sets the exposure time, in microseconds, for each frame. If the value entered is lower than the minimum value for this camera, the camera operates at the minimum value.		
<b>Syntax</b>	<code>SetExposureTime (int expTime)</code> <code>CamSetExposureTime (int nCamNum, int expTime)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>expTime</b>	Number of microseconds for exposing each frame
	<b>Output</b>	<b>none</b>	

## SetFrameRate / CamSetFrameRate

<b>Description</b>	Sets the frame rate of the sensor output. The vertical blanking register is set so that the frame rate is controlled. If the frame rate given is higher than the sensor can deliver the sensor output is limited to the maximum output possible. If the requested frame rate is lower than the vertical blanking can control, then the frame rate will be as low as the highest vertical blanking value allows.		
<b>Syntax</b>	<code>SetFrameRate (float frameRate)</code> <code>CamSetFrameRate (int nCamNum, float frameRate)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>frameRate</b>	Number of frames to be output by the sensor per second
	<b>Output</b>	<b>none</b>	

## GetExposureTime / CamGetExposureTime

<b>Description</b>	Gets the actual exposure time (in microseconds) for each frame according to the <b>ShutterWidth</b> , <b>ShutterDelay</b> , and capture resolution.		
<b>Syntax</b>	<pre>GetExposureTime (int *expTime) CamGetExposureTime (int nCamNum, int *expTime)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Input</b>	<b>expTime</b>	Number of microseconds each frame is exposed
	<b>Output</b>	<b>none</b>	

## CalcShutWidthForExpTime / CamCalcShutWidthForExpTime

<b>Description</b>	Returns the shutter width that should be programmed in order to get the exposure time (in microseconds) indicated.		
<b>Syntax</b>	<pre>CalcShutWidthForExpTime (int expTime, int *shutWidth) CamCalcShutWidthForExpTime (int nCamNum, int expTime, int *shutWidth)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Input</b>	<b>expTime</b>	Number of microseconds to expose each frame
	<b>Output</b>	<b>shutWidth</b>	Value of shutter width to program in order to get desired Exposure time

## 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 this means that only 1/4 of the pixels coming from the camera are displayed, since every alternate row and column are skipped.
  - c. **QUARTER** - Only 1/16 of the pixels received from the camera are displayed.

## GetResolution / CamGetResolution

<b>Description</b>	Returns the current resolution of the Preview Screen.		
<b>Syntax</b>	<pre>GetResolution(int *pnResolution) CamGetResolution(int nCamNum, int *pnResolution)</pre>		
<b>Parameters</b>			
	<b>Input</b>	nCamNum	Camera number
	<b>Output</b>	pnResolution	<pre>FULL    = 0, HALF    = 1, QUARTER = 2, EIGHTH  = 3, // not yet supported</pre>

## SetResolution / CamSetResolution

<b>Description</b>	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.		
<b>Syntax</b>	<pre>SetResolution(int nResolution) CamSetResolution(int nCamNum, int nResolution)</pre>		
<b>Parameters</b>			
	<b>Input</b>	nCamNum	Camera number
		nResolution	<pre>FULL    = 0, HALF    = 1, QUARTER = 2, EIGHTH  = 3, // not yet supported</pre>
	<b>Output</b>	None	

## GetCaptureResolution / CamGetCaptureResolution

<b>Description</b>	Returns the current camera output resolution.		
<b>Syntax</b>	<pre>GetCaptureResolution(int *pnResolution) CamGetCaptureResolution(int nCamNum, int *pnResolution)</pre>		
<b>Parameters</b>			
	<b>Input</b>	nCamNum	Camera number
	<b>Output</b>	pnResolution	<pre>FULL    = 0, HALF    = 1, QUARTER = 2, EIGHTH  = 3, // not yet supported.</pre>

## SetCaptureResolution / CamSetCaptureResolution

<b>Description</b>	Sets the camera output resolution. Video streaming is stopped before resolution is set.		
<b>Syntax</b>	<pre>SetCaptureResolution(int nResolution) CamSetCaptureResolution(int nCamNum, int nResolution)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nResolution</b>	<pre>FULL    = 0, HALF    = 1, QUARTER = 2, EIGHTH  = 3, // not yet supported.</pre>
	<b>Output</b>	<b>None</b>	

## Setting a Region of Interest (ROI)/Frame of View (FOV)

A specific area or subset of the total available resolution may be defined as a Region of Interest, ROI. This ROI is captured and/or displayed according to the settings used. See [SetCaptureResolution](#) and/or [SetResolution](#) for more information. The SetRegion and GetRegion commands control the ROI requested from the camera.

## GetRegion / CamGetRegion

<b>Description</b>	Gets the coordinates of the region of interest, ROI, programmed in the registers of the camera.		
<b>Syntax</b>	<pre>GetRegion(int *startX, int *startY, int *width, int *height) CamGetRegion(int nCamNum, int *startX, int *startY, int *width, int *height)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*startX</b>	X coordinate value (column number) for region's left side
		<b>*startY</b>	Y coordinate (row number) for region's top row
		<b>*width</b>	Width in pixels from startX
		<b>*height</b>	Height in pixels from startY
	<b>Output</b>	<b>None</b>	

## SetRegion / CamSetRegion

<b>Description</b>	Sets the region of interest, ROI, to be output by the camera. The width and height must be even numbers. Video streaming is stopped before region is set.		
<b>Syntax</b>	<pre>SetRegion(int startX, int startY, int width, int height) CamSetRegion(int nCamNum, int startX, int startY, int width, int height)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>startX</b>	X coordinate value (column number) for region's left side must be an even value
		<b>startY</b>	Y coordinate (row number) for region's top row must be an even value
		<b>width</b>	Width in pixels from startX must be an even value
		<b>height</b>	Height in pixels from startY must be an even value
	<b>Output</b>	<b>None</b>	

## Binning

Binning can be set when rows and columns are skipped as in HALF and QUARTER resolutions. The rows/columns displayed are averaged with the skipped rows/columns. The picture should be a bit smoother but the FPS is similar to that of a FULL Resolution frame. This could mean a lower FPS value.

## SetCapBinning / CamSetCapBinning

<b>Description</b>	Sets whether binning will be used by the sensor for video output. Only available if HALF or QUARTER resolution is selected.		
<b>Syntax</b>	<pre>SetCapBinning(bool bBinning) CamSetCapBinning(int nCamNum, bool bBinning)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>bBinning</b>	<b>True</b> = use binning for video output <b>False</b> = don't use binning (default)
	<b>Output</b>	<b>None</b>	

## GetCapBinning / CamGetCapBinning

<b>Description</b>	Returns whether binning has been selected.		
<b>Syntax</b>	<code>SetCapBinning (bool *bBinning)</code> <code>CamSetCapBinning (int nCamNum, bool *bBinning)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>bBinning</b>	<b>True</b> = binning is used for video output <b>False</b> = binning is not being used

## Snapshot Mode

When the camera is in Snapshot mode frame output is stopped until an external trigger is set. Each time the trigger is tripped a snapshot is taken, and displayed in the SampleApplication video screen. Camelot cameras are available with an optional external trigger.

## EnterSnapShotMode / CamEnterSnapShotMode

<b>Description</b>	Enters or leaves Snapshot Mode. When in Snapshot Mode the camera stops capturing and streaming video until a trigger is set. When entering or leaving Snapshot Mode, the FPS calculations start from 0.0 Fps.		
<b>Syntax</b>	<code>EnterSnapShotMode (bool snapshotMode)</code> <code>CamEnterSnapShotMode (int nCamNum, bool snapshotMode)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>snapshotMode</b>	<b>True</b> = Snapshot mode is ON <b>False</b> = Snapshot mode is OFF
	<b>Output</b>	<b>None</b>	

## GetSnapShotMode / CamGetSnapShotMode

<b>Description</b>	Returns whether camera is in Snapshot Mode.		
<b>Syntax</b>	<code>CamGetSnapShotMode (bool *SnapMode)</code> <code>CamGetSnapShotMode (int nCamNum, bool *SnapMode)</code>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>SnapMode</b>	<b>True</b> = Snapshot mode is ON <b>False</b> = Snapshot mode is OFF

## GetSnapshot / CamGetSnapshot

<b>Description</b>	Imitates the external trigger and instructs the camera to capture one frame and send it to the PC.		
<b>Syntax</b>	<pre>GetSnapshot(<b>bool</b> withFlash) CamGetSnapshot(<b>int</b> nCamNum, <b>bool</b> withFlash)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>withFlash</b>	<b>True</b> = use the external flash (optionally supplied with camera) when snapshot is triggered <b>False</b> = don't use flash
	<b>Output</b>	<b>ret</b>	<b>True</b> = ACK received from camera <b>False</b> = no ACK received

## Capturing Frames to File

### CaptureFrame / CamCaptureFrame

<b>Description</b>	Captures the next frame from the camera and saves it as a RAW, BMP or JPEG image. If in Snapshot mode, will capture the next frame after the GetSnapshot() command is called.		
<b>Syntax</b>	<pre>CaptureFrame(<b>char*</b> a_sFileName, SAVE_FRAME_TYPE type, <b>bool</b> bmpQuality, <b>int</b> jpegQuality) CamCaptureFrame(<b>int</b> nCamNum, <b>char*</b> a_sFileName, SAVE_FRAME_TYPE type, <b>bool</b> bmpQuality, <b>int</b> jpegQuality)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>a_sFileName</b>	Valid file name – does not have to exist. If exists, is overwritten.
		<b>type</b>	Frame type to be saved according to table: RAW = 0, BMP = 1, JPG = 2
		<b>bmpQuality</b>	False – regular Bayer conversion True – Bayer High quality conversion. If RAW image, n.a.
		<b>jpegQuality</b>	If JPG image, quality from 0 (low) to 100 (high).
	<b>Output</b>	<b>None</b>	

## CaptureFrameDone / CamCaptureFrameDone

<b>Description</b>	Used to poll if frame was captured. After calling CaptureFrame. This function will return whether or not the frame was saved.		
<b>Syntax</b>	<pre>CaptureFrameDone (bool *done) CamCaptureFrameDone (int nCamNum, bool *done)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
<b>Output</b>	<b>done</b>	0 - frame not saved, 1 - frame is saved	
	<b>int</b>	1 -, 0 -	
<b>Return Values</b>	<b>FALSE - 0</b>	Frame not saved (yet)	
	<b>TRUE - 1</b>	Frame saved	

## CaptureFrameDoneEx / CamCaptureFrameDoneEx

<b>Description</b>	Returns TRUE when the amount of frames specified has been captured.		
<b>Syntax</b>	<pre>CaptureFrameDone (bool *done, int *pFrameCount) CamCaptureFrameDone (int nCamNum, bool *done, int *pFrameCount)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>pFrameCount</b>	How many frames should be captured before this command returns TRUE	
<b>Output</b>	<b>done</b>	0 - pFrameCount frames not yet saved, 1 - pFrameCount frame(s) saved	
	<b>int</b>	1 -, 0 -	
<b>Return Values</b>	<b>FALSE - 0</b>	Frame(s) not saved (yet)	
	<b>TRUE - 1</b>	Frame(s) saved	

## CaptureFrameFileBuf / CamCaptureFrameFileBuf

For this composite command to work, the following structures must be defined. All the parameters needed for saving the video are included.

Save Info structure:

```
typedef struct
{
    SAVE_FRAME_TYPE Type; // Frame type - RAW, BMP (RGB32), JPG
    unsigned char Quality; // BMP - >0 is high quality, JPG- 0-100
    char *Buffer; // if not NULL will save to buffer
    int BufferLen;
    char *FileName; // if not NULL, will save to file
} tSaveFrameInfo;
```

<b>Description</b>	Captures the next frame from the camera and saves it as a RAW, BMP or JPG image. This command can be called while in Snapshot mode or in video streaming mode.		
<b>Syntax</b>	<pre>CaptureFrameFileBuf(tSaveFrameInfo *SaveInfo); CamCaptureFrameFileBuf(int nCamNum, tSaveFrameInfo *SaveInfo);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>SaveInfo</b>	Parameters for saving file
	<b>Output</b>	<b>None</b>	

## GetFrameWithFlash / CamGetFrameWithFlash

<b>Description</b>	While streaming video, two frames are captured with flash (in the first the flash may be weaker) and streaming continues. The flash must be defined using the PWM controls, and <code>EnableTimers</code> set to 0 (flash for frame) See <a href="#">SetPwmTimer</a> .		
<b>Syntax</b>	<pre>GetFrameWithFlash() CamGetFrameWithFlash(int nCamNum)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>None</b>	

## Preview Controls

These commands control the way the video is displayed on the PC. They only affect the RGB image in the display, and are performed during or after Bayer conversion.

### GetFlipHorizontal / CamGetFlipHorizontal

<b>Description</b>	Returns whether the picture is flipped horizontally.		
<b>Syntax</b>	<pre>GetFlipHorizontal (bool *pbFlipHorizontal) CamGetFlipHorizontal (int nCamNum, bool *pbFlipHorizontal)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pbFlipHorizontal</b>	<b>True</b> = picture is flipped horizontally <b>False</b> = picture isn't flipped horizontally

### SetFlipHorizontal / CamSetFlipHorizontal

<b>Description</b>	Sets whether the picture should be flipped horizontally.		
<b>Syntax</b>	<pre>SetFlipHorizontal (bool bFlipHorizontal) CamSetFlipHorizontal (int nCamNum, bool bFlipHorizontal)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>bFlipHorizontal</b>	<b>True</b> = flip the picture horizontally <b>False</b> = don't flip the picture horizontally
	<b>Output</b>	<b>None</b>	

### GetFlipVertical / CamGetFlipVertical

<b>Description</b>	Returns whether the picture is flipped vertically.		
<b>Syntax</b>	<pre>GetFlipVertical (bool *pbFlipVertical) CamGetFlipVertical (int nCamNum, bool *pbFlipVertical)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pbFlipVertical</b>	<b>True</b> = picture is flipped vertically <b>False</b> = picture isn't flipped vertically

## SetFlipVertical / CamSetFlipVertical

<b>Description</b>	Sets whether the picture should be flipped vertically.		
<b>Syntax</b>	<pre>SetFlipVertical(<b>bool</b> bFlipVertical) CamSetFlipVertical(<b>int</b> nCamNum, <b>bool</b> bFlipVertical)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>bFlipVertical</b>	<b>True</b> = flip the picture vertically <b>False</b> = don't flip the picture vertically
	<b>Output</b>	<b>None</b>	

## SetRotation / CamSetRotation

<b>Description</b>	Sets the angle in degrees (0, 90, 180 and 270) for rotating the image. Other illegal values will default to Rotation of 0.		
<b>Syntax</b>	<pre>SetRotation(<b>int</b> nRotation) CamSetRotation(<b>int</b> nCamNum, <b>int</b> nRotation)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nRotation</b>	DON'T ROTATE = 0, 90, 180, 270
	<b>Output</b>	<b>None</b>	

## SetCrosshair / CamSetCrosshair

<b>Description</b>	Puts a green crosshair in the middle of the screen. This is helpful for focusing and centering camera on specific points of an object.		
<b>Syntax</b>	<pre>SetCrosshair(<b>bool</b> crosshair) CamSetCrosshair(<b>int</b> nCamNum, <b>bool</b> crosshair)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>crosshair</b>	<b>True</b> - displays green crosshair <b>False</b> - no crosshair in display
	<b>Output</b>	<b>None</b>	

## GetGreyScale / CamGetGreyScale

<b>Description</b>	Returns whether the picture is shown in greyscale.		
<b>Syntax</b>	<pre>GetGreyScale(<b>bool</b> *pbGreyScale) CamGetGreyScale(<b>int</b> nCamNum, <b>bool</b> *pbGreyScale)</pre>		

Parameters			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>pbGreyScale</b>	<b>True</b> = picture is in greyscale <b>False</b> = picture is shown in color

## SetGreyScale / CamSetGreyScale

<b>Description</b>	Sets whether the picture should be shown using greyscale.		
<b>Syntax</b>	<code>SetGreyScale (bool bGreyScale)</code> <code>CamSetGreyScale (int nCamNum, bool bGreyScale)</code>		
Parameters			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>bGreyScale</b>	<b>True</b> = show picture in greyscale <b>False</b> = show picture in color
	<b>Output</b>	<b>None</b>	

## SetMonochrome / CamSetMonochrome

<b>Description</b>	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).		
<b>Syntax</b>	<code>SetMonochrome (bool mono)</code> <code>CamSetMonochrome (int nCamNum, bool mono)</code>		
Parameters			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>mono</b>	<b>True</b> - sensor is monochrome <b>False</b> - sensor is color
	<b>Output</b>	<b>None</b>	

## GetMonochrome / CamGetMonochrome

<b>Description</b>	Returns whether sensor outputs monochrome (greyscale) data.		
<b>Syntax</b>	<code>GetMonochrome (bool *mono)</code> <code>CamGetMonochrome (int nCamNum, bool *mono)</code>		
Parameters			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>mono</b>	<b>True</b> - sensor is monochrome <b>False</b> - sensor is color

## SetFixBadPixels / CamSetFixBadPixels

<b>Description</b>	<p>Sets whether "bad" pixels should be fixed. "Bad pixels" are defined by one or two wrong colored pixels in an area of a known hue/color. The "bad pixels" originate from the sensor and are fixed on the PC.</p> <p>Only if width and height are at least 4 can this be performed. The "bad pixels" are fixed in stages, so it takes a few seconds to fix all of them on frames of 5Mp.</p>		
<b>Syntax</b>	<pre>SetFixBadPixels (int fOn, int *pnBadPixels) CamSetFixBadPixels (int nCamNum, int fOn, int *pnBadPixels)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>fOn</b>	0 - OFF 1 - ON, anything else will return number of bad pixels without fixing
		<b>pnBadPixels</b>	Number of bad pixels (if fOn > 1)
	<b>Output</b>	<b>None</b>	

## Advanced API Commands

### SetTestData / CamSetTestData

<b>Description</b>	<p>Specifies whether the camera should output Test Data or captured video images. This option is provided for developers to check if the data received is actually the data sent and other sensor independent features.</p> <p>WVGA, 1.3Mp and 3Mp cameras have only 1 option – to choose a value that is output for even columns, and its complement value - for odd columns. For example, 0x12 becomes 0xED.</p> <p>The 5Mp and 10Mp cameras have other options for creating patterns.</p>		
<b>Syntax</b>	<pre>SetTestData (bool useTestData, int type,              int redData, int greenData, int blueData, int barWidth) CamSetTestData (int nCamNum, bool useTestData, int type, int                 redData, int greenData, int blueData, int barWidth);</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>useTestData</b>	<b>True</b> – use Test Data <b>False</b> – don't use Test Data
		<b>type</b>	<p>Only applicable for 5Mp sensor. Values 0-8.</p> <p>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</p> <p>For detailed description, see Micron's data sheets.</p>
		<b>redData</b>	Value for <b>R</b> pixel (in 5Mp). Value for Test Data in 1.3Mp and 3Mp – is output and then its compliment in a pattern.
		<b>greenData</b>	Value for <b>G</b> pixel (in 5Mp).
		<b>blueData</b>	Value for <b>B</b> pixel (in 5Mp).
		<b>barWidth</b>	Width of bars if type 6, 7, or 8 should be an odd number
	<b>Output</b>	<b>None</b>	

## SetLowPowerMode / CamSetLowPowerMode

<b>Description</b>	Specifies whether the camera should enter Low Power mode. If video is now streaming the camera will enter Low Power mode when streaming stops. <b>This command is not supported in any of the Camelot models or IDM-500 cameras.</b>		
<b>Syntax</b>	<pre>SetLowPowerMode (bool fOn) CamSetLowPowerMode (int nCamNum, bool fOn)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>fOn</b>	<b>True</b> - use LowPower mode <b>False</b> - don't use Low Power Mode

## SetBadFrameDetection / CamSetBadFrameDetection

<b>Description</b>	Specifies whether the camera should detect and correct Bad Frames.		
<b>Syntax</b>	<pre>SetBadFrameDetection (bool fOn) CamBadFrameDetection (int nCamNum, bool fOn)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>fOn</b>	<b>True</b> - detect and correct bad frames <b>False</b> - don't detect bad frames

## SetGetRawFullData / CamSetGetRawFullData

<b>Description</b>	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 <a href="#">SetPllRate</a> command) before requesting the RAW frame. <b>This command is not supported by the IDM-500 cameras.</b> <b>Explanation of the PLL Rate for 5Mp cameras</b> We recommend using as low a PLL rate as possible for the the frame rate required. The fewer frames the sensor captures, the more time the DSP has for processing them. Since the USB bandwidth doesn't allow for more than 8.5 fps, there is no reason for the sensor to output 14fps.		
<b>Syntax</b>	<pre>SetGetRawFullData (SET_RAW_DATA_API *rawDataMsg) CamSetGetRawFullData (int nCamNum, SET_RAW_DATA_API *rawDataMsg)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>rawDataMsg</b>	<b>Pointer to</b> SET_RAW_DATA_API struct defined as: <pre>unsigned int frameType; // 0 - regular, 1 - RAW unsigned int numBits; unsigned int resolution; unsigned int withLut; unsigned int count; unsigned int withFlash;</pre>
	<b>Output</b>	<b>None</b>	

## Description of the PLL Rate for 5Mp cameras

We recommend using as low a PLL rate as possible for the frame rate required. The fewer frames the sensor captures, the more time the DSP has for processing them. Since the USB bandwidth doesn't allow for more than 8.5 fps, there is no reason for the sensor to output 14fps.

### SetPllRate / CamSetPllRate

<b>Description</b>	Sets the camera's PLL (internal clock) rate. Video streaming is stopped before PLL is set. <b>Default:</b> 60MHz. <b>This option is only supported by the 5Mp models.</b>		
<b>Syntax</b>	<pre>SetPllRate(int pllRate) CamSetPllRate(int nCamNum, int pllRate)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>pllRate</b>	Sets the internal clock in Mhz. Values must be in the range of 16 – 96.
	<b>Output</b>	<b>None</b>	

### GetPllRate / CamGetPllRate

<b>Description</b>	Gets the camera's PLL (internal clock) rate. <b>This option is only supported by the 5Mp models.</b>		
<b>Syntax</b>	<pre>GetPllRate(int *pllRate) CamGetPllRate(int nCamNum, int *pllRate)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
	<b>Output</b>	<b>*pllRate</b>	Gets the internal clock in Mhz.

## Commands Using the GPIO Connector on the Camera

Camelot Cameras have a 20-pin Molex connector, described in [Appendix A](#) on page 64.

The IDM-500 series has a 5-pin Molex connector. These pins can be defined for specific uses in specific applications.

### SetPwmTimer / CamSetPwmTimer

<b>Description</b>	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 high. The width must be less or equal to the period for each timer. The timers (TMR1, TMR2 and TMR3) are started simultaneously and therefore synchronized. <b>This command is not supported in IDM-500 cameras.</b>		
<b>Syntax</b>	<pre>SetPwmTimer (PWM_API pwmTimer) CamSetPwmTimer (int nCamNum, PWM_API *pwmTimer)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>PWM_API</b>	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 unsigned int EnableTimers;  // 1- constantly on,                              // 0 - will flash only                              // for specified frame  unsigned int timeout; // 0 - no timeout,                      // or number of milliseconds                      // to shut off flash - even if                      // on "constantly"           </pre>	
<b>Output</b>	<b>None</b>	See <a href="#">The Appendix</a> for the connector description	

## Using General Purpose Input/Output (GPIO)

Up to nine GPIO (general purpose input/output) pins are reserved for the user to set and read. These pins can be configured as input, output or trigger – which means they either read a value from the camera, set a value or trigger a frame.

These GPIO pins can turn on/off a circuit or transmit information from the camera.

```
// GPIO options
#define GPIO_OUTPUT      0
#define GPIO_INPUT      1
#define GPIO_TRIGGER     2
#define GPIO_INTERRUPT   3
#define GPIO_STROBE     4 // n.a. yet

#define MAX_IO_PINS     20 // 9 of which are provided in the 20 pin connector

typedef struct
{
    struct
    {
        int Config; // GPIO_OUTPUT or GPIO_INPUT or GPIO_TRIGGER
        int Value; // 0 or 1
    } Gpio[MAX_IO_PINS];
} GPIO_API;
```

## GetGPIO / CamGetGPIO

<b>Description</b>	Gets the current state and values of up to 9 GPIO (general purpose input/output) pins that are reserved for end user use.		
<b>Syntax</b>	<pre>GetGPIO(GPIO_API *gpioApi) CamGetGPIO(int nCamNum, GPIO_API *gpioApi)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*gpioApi</b>	A pointer to a GPIO_API struct (see above)
	<b>Output</b>	<b>None</b>	

## SetGPIO / CamSetGPIO

<b>Description</b>	Sets the current state and values of up to 9 GPIO (general purpose input/output) pins that are reserved for end user use.		
<b>Syntax</b>	<pre>SetGPIO(GPIO_API *gpioApi) CamSetGPIO(int nCamNum, GPIO_API *gpioApi)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*gpioApi</b>	A pointer to a GPIO_API struct (see above)
	<b>Output</b>	<b>None</b>	

## GetGPIO\_Pins / CamGetGPIO\_Pins

<b>Description</b>	Gets the current values of the up to 20 GPIO, general purpose input/output, pins that are reserved for application use. Each bit represents a pin and can either be high (1) or low (0).		
<b>Syntax</b>	<pre>GetGPIO_Pins (int *pins) CamGetGPIO_Pins (int nCamNum, int *pins)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>*pins</b>	A pointer to 32bit variable representing states of up to 20 GPIOs	
<b>Output</b>	<b>None</b>		

## SetGPIO\_Pins / CamSetGPIO\_Pins

<b>Description</b>	Sets the current values of the up to 20 GPIO, general purpose input/output, pins that are reserved for application use. Each bit represents a pin and can either be high (1) or low(0).		
<b>Syntax</b>	<pre>SetGPIO_Pins (int *pins) CamSetGPIO_Pins (int nCamNum, int *pins)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>*pins</b>	A pointer to 32bit variable representing states of up to 20 GPIOs	
<b>Output</b>	<b>None</b>		

## CheckLeds / CamCheckLeds

<b>Description</b>	Checks whether the three external LEDs (optionally provided with the camera) are functioning. <b>To utilize this command a separate control board is required.</b> <b>This command is not supported in IDM-500 cameras.</b>		
<b>Syntax</b>	<pre>int CheckLeds () CamCheckLeds (int nCamNum)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
<b>Output</b>	<b>int</b>	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	

## Setting/Getting Sensor Register Values

### GetRegVal / CamGetRegVal

<b>Description</b>	Returns value of specified register. Register addresses and values are in HEX notation. If fails returns -1.		
<b>Syntax</b>	<pre>GetRegVal(int nRegAddr, int *pnRegVal) CamGetRegVal(int nCamNum, int nRegAddr, int *pnRegVal)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nRegAddr</b>	Register number
	<b>Output</b>	<b>pnRegVal</b>	Value

### SetRegVal / CamSetRegVal

#### WARNING

This command 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.

<b>Description</b>	Sets specified register value. Register addresses and values are in HEX notation.		
<b>Syntax</b>	<pre>SetRegVal(int nRegAddr, int nRegVal) CamSetRegVal(int nCamNum, int nRegAddr, int nRegVal)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nRegAddr</b>	Register number
		<b>nRegVal</b>	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.
	<b>Output</b>	<b>None</b>	

## GetRegValues / CamGetRegValues

<b>Description</b>	Outputs register values to a <b>REG_LIST_API</b> structure as defined in <b>hostCamApi.h</b>		
<b>Syntax</b>	<pre>GetRegValues (REG_LIST_API *regApi, char numSection CamGetRegValues (int nCamNum, REG_LIST_API *regApi, char numSection)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>numSection</b>	Each regApi structure holds 100 registers and values. numSection specifies which registers should be output. numSection = 0 – registers 0-99 = 1 – registers 100-199 =2 – registers 200-299, etc.	
<b>Output</b>	<b>regApi</b>	Structure with up to MAX_REG_NUM registers and their values	

## SetRegValues / CamSetRegValues

### WARNING

This command 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.

<b>Description</b>	Sets register values according to list supplied in structure. Register addresses and values are in HEX notation.		
<b>Syntax</b>	<pre>SetRegValues (REG_LIST_API regApi) CamSetRegValues (int nCamNum, REG_LIST_API *regApi)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>regApi</b>	Structure with up to MAX_REG_NUM registers and their values	
<b>Output</b>	<b>None</b>		

## Look-up Table (LUT)

### UploadLUT / CamUploadLUT

<b>Description</b>	<p>Uploads a Look-up Table, LUT, for translating raw pixel values in the camera before being output. 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. After the LUT is successfully loaded, SetLUT may be called in order to start using the LUT for new data.</p> <p><b>Not all of the Camelot models support this command. This command is not supported in the IDM-500 series.</b></p>		
<b>Syntax</b>	<pre>UploadLut(char* sFileName, int numBits, bool transform) CamUploadLut(int nCamNum, char *sFileName, int numBits, bool transform)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>sFileName</b>	Valid file name – must exist. Each value should be on a separate line.
		<b>numBits</b>	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.
		<b>transform</b>	Whether values should be transformed since 8 MSB are in lower byte. Should be <b>true</b> .
	<b>Output</b>	<b>None</b>	
<b>Return Values</b>		<b>true</b>	Uploaded successfully
		<b>false</b>	LUT was not uploaded successfully

### SetLUT / CamSetLUT

<b>Description</b>	<p>Can be set only after a LUT is successfully uploaded. Indicates to camera that all data should be translated with the uploaded LUT (Look up Table).</p> <p><b>This command is not supported in IDM-500 cameras.</b></p>		
<b>Syntax</b>	<pre>SetLut(bool useLUT) CamSetLUT(int nCamNum, bool useLUT)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>useLUT</b>	Whether LUT should be used on new data
	<b>Output</b>	<b>None</b>	

## IMAGE PROCESSING COMMANDS

---

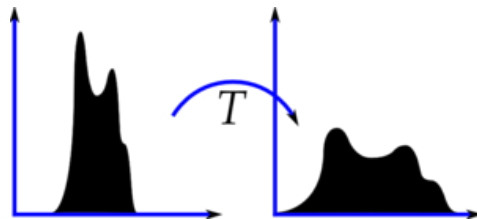
### Color Correction Options - Processing on PC

#### Histogram Stretching

Saturates the upper and lower  $s\%$  of the images color range ( $[lo, hi]$ ) and linearly stretches the  $[lo+s\%, hi-s\%]$  range to  $[0\%,100\%]$ .

This provides better color separation.

#### Histogram Equalization



The Histogram is stretched in a non-linear manner. This method usually increases the global [contrast](#) of many images, especially when the usable [data](#) of the image is represented by close contrast values. Through this adjustment, the [intensities](#) can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

#### Color Correction (CCM)

A camera's sensor may deliver images in which the colors are not true. A color correction matrix can be generated and applied to images to provide optimal color correction. A  $3 \times 3$  matrix can be applied to each RGB triplet resulting in a corrected triplet R'G'B'. It is often more effective to generate a  $4 \times 3$  matrix that introduces a fixed offset for each color component into the correction.

## SetColorCorr / CamSetColorCorr

<b>Description</b>	Sets the type of color correction to be implemented on the PC for all incoming frames. If using Color Correction, default values can be found in the cc.ini file.		
<b>Syntax</b>	<pre>SetColorCorr (COLOR_CORR_TYPE type) CamSetColorCorr (int nCamNum, COLOR_CORR_TYPE type)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>type</b>	Type of color correction according to table: <pre>CC_OFF = 0, CC_CLR_CORR_MATRIX = 1, CC_HIST_EQUALIZATION = 2, CC_HIST_STRETCH = 4, CC_GAMMA = 8</pre>
	<b>Output</b>	<b>None</b>	

## AWB (Automatic White Balance)

White balance (WB) is the process of removing unrealistic color casts, so that objects which appear white in person are rendered white in your photo. Proper camera white balance has to take into account the "color temperature" of a light source, which refers to the relative warmth or coolness of white light. Our eyes are very good at judging what is white under different light sources; however digital cameras must be programmed to perform auto white balance (AWB). An incorrect WB can create unsightly blue, orange, or even green color casts, which are unrealistic and particularly damaging to portraits.

## AGC (Automatic Gain Control)

For a camera to produce a quality picture it needs a minimum amount of light. When the lighting conditions are reduced below this minimum level the AGC (Automatic gain control) increases the amount of amplification to bring the video signal back up to the minimum required level. When the intensity of the light source is too high the AGC reduces the gain, thereby providing a better quality picture.

**The AWB and AGC commands are not supported in IDM-500 cameras.**

Image Control structure for advanced use of AWB and AGC:

```
typedef struct
{
    //Abbreviated
    int          fAGC;          //Switch AGC ON|OFF 1|0
    unsigned long periodAWB_sec; //periodAWB_sec > 0: AWB is done periodically
                                //    every no less than periodAWB_sec.
                                //periodAWB_sec = 0: AWB OFF.
                                //A scheduled AWB calculation will be
                                //postponed while an AGC run is ongoing.

    Int         reserved[4];    //for future stuff
    //-----
    //General
    int         wcell          //Pixels sampled on (width / wcell) x (height / vcell) grid
    , vcell;          //Default: wcell = vcell = 32
    //AGC -----
    int         mid_brightness // desired average brightness of image. Default: 128
    , brightness_margin; //Brightneses in range
                                // [mid_brightness+/-brightness_margin] are acceptable
                                // Default: 8
    int         saturation_threshold //Any brightness < saturation_threshold is
                                // considered black-saturated.
                                //Only brightnesses of 255 are considered
                                // white-saturated.
                                // Default: 16

    , saturation_permil; //Number of tenths of percent of pixels.
                                //No brightness increase will be implemented if more
                                // than saturation_permil of them are white-saturated.
                                //No brightness decrease will be implemented if more
                                // than saturation_permil of them are black-saturated.
    unsigned long t_take_effect_ms; //Min number of ms between sensor register
                                // brightness 'set's.
                                // Default: 0 (delay betw. successive frames)
    int         exposure_lo, exposure_hi //Exposure will not be incremented beyond this
                                // range. Default: 50,200
    , gain_lo, gain_hi //Gain will not be incremented beyond this
                                // range. Default: 8,200
    , gain_step; //Gain increment used in algorithm. Default: 1
                                //Higher values can result in jerky algorithm
                                // convergence.
    int         initial_gain, initial_exposure; //AGC will start with these values
                                //Recommended values: 60, 30; AGC will
                                // not work well in the low-gain range
    //AWB -----
    int         fDrawAWBIndicator; //On-frame indicator that AWB is in process 1|0
} IMAGE_CTRL_API;
```

## SetAGC\_AWB / CamSetAGC\_AWB

<b>Description</b>	Sets if a camera should perform Automatic Gain Control, AGC, and/or Automatic White Balance, AWB, on data. Each of these commands dynamically changes gain values and shutter width. When AGC and AWB are disabled, the last, calculated, shutter width and gain values remain. <b>This command is not supported in IDM-500 cameras.</b>		
<b>Syntax</b>	<pre>SetAGC_AWB (IMAGE_CTRL_API *pica, int icaSize) CamSetAGC_AWB (int nCamNum, IMAGE_CTRL_API *pica, int icaSize)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*pica</b>	Pointer to struct IMAGE_CTRL_API defined above
		<b>icaSize</b>	Size of ImageControlApi structure. If size of 6 int, then is "abbreviated version" with just AWB period and AGC on/off - all other values are defaults.
	<b>Output</b>	<b>none</b>	

## SingleFrameAWB / CamSingleFrameAWB

<b>Description</b>	Request the camera to perform the Automatic White Balance, AWB, algorithm on a single frame. AWB will be performed a specified number of times and then cease. This command is not supported in IDM-500 cameras.		
<b>Syntax</b>	<pre>SingleFrameAWB (int nRepeat) CamSingleFrameAWB (int nCamNum, int nRepeat)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>nRepeat</b>	On how many frames should the AWB algorithm be performed.
	<b>Output</b>	<b>none</b>	

## PC\_AWB / CamPC\_AWB

<b>Description</b>	Request the camera to perform the Automatic White Balance, AWB, algorithm on the frame on the PC. The new, calculated gain values will be sent to the camera.		
<b>Syntax</b>	<pre>PC_AWB (int period_ms) CamPC_AWB (int nCamNum, int period_ms)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>period_ms</b>	How often should the AWB algorithm be performed?
	<b>Output</b>	<b>none</b>	

## CONFIGURING THE CAMELOT IP CAMERAS

The Camelot IP cameras can be configured using the **IP\_TechnicianSetup.exe** program. When Camelot IP cameras are shipped, they have a default IP address of **192.168.168.253**.

Every camera is shipped with 3 IP configurations – one which is the “active” configuration and two others that are “defaults”. The “active” configuration is set using the **IP\_TechnicianSetup** or the **SetIpConfig** command below. The default configurations are set using the **SetIpDefaultConfig** command below. In order to switch to one of the default settings a reset pulse (switches to default 1) or a reset long-press (over 5 seconds – switches to default 2) is done to the IP camera.

For more specific instructions relating to Camelot IP cameras please refer to the **ID\_Camelot\_IP\_Guide.pdf**.

The following API commands allow the user to programmatically configure the cameras using the following structure:

```
typedef struct tNET_CONFIG_INFO
{
    short use_dhcp;           // 0 - don't use DHCP, 1 - use DHCP
    unsigned char mac_addr[6]; // MAC address - read only
    int ipaddr;              // IP address
    int netmask;             // IP netmask
    int gateway;             // IP gateway
    unsigned short controlPort; // PC control port - read only
    unsigned short dataPort;   // PC control port - read only
} NET_CONFIG_INFO;
```

### SetIpConfig / CamSetIpConfig

<b>Description</b>	Sets IP configuration according to the NET_CONFIG_INFO structure above.		
<b>Syntax</b>	SetIpConfig(NET_CONFIG_INFO *ipConfig) CamSetIpConfig(int nCamNum, NET_CONFIG_INFO *ipConfig)		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>*ipConfig</b>	Pointer to struct NET_CONFIG_INFO defined above
	<b>Output</b>	<b>none</b>	

## GetIpConfig / CamGetIpConfig

<b>Description</b>	Gets IP configuration according to the NET_CONFIG_INFO structure above.		
<b>Syntax</b>	<pre>GetIpConfig(NET_CONFIG_INFO *ipConfig) CamGetIpConfig(int nCamNum, NET_CONFIG_INFO *ipConfig)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>*ipConfig</b>	Pointer to struct NET_CONFIG_INFO defined above	
<b>Output</b>	<b>none</b>		

## SetIpDefaultConfig / CamSetIpDefaultConfig

<b>Description</b>	Set IP default configurations – similar to SetIPConfig – but refers to default configurations.		
<b>Syntax</b>	<pre>SetIpConfig(int which, NET_CONFIG_INFO *ipConfig) CamSetIpConfig(int nCamNum, int which, NET_CONFIG_INFO *ipConfig)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>which</b>	Which default area (1 or 2)	
	<b>*ipConfig</b>	Pointer to struct NET_CONFIG_INFO defined above	
<b>Output</b>	<b>none</b>		

## GetIpDefaultConfig / CamGetIpDefaultConfig

<b>Description</b>	Get IP default configurations – similar to GetIPConfig – but refers to default configurations.		
<b>Syntax</b>	<pre>GetIpConfig(int which, NET_CONFIG_INFO *ipConfig) CamGetIpConfig(int nCamNum, int which, NET_CONFIG_INFO *ipConfig)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>which</b>	Which default area (1 or 2)	
	<b>*ipConfig</b>	Pointer to struct NET_CONFIG_INFO defined above	
<b>Output</b>	<b>none</b>		

---

## SetPortRange / CamSetPortRange

<b>Description</b>	Set range of ports to be used by IP application.		
<b>Syntax</b>	<code>SetPortRange(int numPorts, unsigned short portStart)</code> <code>CamSetPortRange(int numPorts, unsigned short portStart)</code>		
<b>Parameters</b>			
	<b>Input</b>	numPorts	How many ports are in the range
		portStart	From which port to start
	<b>Output</b>	none	

## USING THE CALLBACK COMMANDS

---

Callback commands are commands that are accessed whenever an event or interrupt occurs. In the Camelot API, the callback command is accessed, if so defined, for each new frame.

The callback command(s) must be of the following type:

```
typedef void (__stdcall *funcDataPtr) (void *dataPtr, int size,
                                       FRAME_TYPE type, CB_DATA *pCbData);
```

where FRAME\_TYPE is defined as:

```
// Frame types output from camera
typedef enum
{
    NO_FRAME      = 0,
    BAYER_8_BIT   = 10, // regular file type for display
    BAYER_RAW,    // 8, 10 or 12 bit raw full data
    FILLER,       // throw away
    RGB_24,       // BMP using RGB24
    CONTROL_DATA, // msgs from camera
    MONO_RED,     // mono frame - use only red pixels
    MONO_GREEN,   //           - use only green pixels
    MONO_BLUE,    //           - use only blue pixels
    RGB_32,       // BMP using RGB32
    BLOCK_DATA,   // blocks sent, described in SYNC message
    RGB_8_MONO    // RGB8
} FRAME_TYPE;
```

and where CB\_DATA is defined as:

```
typedef struct
{
    int pCamelotDlg; // set when activating CB command -
                    // pointer to calling dialog
    int pCam;       // set when activate CB function - pointer to
                    // camera
    int camNum;     // set when activate CB function - camera number
                    // in dialog's listbox
    tFrameInfo FrameInfo; // supplied by dll - data from camera
                    // - describes captured, raw frame
    int pUserData;  // supplied by user - specific to CB
                    // functionality - used like void *
} CB_DATA;
```

In the CamelotView sample, 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:

```
DataCallback( unsigned char *dataPtr, int size, FRAME_TYPE type,
              CB_DATA *pCbData)
```

You can program the Sample Application to use a callback function whenever:

- a raw frame (Bayer 8bit) is received
- a RAW-FULL frame is received (Bayer 8,10, or 12-bit)
- a frame has been converted to RGB (RGB32)

Any parameters or information needed by the command can be sent using the `pUserData` pointer in `CB_DATA`.

There are three call back examples provided with the application:

1. a Histogram can be applied to all RAW (`BAYER_8_BIT`) data,
2. an Edge filter function can be applied to all RGB32 frames
3. if a Callback function is defined for RAW FULL frames (8, 10, 12-bit), then they are saved to the disk. Otherwise, the RAW FULL frames are converted to 8 bit data using a gamma LUT file, and displayed.

In order to setup the callback function capability, first call `SetDataCBPtr` with a pointer to a function with the same format as `DataCallback` above.

Then set the specific type of callback required by calling one (or more) of the following:

- `SetRawFrameDataCB` – for raw frames before any type of processing
- `SetRawFullFrameCB` – for RAW FULL frames (8, 10, or 12 bits)
- `SetRGBFrameDataCB` – for RGB32 frames – after Bayer conversion

#### NOTE

The pointer to the frame data (`*dataPtr`) is only valid within the callback function. If the data is to be used by another thread or after the callback exits, the data must be copied to another buffer.

## SetDataCBPtr / CamSetDataCBPtr

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

## SetRawFrameDataCB / CamSetRawFrameDataCB

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

## SetRawFullFrameCB / CamSetRawFullFrameCB

<b>Description</b>	Sets whether a callback function should be called for each raw Full frame received (as a result of the <a href="#">SetGetRawFullData</a> function on page 41).		
<b>Syntax</b>	<pre>SetRawFullFrameCB (bool useCB, void *pUserdata) CamSetRawFullFrameCB (int nCamNum, bool useCB, void *pUserdata)</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>useCb</b>	Whether this callback function should be set.
		<b>pUserData</b>	Pointer to user data for the Callback function
	<b>Output</b>	<b>None</b>	

## SetRGBFrameDataCB / CamSetRGBFrameDataCB

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

## SaveRawFrameToFileBuf / CamSaveRawFrameToFileBuf

<b>Description</b>	Saves a captured Raw image to a file in RAW, BMP or JPG format. This command can only be used within the RawFrame or RawFullFrame callbacks.		
<b>Syntax</b>	<pre>SaveRawFrameToFileBuf(unsigned char *pRawFrame, tFrameInfo *FrameInfo, tSaveFrameInfo *SaveInfo) CamSaveRawFrameToFileBuf(int nCamNum, unsigned char *pRawFrame, tFrameInfo *FrameInfo, tSaveFrameInfo *SaveInfo) {</pre>		
<b>Parameters</b>			
	<b>Input</b>	<b>nCamNum</b>	Camera number
		<b>pRawFrame</b>	Pointer to the raw data
		<b>FrameInfo</b>	Information about frame – width, height, bytes per pixel, etc.
		<b>SaveInfo</b>	Parameters for saving file.
	<b>Output</b>	<b>None</b>	

## SetMsgCBPtr

<b>Description</b>	Sets pointer to an error message callback function. The dll functions FillCameraList, and AccessCamera also send this pointer.		
<b>Syntax</b>	<b>SetMsgCBPtr (funcMsgPtr funcPtr)</b>		
<b>Parameters</b>			
	<b>Input</b>	<b>funcMsgCb</b>	Pointer to a Callback function with Error messages that will be called from the dll if an error occurs. Recovery code can be inserted if needed.
	<b>Output</b>	<b>None</b>	

## 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 **ID\_debugger** window by running **ID\_debugger.exe**.

**ID\_debugger.exe** is included on the Installation CD.

3. In the PC code, call the **DebugPrintx(msg[,val1, val2])** function with an ASCII message as a parameter.

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

4. To display messages from the Camera's DSP, use the [GetLogs](#) function described below.

### DebugPrint

<b>Description</b>	Prints a message to a debug window, if open, while in DEBUG mode.		
<b>Syntax</b>	DebugPrint1(char * Str)		
<b>Parameters</b>			
	<b>Input</b>	<b>*Str</b>	Char string – ASCII message to be printed
	<b>Output</b>	<b>none</b>	
<b>Syntax</b>	DebugPrint2(char * Str, int Val1)		
<b>Parameters</b>			
	<b>Input</b>	<b>*Str</b>	Char string – ASCII message to be printed
		<b>Val1</b>	Number to be printed
	<b>Output</b>	<b>none</b>	
<b>Syntax</b>	DebugPrint3(char * Str, int Val1, int Val2)		
<b>Parameters</b>			
	<b>Input</b>	<b>*Str</b>	Char string – ASCII message to be printed
		<b>Val1</b>	Number to be printed
		<b>Val2</b>	Number to be printed
	<b>Output</b>	<b>none</b>	

## GetLogs / CamGetLogs

<b>Description</b>	Gets log messages saved in the camera for debugging purposes. These messages are sent to a separate <a href="#">DebugPrint</a> screen (if opened) and can be used to debug applications. See <a href="#">Using DebugPrint</a> on page 60. <b>This command is not supported in IDM-500 cameras.</b>		
<b>Syntax</b>	GetLogs () CamGetLogs (int nCamNum)		
<b>Parameters</b>			
	<b>Input</b>	nCamNum	Camera number
	<b>Output</b>	None	

## UPGRADING TO THE LATEST SOFTWARE VERSION

---

In order to obtain the most recent FW version, please be in touch with our support team [support@imagine2d.com](mailto:support@imagine2d.com). Visit our website, <http://www.imagine2d.com/> and download and then install the latest version of both the software and camera drivers currently available.

### 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 at [support@imagine2d.com](mailto:support@imagine2d.com) so we can run it for you to assure the camera will boot after loading your file.

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

1. Download the latest Camelot software version from our site to a temporary location. At [www.imagine2d.com](http://www.imagine2d.com), in the Support tab under [Downloads](#).
2. Upload the new firmware (**Camelot\_FW.ldr**) to the camera using the current/old working CamelotView (NOT the file you just downloaded) or the UpdateFW API command [below](#).
  - a. In CamelotView, in the Processing tab (in the Advanced options pane) browse to the new ldr file and click on the Update FW button.



3. When the firmware upload is successfully completed, unplug the camera and close the CamelotSample application
4. Put the new version of **CamelotView.exe** and **Camelot.dll** into - **X:\Program Files\Imaging Diagnostics\Camelot**. Where: **X:\** is the system disk. (It's a good idea to keep a backup of the older version.)
5. If using Direct Show, copy the Camelotfilter.ax file into the folder **X:\Windows\system32** where: **X:\** is the system disk.
6. Reconnect the camera, copy and then run the NEW **CamelotView.exe** file.

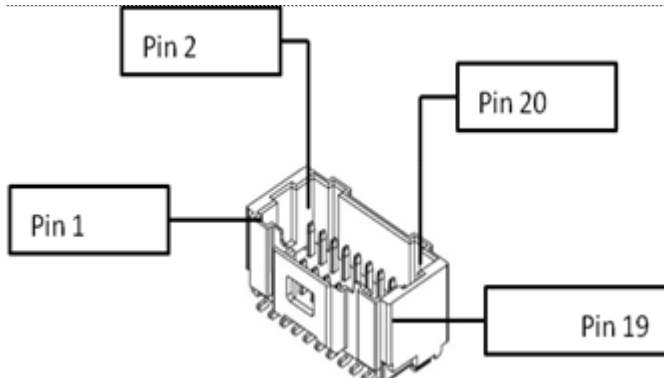
## UpdateFW / CamUpdateFW

<b>Description</b>	Updates the camera Firmware. Video streaming is stopped before firmware is updated.		
<b>Syntax</b>	<pre>UpdateFW(char* a_sFileName) CamUpdateFW(int nCamNum, char *a_sFileName)</pre>		
<b>Parameters</b>			
<b>Input</b>	<b>nCamNum</b>	Camera number	
	<b>a_sFileName</b>	Valid file name of type *.ldr- must exist and must be in correct analog loader file format.	
<b>Output</b>	<b>None</b>		
<b>Return Values</b>	<b>S_OK</b>	0 – updated successfully	
	<b>S_FALSE</b>	1 – camera FW was not updated successfully	

# Appendix A MOLEX CONNECTOR TABLE

## Molex Connector

The table below shows the pin connection table for the Molex Connector located on the rear of the Camelot series cameras.



**Table 2: Molex Connector Table**

Pin Num	Pin Name	Description	Remarks
<b>1</b>	GPIO1	DSP_GPIO1	PJ0-Through serial resistor
<b>2</b>	Power	Power	
<b>3</b>	GPIO2	DSP_GPIO2	PJ1-Through serial resistor
<b>4</b>	Power	Power	
<b>5</b>	GPIO3	DSP_GPIO3	PJ2-Through serial resistor
<b>6</b>	PWM1	PB9	TMR1
<b>7</b>	GPIO4	SPI1_SS	PG11-Through serial resistor and capacitor
<b>8</b>	PWM2	PB10	TMR2
<b>9</b>	GPIO5	SPI1_SCK	PG8-Through serial resistor
<b>10</b>	PWM3	PB11	TMR3
<b>11</b>	GPIO6	SPI1_MISO	PG9-Through serial resistor
<b>12</b>	PWM4	PA13	TMR7
<b>13</b>	GPIO7	SPI1_MOSI	PG10-Through serial resistor
<b>14</b>	PWM5	PH2	TMR8
<b>15</b>	GPIO8	UART2-TX	PB4-Through serial resistor
<b>16</b>	PWM6	PH3	TMR9
<b>17</b>	GND	GND	
<b>18</b>	PWM7	PH4	TMR10
<b>19</b>	GND	GND	
<b>20</b>	GPIO9	UART2-RX	PB5-Through serial resistor

**Imaging Diagnostics** invites comments and feedback on this and all of our products and documentation. Please contact us at one of the email addresses below.

**Website**

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

**Support**

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

[support@imagine2d.com](mailto:support@imagine2d.com)

**Sales**

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

[sales@imagine2d.com](mailto:sales@imagine2d.com)



A member of the

