THORLABS

CCD and CMOS Cameras

DCU223x, DCU224x
DCC1240x
DCC1545M, DCC1645C
DCC3240X

DCx Camera Functional Description and SDK Manual

2015

Version: 4.60
Date: 12/7/2015

Copyright © 2015 Thorlabs Scientific Imaging

Contents

Foreword 8

1 General Information 9

1.1 Safety 9
1.2 Ordering Codes and Accessories 11
1.3 Requirements 11
1.4 DCx Camera Family 12
1.5 Contents 14
1.6 What's New in this Version? 16

2 Camera Basics 18

2.1 Operating Modes 18
2.1.1 Freerun Mode 18
2.1.2 Trigger Mode 20
2.1.3 Standby 21
2.2 Image Display Modes 22
2.3 Sensor 25
2.3.1 Sensor Sizes 25
2.3.2 Micro Lenses 26
2.3.3 Color Filter (Bayer filter) 29
2.3.4 Hot Pixels 31
2.3.5 Shutter Methods 33
2.3.6 Line Scan Mode 37
2.4 Reading Out Partial Images 38
2.4.1 Area of Interest (AOI) 38
2.4.2 Subsampling 42
2.4.3 Binning 43
2.5 Digitizing Images 44
2.5.1 Characteristics and LUT 44
2.5.2 Bit Depth and Digital Contrast Adjustment 47
2.6 Camera Parameters 50
2.6.1 Pixel Clock, Frame Rate, Exposure Time 50
2.6.2 Gain and Offset 51
2.6.3 Automatic Image Control 51
2.6.4 Applying New Parameters 53
2.7 Firmware and Camera Start 53
2.8 Digital Inputs / Outputs 53
2.8.1 Using Digital Inputs/Outputs 54
2.8.2 Flash Timing (Trigger Mode) 54
2.8.3 Flash Timing (Freerun Mode) 56
2.8.4 Serial Interface RS-232 (DC3240x only) 57

2.9 USB Interface 58
2.9.1 History and Development 58
2.9.2 Structure and Topology 58
2.9.3 USB 2.0 Cabling and Connectors 59
2.9.4 USB 3.0 Cabling and Connectors 60
2.9.5 Data Transmission and Bandwidth 60
3 Operation 62
3.1 uc480 Quick Start 62
3.2 Installation and Connection 64
3.2.1 System Requirements 64
3.2.2 DCx Driver Compatibility 65
3.2.3 Connecting a DCx Camera 65
3.3 Application Notes by Camera Model 67
3.3.1 DCC1240x / DCC3240x Application Notes 67
3.3.2 DCC1545M Application Notes 70
3.3.3 DCC1645C Application Notes 71
3.3.4 DCU223x Application Notes 71
3.3.5 DCU224x Application Notes 71
3.4 Installed uc480 Programs 72
3.4.1 uc480 Camera Manager 72
3.4.1.1 Camera List 73
3.4.1.2 Control Center 74
3.4.1.3 General Information 75
3.4.1.4 Camera Information 76
3.4.1.5 Creating a Support File 76
3.4.1.6 Additional Functions 76
3.4.1.7 Parameters 80
3.4.2 uc480 Hotpixel Editor 81
4 Programming (SDK) 84
4.1 First Steps to uc480 Programming 85
4.2 How to Proceed 87
4.2.1 Preparing Image Capture 87
4.2.1.1 Querying Information 87
4.2.1.2 Opening and Closing the Camera 88
4.2.1.3 Allocating Image Memory 88
4.2.1.4 Image Memory Sequences 90
4.2.2 Selecting the Display Mode 91
4.2.3 Capturing Images 92
4.2.3.1 Image Capture Modes 92
4.2.3.2 Event / Message Handling 100
4.2.4 Setting Camera Parameters 104
4.2.4.1 Setting and Getting Parameters 104

4.2.4.2 Automatic Image Control 107
4.2.4.3 Image Pre-processing 108
4.2.4.4 Get Camera Status 108
4.2.4.5 Using the Camera EEPROM 108
4.2.5 Saving Images and Videos 109
4.2.5.1 Saving and Loading Single Frames 109
4.2.5.2 Capturing AVIs 109
4.2.6 Using Inputs and Outputs 112
4.2.6.1 Input/Output Control 112
4.3 Function Descriptions 115
4.3.1 is_AddToSequence 116
4.3.2 is_AllocImageMem 117
4.3.3 is_AOI 119
4.3.4 is_AutoParameter 129
4.3.5 is_Blacklevel 132
4.3.6 is_CameraStatus 135
4.3.7 is_CaptureStatus 137
4.3.8 is_CaptureVideo 140
4.3.9 is_ClearSequence 142
4.3.10 is_ColorTemperature 143
4.3.11 is_Configuration 147
4.3.12 is_Convert 154
4.3.13 is_CopyImageMem 156
4.3.14 is_CopyImageMemLines 157
4.3.15 is_DeviceFeature 158
4.3.16 is_DeviceInfo 163
4.3.17 is_DirectRenderer 165
4.3.18 is_DisableEvent 175
4.3.19 is_EdgeEnhancement 176
4.3.20 is_EnableAutoExit 179
4.3.21 is_EnableEvent 180
4.3.22 is_EnableMessage 183
4.3.23 is_ExitCamera 185
4.3.24 is_ExitEvent 186
4.3.25 is_ExitImageQueue 187
4.3.26 is_Exposure 188
4.3.27 is_ForceTrigger 195
4.3.28 is_FreeImageMem 197
4.3.29 is_FreezeVideo 199
4.3.30 is_Gamma 201
4.3.31 is_GetActiveImageMem 202
4.3.32 is_GetActSeqBuf 204
4.3.33 is_GetAutoInfo 206

4.3.34 is_GetBusSpeed 210
4.3.35 is_GetCameraInfo 211
4.3.36 is_GetCameraList 213
4.3.37 is_GetCameraLUT 215
4.3.38 is_GetColorConverter 216
4.3.39 is_GetColorDepth 217
4.3.40 is_GetDLLVersion 218
4.3.41 is_GetError 219
4.3.42 is_GetFramesPerSecond 220
4.3.43 is_GetFrameTimeRange 221
4.3.44 is_GetImageHistogram 223
4.3.45 is_GetImageInfo 226
4.3.46 is_GetImageMem 230
4.3.47 is_GetImageMemPitch 231
4.3.48 is_GetNumberOfCameras 233
4.3.49 is_GetOsVersion 234
4.3.50 is_GetSensorInfo 235
4.3.51 is_GetSensorScalerInfo 237
4.3.52 is_GetSupportedTestImages 238
4.3.53 is_GetTestImageValueRange 240
4.3.54 is_GetTimeout 241
4.3.55 is_GetUsedBandwidth 242
4.3.56 is_GetVsyncCount 243
4.3.57 is_HasVideoStarted 244
4.3.58 is_HotPixel 245
4.3.59 is_ImageFile 250
4.3.60 is_ImageFormat 254
4.3.61 is_InitCamera 261
4.3.62 is_InitEvent 265
4.3.63 is_InitImageQueue 267
4.3.64 is_InquireImageMem 269
4.3.65 is_IO 271
4.3.66 is_IsVideoFinish 284
4.3.67 is_LockSeqBuf 286
4.3.68 is_LUT 287
4.3.69 is_Measure 290
4.3.70 is_ParameterSet 294
4.3.71 is_PixelClock 297
4.3.72 is_ReadEEPROM 300
4.3.73 is_RenderBitmap 302
4.3.74 is_ResetToDefault 304
4.3.75 is_SetAllocatedImageMem 306
4.3.76 is_SetAutoParameter 309

4.3.77 is_SetBinning 317
4.3.78 is_SetCameraID 320
4.3.79 is_SetColorConverter 322
4.3.80 is_SetColorCorrection 324
4.3.81 is_SetColorMode 326
4.3.82 is_SetDisplayMode 330
4.3.83 is_SetDisplayPos 334
4.3.84 is_SetErrorReport 335
4.3.85 is_SetExternalTrigger 336
4.3.86 is_SetFrameRate 338
4.3.87 is_SetGainBoost 340
4.3.88 is_SetGamma 341
4.3.89 is_SetHardwareGain 343
4.3.90 is_SetHWGainFactor 346
4.3.91 is_SetImageMem 348
4.3.92 is_SetOptimalCameraTiming 349
4.3.93 is_SetRopEffect 351
4.3.94 is_SetSaturation 353
4.3.95 is_SetSensorScaler 354
4.3.96 is_SetSensorTestImage 357
4.3.97 is_SetSubSampling 359
4.3.98 is_SetTimeout 363
4.3.99 is_SetTriggerCounter 365
4.3.100 is_SetTriggerDelay 366
4.3.101 is_StopLiveVideo 367
4.3.102 is_Trigger 368
4.3.103 is_TriggerDebounce 370
4.3.104 is_UnlockSeqBuf 374
4.3.105 is_WaitEvent 375
4.3.106 is_WaitForNextImage 376
4.3.107 is_WriteEEPROM 378
4.4 AVI Function Descriptions 380
4.4.1 isavi_AddFrame 380
4.4.2 isavi_CloseAVI 381
4.4.3 isavi_DisableEvent 381
4.4.4 isavi_EnableEvent 382
4.4.5 isavi_ExitAVI 383
4.4.6 isavi_ExitEvent 384
4.4.7 isavi_GetAVIFileName 385
4.4.8 isavi_GetAVIFileNameW 386
4.4.9 isavi_GetAVISize 387
4.4.10 isavi_GetnCompressedFrames 388
4.4.11 isavi_GetnLostFrames 389

4.4.12 isavi_InitAVI 390
4.4.13 isavi_InitEvent 391
4.4.14 isavi_OpenAVI 392
4.4.15 isavi_OpenAVIW 393
4.4.16 isavi_ResetFrameCounters 394
4.4.17 isavi_SetFrameRate 395
4.4.18 isavi_SetImageQuality 396
4.4.19 isavi_SetImageSize 397
4.4.20 isavi_StartAVI 399
4.4.21 isavi_StopAVI 400
4.5 RAW function descriptions 401
4.5.1 israw_AddFrame 401
4.5.2 israw_CloseFile 401
4.5.3 israw_ExitFile 402
4.5.4 israw_GetFrame 402
4.5.5 israw_GetImageInfo 403
4.5.6 israw_GetSize 404
4.5.7 israw_InitFile 404
4.5.8 israw_OpenFile 405
4.5.9 israw_SeekFrame 406
4.5.10 israw_SetImageInfo 406
4.6 Obsolete functions 407
4.7 Programming Notes 410
4.7.1 Programming in C/C++ 410
4.7.2 Programming in C# 411
4.7.3 Programming in VB.NET 412
4.7.4 Programming in Delphi 412
4.7.5 Programming with ActiveX 412
4.7.6 Thread Programming 412
4.8 Lists 414
4.8.1 Complete List of All Return Values 414
4.8.2 Error Codes of AVI Functions 418
4.8.3 Linux: Not Supported Functions 418
5 Specifications 420
5.1 Model Comparison 421
5.2 Model Naming Conventions 423
5.3 Camera and Sensor Data 423
5.3.1 DCC1240x / DCC3240x 424
5.3.2 DCC1545M 427
5.3.3 DCC1645C 429
5.3.4 DCU223x 431
5.3.5 DCU224x 434

5.4 Mechanical Specifications 437
5.4.1 DCU223x, DCU224x 438
5.4.2 DCC1240x 439
5.4.3 DCC3240x 440
5.4.4 DCC1545M, DCC1645C 441
5.4.5 Flange Back Distance 442
5.4.5.1 Calculating the Flange Back Distance 442
5.4.5.2 Maximum Immersion Depth for Lenses 444
5.4.6 Position Accuracy of the Sensor 445
5.4.7 Filter Glasses 446
5.4.7.1 Filter Types 446
5.4.7.2 Mounting the Filter 450
5.4.7.3 Cleaning the Filter Glasses 451
5.4.8 Ambient Conditions 453
5.5 Camera Interface 454
5.5.1 DCU223x, DCU224x, DCC1240x 454
5.5.1.1 I/O Connector - Pin Assignment 454
5.5.1.2 Digital Input (Trigger) Circuit 456
5.5.1.3 Digital Output (Flash) Circuit 457
5.5.2 DCC3240x 459
5.5.2.1 I/O Connector Pin Assignment 459
5.5.2.2 GPIO Interface 460
5.5.2.3 Digital Input (Trigger) Circuit 462
5.5.2.4 Digital Output (Flash) Circuit 463
5.5.2.5 RS-232 Serial Interface 464
5.5.3 Camera EEPROM Specification 465
5.6 Accessories for DCx cameras 466
5.6.1 Accessories for DCU22xX / DCC1240X 466
5.6.2 Accessories for DCC1x45X 467
5.6.3 Accessories for DCC3240x 467
6 Appendix 468
6.1 Troubleshooting/FAQ 468
6.1.1 PCs with Energy Saving CPU Technology 469
6.2 Status LED on USB DCx Cameras 470
6.3 Color and Memory Formats 472
6.4 uc480 Parameter File (ini file) 475
6.5 Definition of IP Protection Classes 482
6.6 History of API functions 482
6.7 Certifications and Compliances 484
6.8 Thorlabs 'End of Life' Policy (WEEE) 486
6.9 Exclusion of Liability and Copyright 487
6.10 Thorlabs Worldwide Contacts 488

Warning

Sections marked by this symbol explain dangers that might result in personal injury or death. Always read the associated information carefully, before performing the indicated procedure.

Attention

Paragraphs preceeded by this symbol explain hazards that could damage the instrument and the connected equipment or may cause loss of data.

© 2015 Thorlabs Scientific Imaging

1 General Information

Thank you for purchasing a DCx camera!

You should first read the following chapters to get a quick overview on what is new in this software version and on getting started with your new camera.

Getting started

- DCx quick-start
- First steps to DCx Camera programming

Further important information

- What is new in this version?
- Contents of this Manual
- The DCx camera family
- Specifications

Enjoy your new DCx camera!

1.1 Safety

Attention

All statements regarding safety of operation and technical data in this instruction manual will only apply when the unit is operated correctly as it was designed for.

Prior to applying power to the DCx Camera Functional Description and SDK Manual, make sure that the protective conductor of the 3 conductor mains power cord is correctly connected to the protective earth ground contact of the socket outlet! Improper grounding can cause electric shock with damages to your health or even death!

The DCx Camera Functional Description and SDK Manual must not be operated in explosion endangered environments!

Do not remove covers! Do not obstruct the air ventilation slots in the housing!

Do not open the cabinet, there are no parts serviceable by the operator inside!

Refer servicing to qualified personnel!

Only with written consent from *Thorlabs Scientific Imaging* may changes to single components be made or components not supplied by *Thorlabs Scientific Imaging* be used.

This precision device is only serviceable if properly packed into the complete original packaging. If necessary, ask for a replacement package prior to return.

Attention

The following statement applies to the products covered in this manual, unless otherwise specified herein. The statement for other products will appear in the accompanying documentation.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules and meets all requirements of the Canadian Interference-Causing Equipment Standard ICES-003 for digital apparatus. These limits are

designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Thorlabs Scientific Imaging is not responsible for any radio television interference caused by modifications of this equipment or the substitution or attachment of connecting cables and equipment other than those specified by Thorlabs Scientific Imaging. The correction of interference caused by such unauthorized modification, substitution or attachment will be the responsibility of the user.

The use of shielded I/O cables is required when connecting this equipment to any and all optional peripheral or host devices. Failure to do so may violate FCC and ICES rules.

Attention

Mobile telephones, cellular phones or other radio transmitters are not to be used within the range of three meters of this unit since the electromagnetic field intensity may then exceed the maximum allowed disturbance values according to IEC 61326-1.

This product has been tested and found to comply with the limits according to IEC 61326-1 for using connection cables shorter than 3 meters (9.8 feet).

© 2015 Thorlabs Scientific Imaging

1.2 Ordering Codes and Accessories

DCU223M	CCD camera, monochrome, 1024x768 pixel, C mount
DCU223C	CCD camera, color, 1280x1024 pixel, C mount
DCU224M	CCD camera, monochrome, 1280x1024 pixel, C mount
DCU224C	CCD camera, color, 1280x1024 pixel, C mount
DCC1545M	CMOS camera, monochrome, 1280x1024 pixel, CS mount
DCC1645C	CMOS camera, color, 1280x1024 pixel, CS mount
DCC1240M	CMOS camera, monochrome, 1280x1024 pixel, C mount
DCC1240C	CMOS camera, color, 1280x1024 pixel, C mount
CAB-DCU-T1	Trigger cable for DCU22xX and DCC1240X cameras (Trigger In/Out)
CAB-DCU-T2	Trigger cable for DCU22xX and DCC1240X cameras (Trigger In only)
CAB-DCU-T3	I/O cable for DC3240 CMOS USB 3.0 cameras

Thorlabs C Mount Camera Lenses (objectives): See Thorlabs' website

1.3 Requirements

For operating the DCx cameras, the following system requirements must be met:

	Recommended
CPU speed	>2.0 GHz Intel Core i5 or Core i7
Memory (RAM)	8 GByte
For USB DCx cameras: USB host controller	USB 3.0 Super Speed Intel® motherboard chipset
Graphics card	Dedicated AGP/PCIe graphics card Latest version of Microsoft DirectX Runtime 9.0c
Operating system	Windows 8.1 32 or 64 bit Windows 7 32 or 64 bit

Drivers for network cards

To ensure optimum performance of the network connection, you need to install the latest drivers for your network card. We recommend using the drivers of the following versions:

- Intel® chipsets: version 8.8 or higher
- Realtek chipsets: version 5.7 or higher

USB interface

- Onboard USB 2.0 ports usually provide significantly better performance than PCI and PCMCIA USB adapters.
- Current generation CPUs with energy saving technologies can cause bandwidth problems on the USB bus. See section on PCs With Energy Saving CPU Technology.

Large multi-camera systems

Connecting a large number of cameras to a single PC may require a large working memory (RAM). This is especially the case when many cameras with high sensor resolution are used.

If you want to set up such a system we recommend to use PCs with 64 bit operating systems and more than 4 GB of RAM.

Note on color cameras with high frame rates

For uc480 color cameras, the color conversion is done by software in the PC. When you use a color camera with a high frame rate, the conversion might lead to a high CPU load. Depending on the PC hardware used you might not be able to reach the camera's maximum frame rate.

Direct3D graphics functions

The uc480 driver can use Direct3D to display the camera image with overlay information (Microsoft DirectX Runtime had to be installed). On Windows systems, you can use the supplied "DXDiag" diagnostic tool to check whether your graphics card supports Direct3D functions. To start the diagnostic tool, click "Run..." on the Windows start menu (shortcut: Windows+R) and enter "DXDiag" in the input box.

On the "Display" page of the diagnostic tool, click the button for testing the Direct3D functions.

OpenGL graphics functions

For OpenGL version 1.4 or higher must be installed. The OpenGL graphics functions do not work with QT under Linux.

1.4 DCx Camera Family

DCx cameras stand for a range of compact and cost-effective cameras for professional use in industrial, security and non-industrial applications. Equipped with the widely used USB 2.0 and particularly USB 3.0 ports, they can easily be interfaced with a vast variety of systems. The images are digitized in the camera and transmitted digitally to the PC. An additional frame grabber is not required.

DCU cameras have state-of-the-art CCD sensors while the DCC models are CMOS based. The CMOS models use either the global or the rolling shutter method; the CCD models use only the global shutter method.

The DCx cameras are available as monochrome and color versions, DC3240 series has a NIR version as well. The Model Comparison chapter shows the most important features of every series at a glance.

USB 3.0 DCC3240x CMOS Cameras

Compact, fast and lightweight. The new **DCC3240x.** The 29 x 29 x 29 mm small camera housing is not only ultra-compact, but due to its magnesium casing and a total camera weight of 43 g, it is also ultra-lightweight and robust. The powerful camera offers a bandwidth of 400 MByte/s via USB 3.0. Power is supplied via the USB bus, hence an extra power cable is obsolete.

With its lockable Micro USB connector the camera is perfectly suited even for rough environments. Offering trigger and flash as well as two GPIOs (General Purpose I/O), which can also be changed into a serial interface

 © 2015 Thorlabs Scientific Imaging

(RS232). Hence, peripheral devices can easily be triggered or controlled.

But also the camera's inner values are outstanding: brightness corrections are easily realized by a comfortable 12 bit lookup table and hardware gamma. 12 bit color depth offers a by factor 16 increased level of detail compared to the usual 8 bit. Hardware based data preprocessing saves additional CPU resources.

USB 2.0 DCC1240x (CMOS) and DCC22xX (CCD) Cameras

The **DCC1240X and DCC22xX** series feature a robust metal housing with a standard mini-B USB 2.0 connector. Connection is additionally possible via a lockable micro D-sub connector which also carries the opto-isolated I/O signals.

The USB 2.0 interface is meanwhile available in every standard PC and notebook/laptop and provides a gross bandwidth of 60 MByte/s. The camera is connected and powered through the USB port by just a single cable.

USB2.0 DCC1545M and DCC1645C Cameras

The **DCC1x45X** series features extremely compact cameras with high-speed CMOS sensors. The LE models are designed for professional use in non-industrial applications. Through the use of the widespread USB 2.0 technology, the cameras can easily be interfaced with a vast variety of systems. These cameras are available with a plastic housing with CS-mount lens adapter.

1.5 Contents

The DCx Camera Manual contains all the information you need for operating your DCx camera. It comprises the following parts:

Section A: Camera basics

- In this section you will find a lot of important information on the technical background of your USB camera. This section contains explanations on the DCx's operating modes, on sensor technology, important camera parameters, and the USB interfaces. We recommend to read this chapter to become familiar with the general functionality of the DCx Cameras.

Section B: Operation

- Quick start to using your DCxCamera
- Installing and Using DCx Camera software
  These sections show how to connect cameras and start operation using the software tools .
- Application notes by camera model
  This section explains special features and limitations of some camera models.

Section C: Programming

- First steps to programming with your DCxCamera
- How to proceed
  If you are not yet familiar with DCxCamera programming, we suggest that you first explore the basic functional flows in this chapter. The function blocks contain almost all the functions available for the uc480 API ordered by topics. The flowcharts help to easily find the appropriate API function for a certain task.
- Description of functions/Description of AVI functions
  These chapters cover all the functions of the uc480 API in alphabetic order.
  The AVI functions for video recording are implemented by the `uc480_tools.dll` which is also included in the DCxCamera software package.
- Obsolete functions
  This chapter lists obsolote API functions and recommended alternatives.
- Lists and programming notes
  In this chapter, you will find useful information on how to use the DCxCamera programming API. Programming environments, modes for DCxCamera color and image display as well as the automatic image control functions are discussed here.

Section D: Specifications

- Specifications
  All information on the camera's sensor and performance, mechanical as well as electrical specifications are contained in this section.
- Accessories
  Here you will find a list of accessories for DCx Cameras sorted by model.

Appendix

- Information on Troubleshooting
- Status LEDs on USB DCx cameras
- Color and memory formats
- uc480 parameter file (ini file)

 © 2015 Thorlabs Scientific Imaging

- Definition of IP protection classes

1.6 What's New in this Version?

Version 4.60 of the DCxCamera software package includes many new features and enhancements. The following table gives you an overview of the major new functions.

New in Version 4.60

Cameras & functions	Described in chapter
CCD models with hardware revision 3: • Improved long time exposure function • 12 bit per pixel now also available for USB 2.0 cameras	Camera and sensor data
More functions for the USB 3 DCC3240 camera series: • Hot pixel corrections, color conversion, gamma and LUT are now integrated at the hardware level for DCC3240 models. This reduces PC load and further enhances color reproduction quality significantly. • An event/message indicates when the transfer speed changes from USB 2.0 to USB 3.0 and vice versa.	-
The `is_Measure()` function allows the measurement of the image sharpness in the current image. The sharpness is returned as a relative value. For this function a new camera demo is added.	`is_Measure()`
New color formats are added: • `IS_CM_RGB12_PACKED` • `IS_CM_RGBA12_PACKED` • `IS_CM_BGR12_PACKED` • `IS_CM_BGRA12_PACKED` • `IS_CM_RGB8_PLANAR` The following formats are renamed and the previous formats are moved to the `uc480_deprecated.h` header file: • `IS_CM_BGR10V2_PACKED` (new `IS_CM_BGR10_PACKED`) • `IS_CM_RGB10V2_PACKED` (new: `IS_CM_RGB10_PACKED`) • `IS_CM_BGR555_PACKED` (new: `IS_CM_BGR5_PACKED`) The following formats are moved to the `uc480_deprecated.h` header file as they are identical to existing formats: • `IS_CM_BAYER_RG8` (now: `IS_CM_SENSOR_RAW8`) • `IS_CM_BAYER_RG12` (now: `IS_CM_SENSOR_RAW12`) • `IS_CM_BAYER_RG16` (now: `IS_CM_SENSOR_RAW16`)	`is_SetColorMode()` Color and memory formats
New camera demos: • Simultaneous opening of multiple cameras and sending a single software trigger (Multi-camera demo) • Measuring of the image sharpness in an AOI of the current image (Measure sharpness demo)	See separate manual for the uc480 samples
In the `is_AOI()` function the `IS_AOI_MULTI_MODE_AXES` parameter was renamed to `IS_AOI_MULTI_MODE_X_Y_AXES`. The old parameter was moved to the `uc480_deprecated.h` header file.	`is_AOI()`

 © 2015 Thorlabs Scientific Imaging

Older versions

See the History of uc480 Software Versions and History of uc480 API functions chapters.

2 Camera Basics

This chapter explains the basics of DCx Camera technology.

- Operating modes
- Image display modes
- Sensor
- Reading out partial images
- Digitizing images
- Camera parameters
- Firmware and camera start-up
- Digital inputs/outputs
- USB interface

2.1 Operating Modes

DCx Cameras support the following operating modes:

- Freerun mode
- Trigger mode
- Standby

2.1.1 Freerun Mode

In freerun mode, the camera sensor captures one image after another at the set frame rate. Exposure and readout/transfer of the image data are performed in parallel. This allows the maximum camera frame rate to be achieved. The frame rate and the exposure time can be set separately. The captured images can be transferred one by one or continuously to the PC. If trigger mode is active, you need to disable it before activating freerun mode.

Note

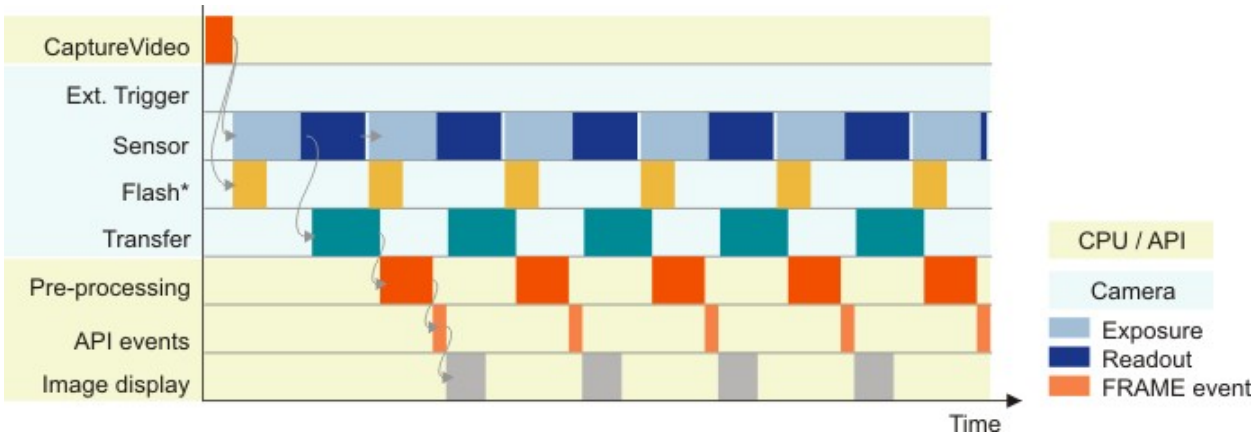
Note on the schematic diagrams: These illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera model and settings. The pre-processing time depends on the API functions you are using (e.g. color conversion, edge enhancement).

For more information on flash timing see the Digital In-/Output (Trigger/Flash) chapter.

Continuous mode (live mode)

Images are captured and transferred continuously. You can use the DCxCamera flash outputs.

 © 2015 Thorlabs Scientific Imaging



Freerun mode (live mode)

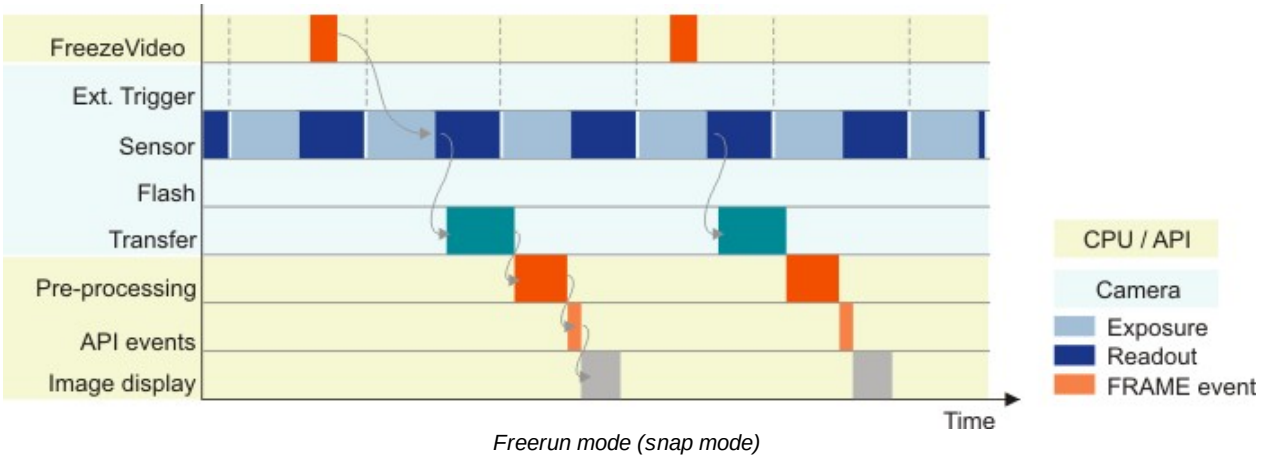
* Flash function optional. See also Digital in-/output (trigger/flash).

Note

In freerun mode the flash function starts with the second image as the setting of the flash timing depends on the finish of the first image. If you change the flash timing during operation, the freerun mode will restart. Therefore the first image after the change is black.

Single frame mode (snap mode)

The next image exposed by the sensor will be transferred. In this mode, flash is not making sense (only manually).



Freerun mode (snap mode)

See also:

- Basics: Shutter methods
- Basics: Trigger mode
- Basics: Applying new parameters

Programming:

- Capture modes

2.1.2 Trigger Mode

In trigger mode, the sensor is on standby and starts exposing on receipt of a trigger signal. A trigger event can be initiated by a software command (software trigger) or by an electrical signal via the camera's digital input (hardware trigger).

This chapter describes the different trigger modes you can use with the DCx Cameras. To choose a mode, go to the Settings icon in the ThorCam application or use the API command.

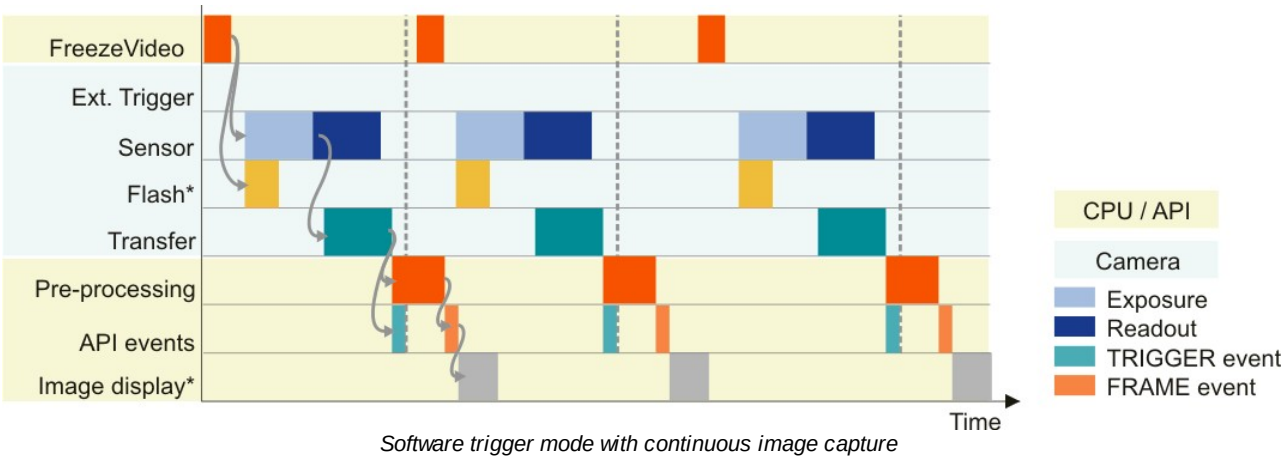
Note

Note on the schematic diagrams: These illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera model and settings. The pre-processing time depends on the API functions you are using (e.g. color conversion, edge enhancement).

For more information on flash timing see the Digital In-/Output (Trigger/Flash) chapter.

Software trigger mode

When this mode is enabled, calling the "Snap" function triggers the capture of an image, which is then transferred to the PC. If you call the "Live" function in this mode, the image capture is triggered continuously and images are transferred continuously.



Software trigger mode with continuous image capture

* Optional flash function. See also Digital input/output (trigger/flash)

Hardware trigger mode

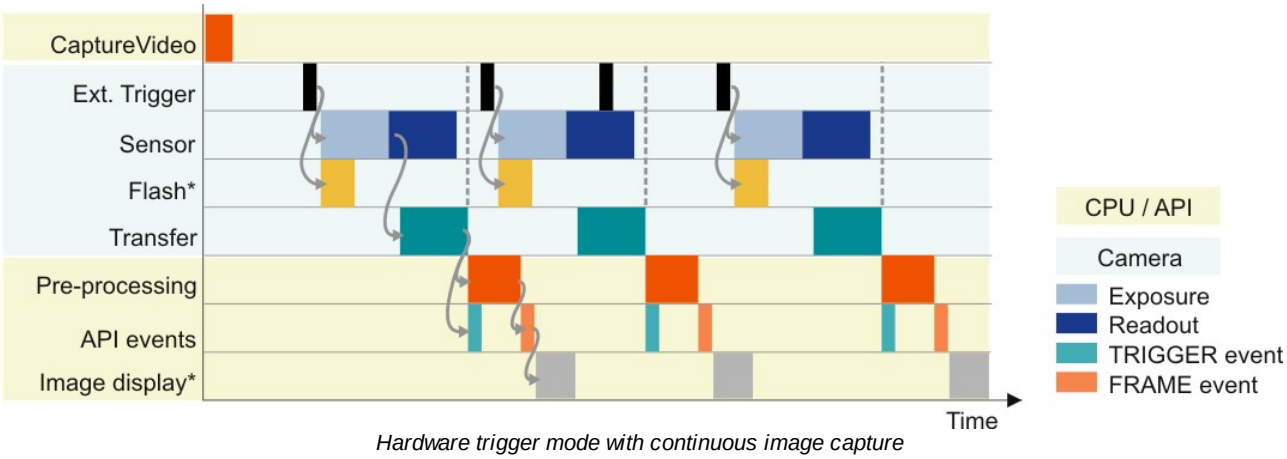
When this mode is enabled, calling the `is_FreezeVideo()` (Snap) function makes the camera ready for triggering just once. When the camera receives an electrical trigger signal, one image is captured and transferred.

If you call the `is_CaptureVideo()` (Live) function, the camera is made ready for triggering continuously. An image is captured and transferred each time an electrical trigger signal is received; the camera is then ready for triggering again (recommended procedure).

Attention

When you use triggered image capture, the camera is only ready to process the next trigger signal after completion of the data transfer to the PC. Trigger events that occur during image exposure or data transfer are ignored. An internal counter records the number of ignored trigger events and can be read out from the PC.

 © 2015 Thorlabs Scientific Imaging



Hardware trigger mode with continuous image capture

* Optional flash function. See also Digital input/output (trigger/flash)

Frame rate in trigger mode

With many sensors, the maximum frame rate is lower in trigger mode than in freerun mode because these sensors expose and transfer sequentially. Which frame rate is possible in trigger mode therefore depends on the exposure time. The time required for capturing a frame in trigger mode can be approximated with the following formula:

$$t_{capture} = \text{Current exposure time} + \left(\frac{1}{\text{max.frame rate}}\right)$$

Example: At the maximum exposure time, the frame rate is about half as high as in freerun mode; at the minimum exposure time, the frame rate is about the same.

Freerun synchronization

This mode is currently not supported by DCx Cameras.

See also:

- Basics: Freerun mode
- Basics: Digital input/output (trigger/flash)
- ThorCam: Settings > Trigger

Programming:

- Image capture modes: Trigger

2.1.3 Standby

DCx Cameras can be set to a power-saving standby mode. Standby mode switches off the sensor of CMOS cameras and the timing board of CCD cameras. The camera remains open in the software.

In standby mode, the camera cools down and the number of hot pixels visible when longer exposure times are used is reduced.

Standby is the default state when the camera is not open in the software. When you open the camera or switch to a different mode (freerun or trigger mode), the camera wakes up from standby mode.

Note

In standby mode, you can continue to use the camera's digital inputs or outputs.

2.2 Image Display Modes

The uc480 driver provides different modes for displaying the captured images on Windows systems. We recommend using the Bitmap mode or the Direct3D functions, depending on your specific application.

For a list of API functions for image display see How to proceed: Image display.

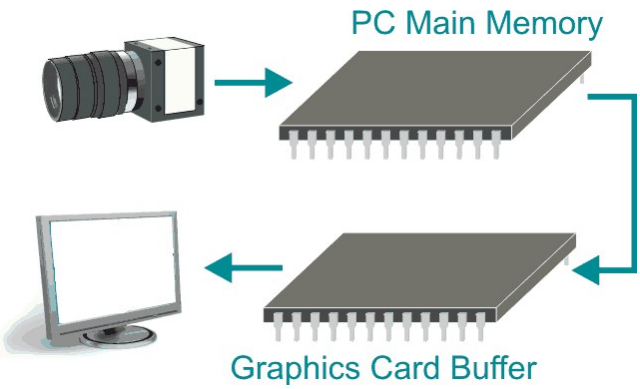
Attention

The "DirectDraw BackBuffer" and "DirectDraw Overlay Surface" display modes are obsolete. Please use the Direct3D functions instead (see also Obsolete functions).

1. Bitmap mode (Device Independent Bitmap, DIB)

In Bitmap mode, images captured by the DCxCamera are written to the random access memory of the PC. Programming the image display is up to the user. The application software uses the `is_RenderBitmap()` function to initiate the image display by the graphics card. This may result in a slightly higher CPU load as compared to the Direct3D display.

The advantage of Bitmap mode is that it is compatible with all graphics cards and that image data in the memory is directly accessible. Programming of overlay functions is up to the user. Since the operating system controls the image display, the image may be completely or partly overlapped by other windows and dialog boxes.



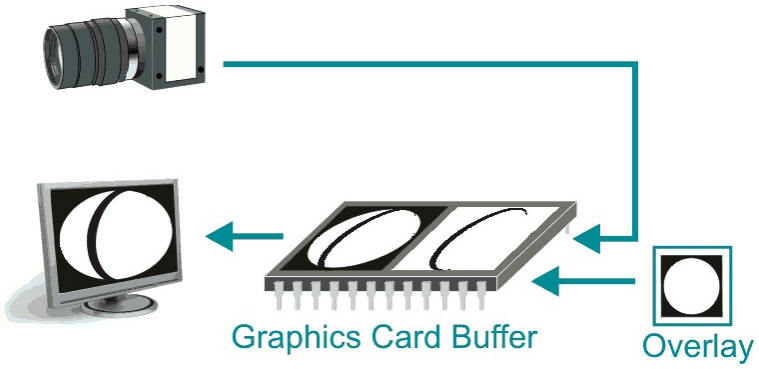
How the Bitmap mode works

2. Direct3D mode (only under Windows with DirectX)

In this mode, the uc480 driver writes the image data to the invisible area of the graphics card. This process runs automatically and does not have to be controlled by the application software. It requires an installed Direct3D driver, sufficient memory on the graphics card and Direct3D function support by the graphics card (see System requirements). For this purpose, graphics cards generally provide better performance than graphics chips integrated on the mainboard. In Direct3D mode, the CPU load may be lower than in Bitmap mode. You can display overlay data and also scale the video image.

The Direct3D mode and the overlay functions can be configured using the `is_DirectRenderer()` API function.

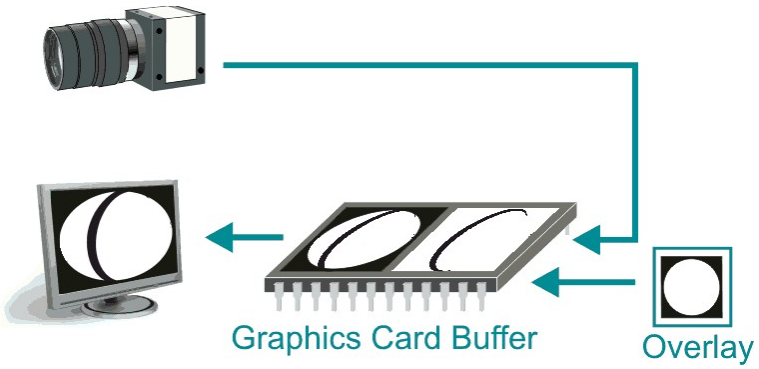
© 2015 Thorlabs Scientific Imaging



How the Direct3D mode works

OpenGL mode

OpenGL stands for Open Graphics Library and it is an interface specification for graphics hardware. Unlike Direct3D OpenGL is not only available under Windows but also under Linux and Mac OS, if OpenGL is supported by the graphics hardware. There are several implementations of OpenGL, such as e.g. NVIDIA® or AMD/ATI. The implementations are always dependent on the graphics card manufacturer.



How the OpenGL mode works

Comparison of the display modes

The following table illustrates the major differences between the display modes:

	Bitmap mode	Direct3D mode	OpenGL mode
Graphics card requirements	Low. No special graphics hardware required. Runs on all systems.	High. Graphics card has to support Direct3D. Does not run on all systems.	High. Graphics card has to support OpenGL.
Operating system	Windows, Linux	Only Windows with DirectX	Cross-platform
Programming effort	Greater. Memory management, event handling and display performed by the application.	Low. Memory management, event handling and display performed by DirectX.	High. OpenGL itself does not provide functions for opening windows or reading files. However, there are related libraries, e.g. GLUT.
CPU load	Slightly increased by copying of data.	Low. Display performed by graphics card.	Low. Display performed by graphics card.
Overlay functions	Not available. A simple overlay can be	Integrated. Complex overlays can be	Integrated.

	programmed by the user.	displayed without flicker.	
Access to image memory	Direct access possible. Image data already provided in user memory.	Possible using Steal Mode. Single images can be copied to the user memory.	Direct access to graphics card and image memory.

© 2015 Thorlabs Scientific Imaging

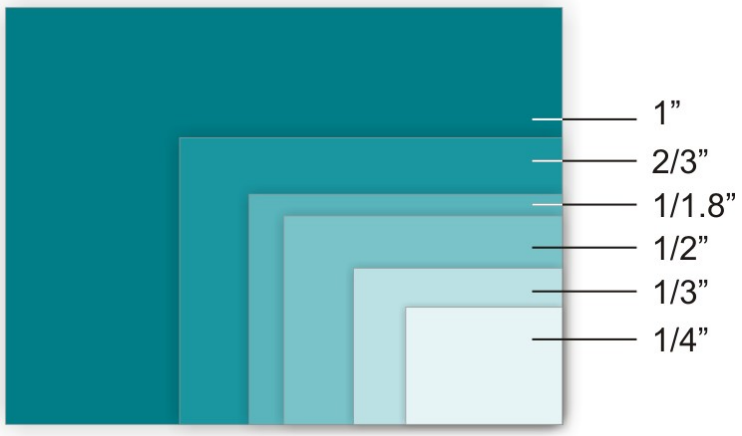
2.3 Sensor

- Sensor sizes
- Micro lenses
- Color filter (Bayer filter)
- Hot pixels
- Shutter methods
- Line scan mode

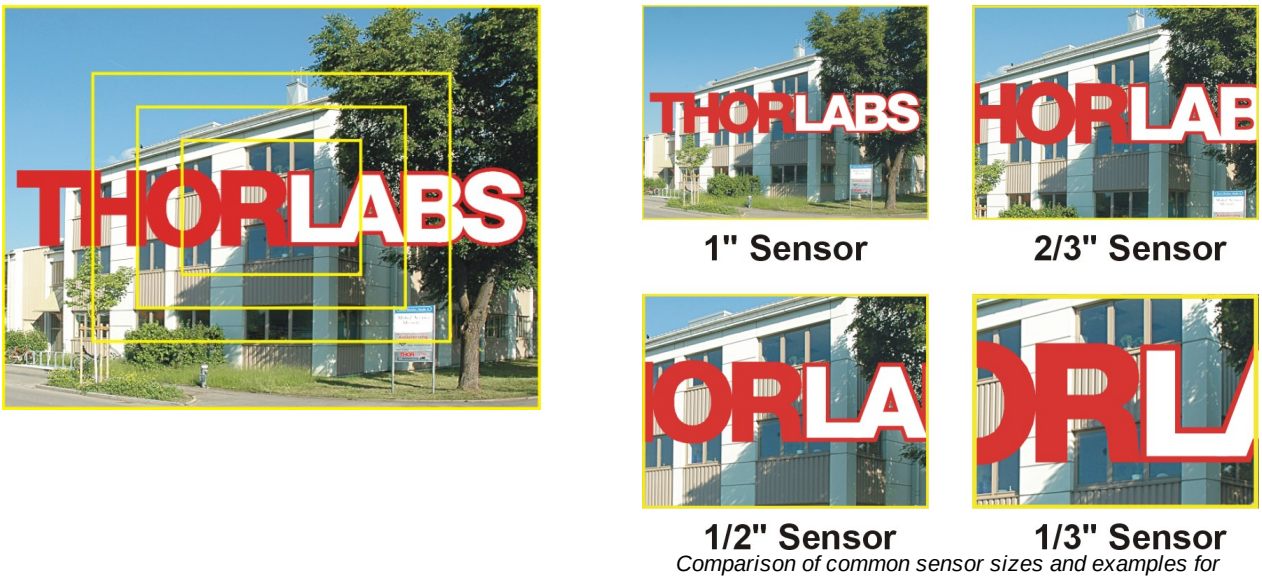
2.3.1 Sensor Sizes

The size of a digital camera sensor is usually specified in inches. However, the specified value does not indicate the actual size of the active sensor area. The sensor size specifications date back to the formerly used tube systems: The curvature of the imaging surface of the camera tube caused distortions to the display, reducing the usable capture area of a 1” tube to a rectangle with a diagonal of 16 mm.

With the introduction of the semiconductor sensor technology, the dimensional specifications were taken over from tube systems. For this reason, a sensor whose active area diagonal measures 16 mm is specified as a 1-inch sensor. The following illustrations show the most common sensor sizes. The diameter in inch multiplied with 2/3 equals approximately the actual sensitiv area in millimeters.



Common sensor sizes (in inch)



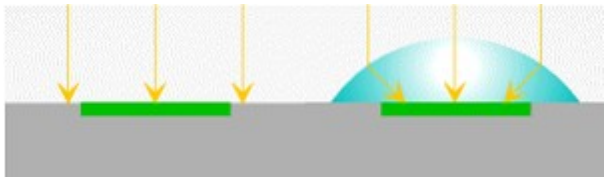
Comparison of common sensor sizes and examples for different fields of view

The size of each single sensor cell (pixel) depends on the size of the active sensor area and the resolution. In general, less pixels over the same sensor area (or a larger sensor area with the same resolution) will result in greater photo sensitivity of the sensor.

2.3.2 Micro Lenses

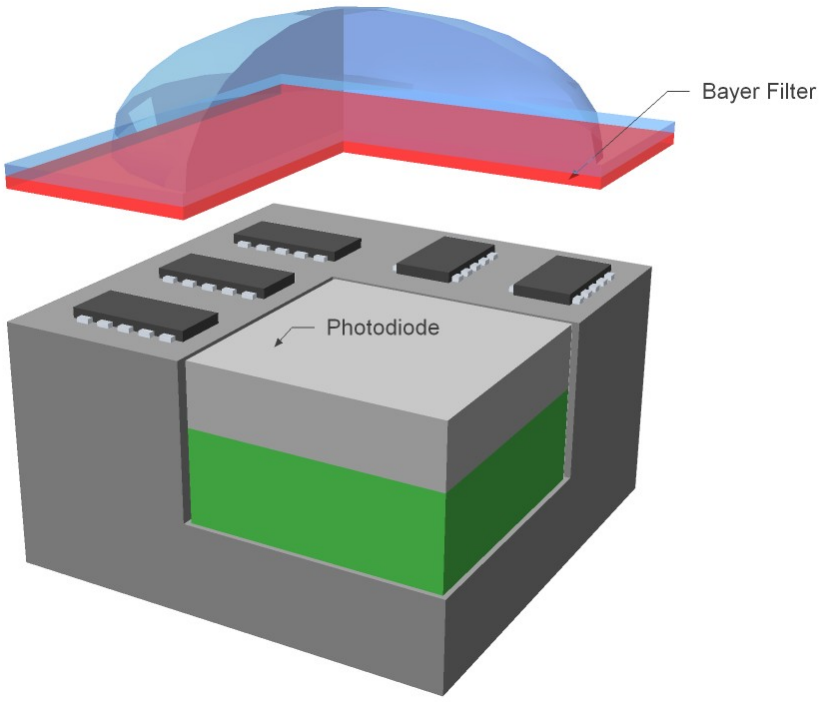
Micro lenses improve the fill factor

The fill factor is the percentage of the pixel area that is exposed to light during exposure. Ideally this would be 100 %. Since other elements are located on the sensor surface besides the light-sensitive photodiodes, this value may be reduced to approx. 30–50 %, depending on the sensor technology. The use of micro lenses compensates for this and increases the fill factor to 90 % or more. Micro lenses collect the light that falls onto a photocell, thus increasing the useable sensor area.



Using micro lenses to increase the effective fill factor

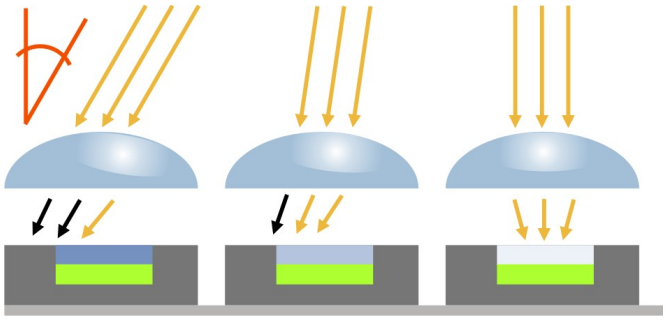
© 2015 Thorlabs Scientific Imaging



CMOS pixel design with Bayer filter (red) and micro lens

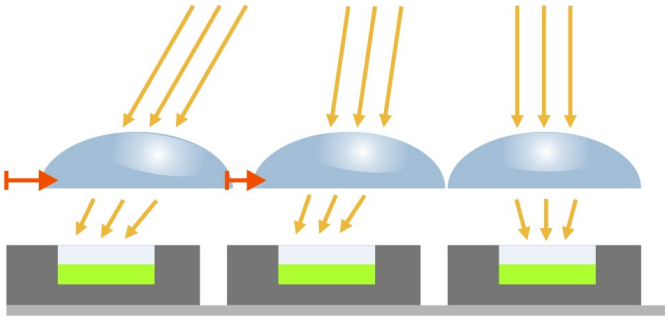
Micro lenses with CRA correction

Some sensors have micro lenses offset to the sensor edge. They compensate for shading created by obliquely incident light. The angle of incident light is called Chief Ray Angle (CRA), the micro lens offset is thus called CRA correction. The amount of micro lens shift is specified in degrees and refers to the micro lenses in the corners of the sensor.



Micro lenses without CRA correction

Image captured without CRA correction shows shading



Micro lenses with CRA correction

Image captured with CRA correction

Note

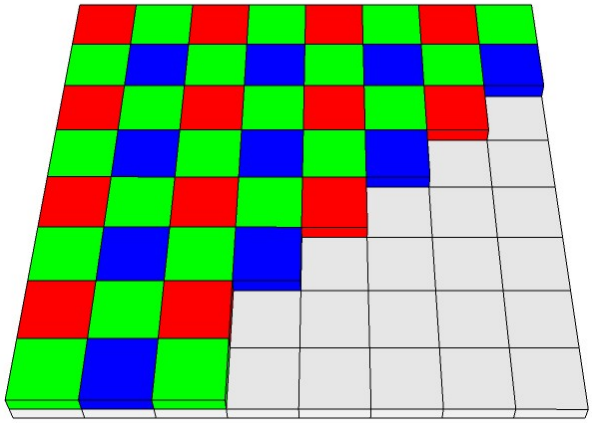
Using parallel light on sensors with CRA correction may cause slight color variations. These may occur, for example, if telecentric lenses are used. The following models are equipped with sensors with offset micro lenses:

- DCC1240x/DCC3240x
- DCC1645C

© 2015 Thorlabs Scientific Imaging

2.3.3 Color Filter (Bayer filter)

For technical reasons, digital image sensors can only detect brightness information, but no color information. To produce color sensors, a color filter is applied to each photocell (pixel). The arrangement of the color filters is illustrated in the following figure. Two out of every four pixels have a green filter, one pixel has a red filter and one has a blue filter. This color distribution corresponds to the color sensitivity of the human eye, and is called the Bayer filter pattern. With the help of the Bayer pattern the correct brightness and color information can be calculated for each pixel. Full sensor resolution is retained.



Bayer RGB filter pattern

Bayer conversion

A Bayer conversion, also referred to as de-Bayering, is carried out to determine the color information from the raw sensor data (raw Bayer). By default all DCx Cameras transmit the image data to the PC in raw Bayer format. The PC then uses the functions of the uc480 API to convert the image data to the color format you need for displaying or further processing the data.

To convert the colors, a filter mask moves over the image and calculates a color value for each pixel from the surrounding pixels. The uc480 API provides two filter masks that differ in image quality and CPU load.

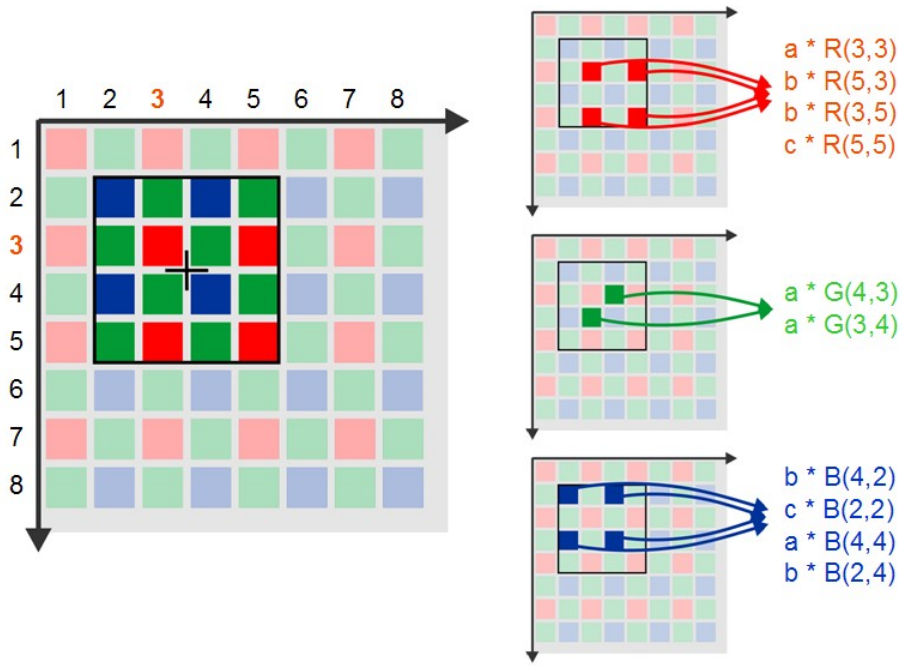
- Normal Quality (Mode `IS_CONV_MODE_SOFTWARE_3x3/IS_CONV_MODE_HARDWARE_3x3`)
  A smaller filter mask is used for conversion. This algorithm has a low load on the CPU. The filter's averaging function may cause a slight blur. Noise is reduced. This filter is recommended for image processing tasks.
- High Quality (Mode `IS_CONV_MODE_SOFTWARE_5x5`)
  A large filter mask is used for conversion. This algorithm offers very accurate color positioning and an increased level of detail. The CPU load is higher than with the normal filter. This filter is recommended for visualization applications.

Note

Software conversion with high quality should only be used for sensors whose green pixels have the same sensitivity. This applies to the following sensors:

- DCU223C / DCU224C
- DCC1240C, DCC3240C

For all other sensors, we recommend using the standard filter mask.



Bayer conversion using the standard mask

See also:

- Color conversion: `is_SetColorConverter()`
- ThorCam: Settings > Camera > Pixel Data Format

© 2015 Thorlabs Scientific Imaging

2.3.4 Hot Pixels

Definition

Hot pixels (or in a broader sense, defective pixels) are pixels that do not react linearly to incident light – or do not react at all. They occur for various reasons, such as contamination during sensor production or sensor age, and with both CCD and CMOS sensors. CCD sensors generally have fewer hot pixels than CMOS sensors under the same operating conditions. With darkened sensors and prolonged exposure times, hot pixels are visible as individual bright dots in the image. The following factors promote the occurrence of hot pixels:

- Long exposure times
- High gain settings
- High sensor operating temperature

Hot pixels detected in a monochrome camera

Hot pixels detected in a color camera

Hot pixel correction

During the manufacture of our cameras, all sensors that will be used in DCx Cameras are checked for hot pixels. In the process, images are taken with a darkened sensor and long exposure times. Pixels with a brightness higher than a specific value are classified as hot pixels. A list of the

coordinates of each hot pixel is stored in the camera EEPROM. The hotpixel correction is done in the uc480 driver. However, some sensors also provide an internal hotpixel correction.

The maximum number of hot pixels stored in a DCx camera is:

DCx model	max. hot pixels stored
DCC1240x, DCC1545M, DCC1645C, DCC3240x (CMOS)	768
DCU223x, DCU224x (CCD)	20

How many hot pixels are on the camera's internal list depends above all on the defined threshold values. It is not an indication of the quality of the sensors used.

When you enable the "Hotpixel correction" function in the DCx software, the software automatically corrects the hot pixels in the captured image by calculating the average from the brightness value of two neighboring pixels. When using color sensors, the hot pixel is corrected with the appropriate color in raw Bayer format, i.e. before color conversion. The correction does not work with activated subsampling and binning factors greater than 2x.

Note

The sensors are tested during manufacturing also for cold pixel and dead pixels. Sensors with dead pixel clusters (more than two neighboring defective pixels of the same color) are rejected by our quality control. When the camera is operated in very warm ambient conditions, other defective pixels can occur, however.

Defining additional hot pixels

If additional hot pixels occur during use of the camera, you can add them to the camera's internal hot pixel list. To do this, use the API function given below.

See also:

- ThorCam: Hot pixel correction
- uc480 Hotpixel Editor
- Programming: `is_HotPixel()`

 © 2015 Thorlabs Scientific Imaging

2.3.5 Shutter Methods

- Global shutter
- Rolling shutter
- Rolling shutter with global start

Note

Note on the schematic diagrams: These illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera model and settings.

For more information on flash timing see the Digital In-/Output (Trigger/Flash) chapter.

General

The image is recorded in the sensor in four phases:

- Reset pixels of the rows to be exposed
- Exposure of pixel rows
- Charge transfer to sensor
- Data readout

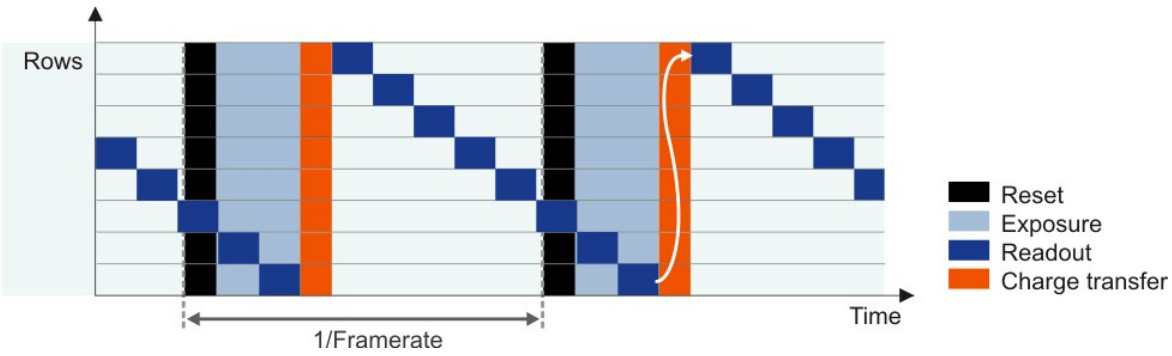
The sensor cells must not be exposed during the readout process. The sensors of the DCx Cameras have no mechanical shutters, but work with electronic shutter methods instead. Depending on the sensor type, either the rolling shutter method or the global shutter method is used.

Global shutter

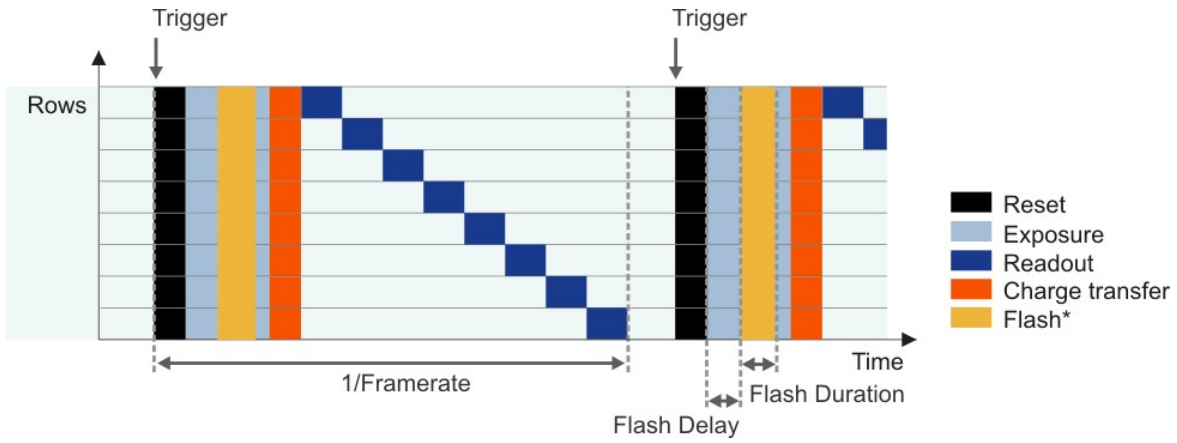
On a global shutter sensor, all pixel rows are reset and then exposed simultaneously. At the end of the exposure, all rows are simultaneously moved to a darkened area of the sensor. The pixels are then read out row by row.

Exposing all pixels simultaneously has the advantage that fast-moving objects can be captured without geometric distortions. Sensors that use the global shutter system are more complex in design than rolling shutter sensors.

All CCD sensors as well as some CMOS sensors use the global shutter method.



Global shutter sensor in live mode



Global shutter sensor in trigger mode

* Optional flash function. The start time and duration are defined by the flash delay and duration parameters (see also Camera settings: I/O).

Rolling shutter

With the rolling shutter method, the pixel rows are reset and exposed one row after another. At the end of the exposure, the lines are read out sequentially. As this results in a time delay between the exposure of the first and the last sensor rows, captured images of moving objects are distorted.

Example for the rolling shutter effect with a moving car

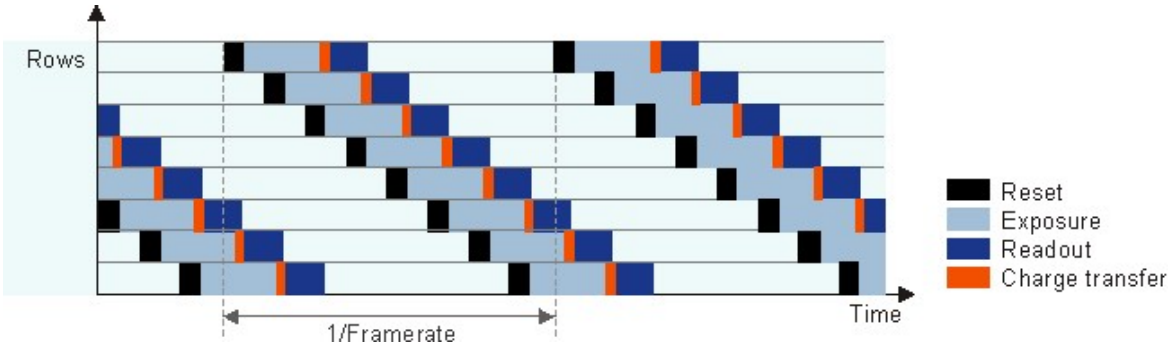
To counteract this effect, the DCx Camera software provides a global flash window where you set the time by which flash activation is delayed. You can also specify the flash duration. This allows implementing a global flash functionality which exposes all rows of a rolling shutter sensor simultaneously.

Rolling shutter sensors offer a higher pixel density compared to global shutter CMOS sensors. The rolling shutter system is used in DCC Cameras with high-resolution CMOS sensors.

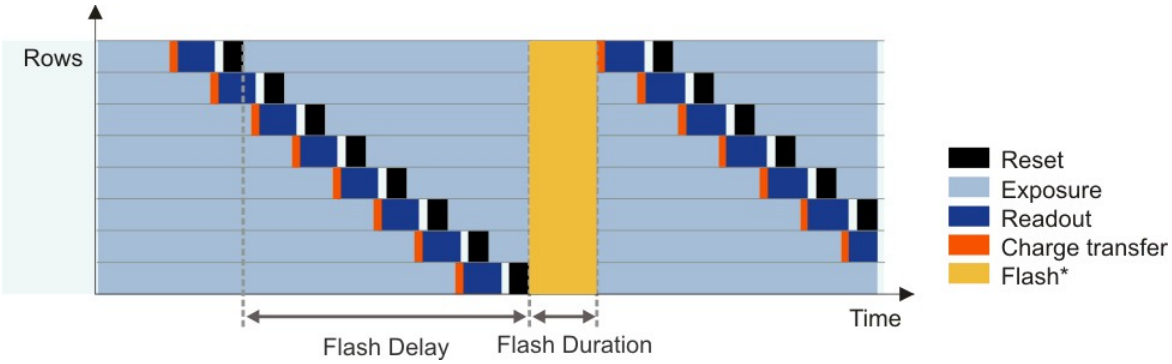
Note

Some CMOS senors with global shutter can be operated also with rolling shutter. The operation in the rolling shutter mode is used to reduce the image noise. This function is only supported from the camera models DCC1240x/DCC3240x.

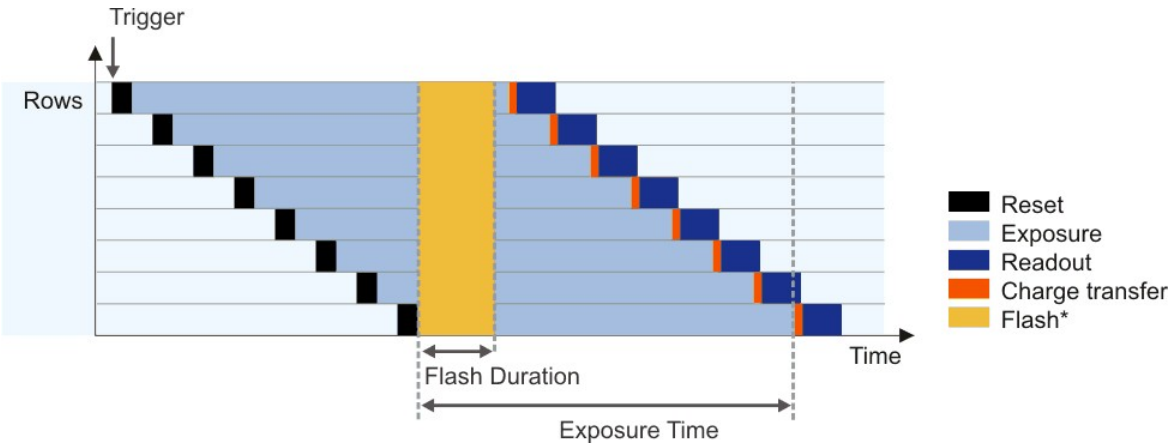
© 2015 Thorlabs Scientific Imaging



Rolling shutter sensor in live mode



Rolling shutter sensor in live mode with global flash window

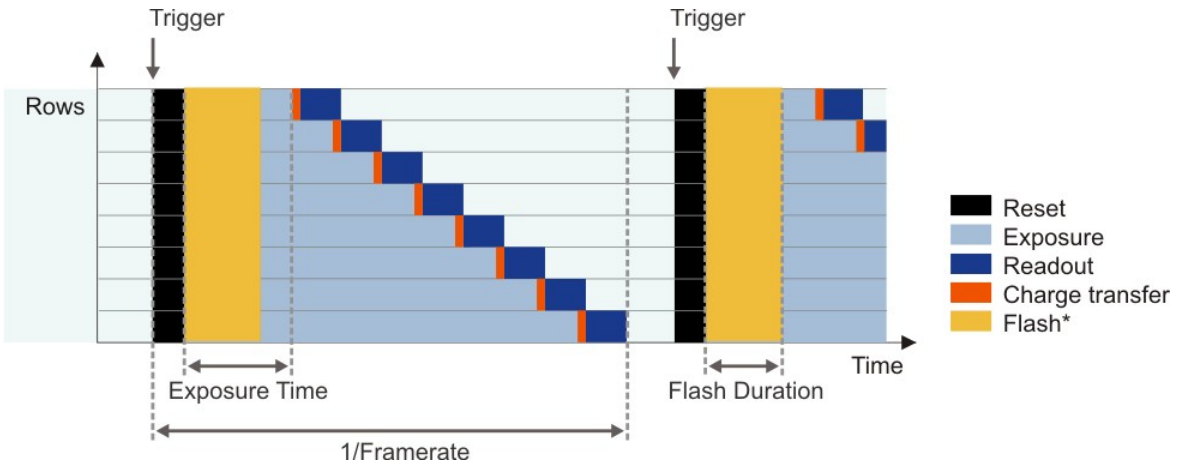


Rolling shutter sensor in triggered mode with global flash window

* Optional flash function. The start time and duration are defined by the flash delay and duration parameters (see also Camera settings: I/O).

Rolling shutter with global start

Some rolling shutter sensors also provide a global start mode, which starts exposure of all rows simultaneously (see illustration). For best results, use a flash for this mode. No light is allowed to fall on the sensor outside the flash period because otherwise the image brightness will be distributed unevenly.



Rolling shutter sensor in trigger mode with global start function

* Optional flash function. The start time and duration are defined by the flash delay and duration parameters (see also Camera settings: I/O).

© 2015 Thorlabs Scientific Imaging

2.3.6 Line Scan Mode

Area scan sensor (matrix)

The sensors of area scan cameras have a matrix of many (usually between several hundred and several thousand) rows and columns of pixels. State-of-the-art area scan sensors use only square pixels with a consistent pixel pitch.

Area scan cameras are suitable for applications in which stationary or moving objects should be captured as completely as possible in one image capture.

Line scan mode

In some applications, however, it is necessary to read out and transfer only one sensor line instead of the entire sensor area. This applies, for example, to endless web inspection systems. These systems often use line scan cameras for this reason. Their sensors have only one pixel row, which they can read out at very high speeds in the kilohertz (kHz) range. Some DCxCamera models have area scan sensors that optionally also offer a line scan mode. This mode can read out any pixel row of the sensor at high speed.

There are two line scan modes to distinguish:

- **Fast line scan**

  In this mode, the sensor achieves very high line scan rates. Several hundred to thousand lines are combined and transferred in one frame. The camera can be triggered on the beginning of a frame, but not on each individual line. You can choose any line of the area scan sensor for the line scan mode. Color images are not supported in this mode because Bayer color sensors need at least two neighboring lines for color calculation.

- **Triggered line scan**

  In this mode, the sensor achieves lower line scan rates than in fast line scan mode. The camera can be triggered on each individual line. Several hundred to thousand lines are combined and transferred in one frame.
  Color images are possible in this mode because Bayer color sensors can use two lines.

Note

The line scan mode is currently only supported by the monochrome DCC1240M and DCC3240M,N models in form of the fast line scan mode. The triggered line scan mode is not supported by any camera model yet.

See also:

- ThorCam: Settings > Shutter

Programming:

- Function: `is_DeviceFeature()`

2.4 Reading Out Partial Images

The camera sensors have defined resolutions which are given as the number of pixels (width x height). However, for some applications it may be necessary to read out only a selected part of the sensor area or to reduce the local resolution. For this purpose, the DCx Cameras provide various functions:

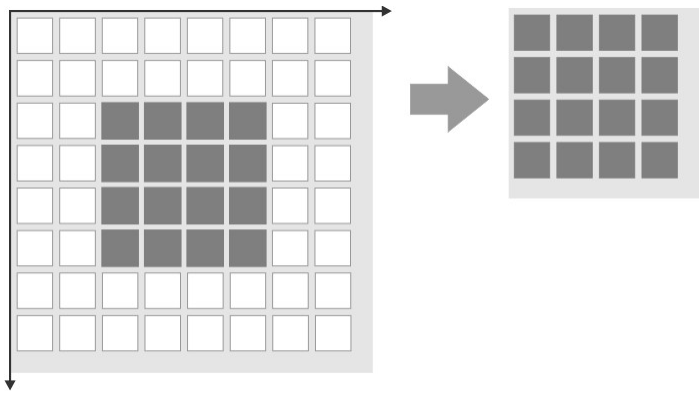
- Area of interest (AOI)
- Subsampling (skipping) pixels
- Binning (combining) pixels

These functions reduce the amount of data to be transferred and thus allow you to increase the frame rate considerably, depending on the camera model.

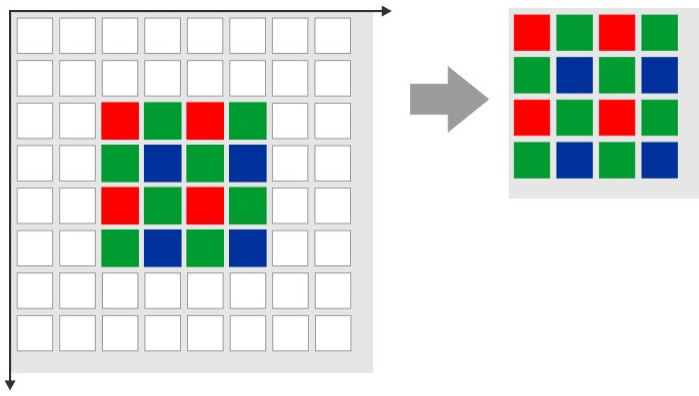
2.4.1 Area of Interest (AOI)

Using this function, you can set the size and position of an area of interest (AOI) within an image. In this case, only data included in this AOI will be read out and transferred to the computer. The smaller partial image enables the camera to use a higher frame rate.

For information on the AOI position grid and the frame rates that your camera model can achieve with AOI, see the model specifications in the Camera and sensor data chapter.



AOI readout on monochrome sensors



AOI readout on color sensors

Note

Step widths for AOI definition (position grid): The available step widths for the position and size of image AOIs depend on the sensor. The values defining the position and size of an AOI have to be integer multiples of the allowed step widths.

For details on the AOI grids of the individual camera models, please see Camera and sensor data

 © 2015 Thorlabs Scientific Imaging

and click a camera model.

Please note that, after defining an AOI, the resulting image may be darker if the camera cannot maintain the originally set exposure time due to the increased frame rate.

Multi AOI

The Multi AOI function allows defining more than one AOI in an image and transferring these AOIs all at the same time. Only DCC1240x models support this feature. In the Multi AOI mode you can define two or four AOIs in one image and transfer them simultaneously. The AOIs are positioned side by side or one below the other, and share the same X or Y axis. This feature is not supported in ThorCam at this time.

 © 2015 Thorlabs Scientific Imaging

Sequence AOI mode

Apart from the multi AOI mode, DCC1240x and DCC3240x also support the sequence AOI mode.This mode allows to define up to four AOIs, which need to have the same size but may differ in position, exposure time or gain settings. This feature is not supported in ThorCam at this time.

See also:

- ThorCam: Crop to Region of Interest
- Programming: `is_AOI()`

2.4.2 Subsampling

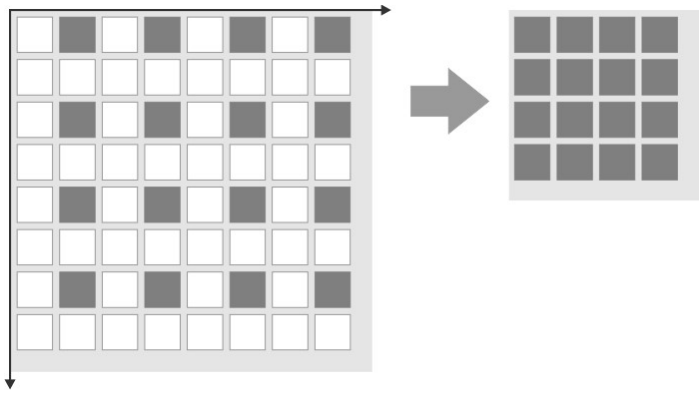
Subsampling is a technique that skips multiple sensor pixels when reading out image data. This reduces the amount of data to be transferred and enables higher camera frame rates. The captured image has a lower resolution but still the same field of view compared to the full-resolution image. This mode can be used as a fast preview mode for high-resolution cameras.

Color subsampling as performed by most color sensors skips pixels while maintaining colors (see illustration). For some monochrome sensors, the camera also performs color subsampling, resulting in slight artifacts.

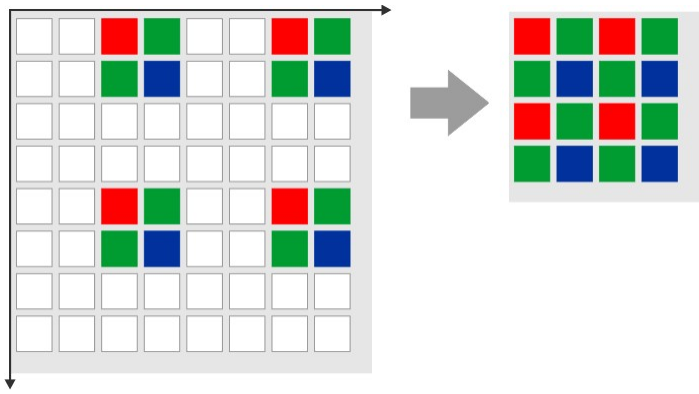
Monochrome sensors and some color sensors ignore the Bayer pattern and the color information gets lost (mono subsampling).

Depending on the model, DCx Cameras support different subsampling factors. Subsampling of horizontal and vertical pixels can be enabled independently.

The Camera and sensor datas chapter lists the subsampling methods and factors supported by each camera model.



Subsampling on monochrome sensors



Subsampling on color sensors

© 2015 Thorlabs Scientific Imaging

2.4.3 Binning

Binning is a function that averages or adds multiple sensor pixels to obtain a single value. This reduces the amount of data to be transferred and enables higher camera frame rates. The captured image has a lower resolution but still the same field of view compared to the full-resolution image. This mode can be used as a fast preview mode for high-resolution cameras.

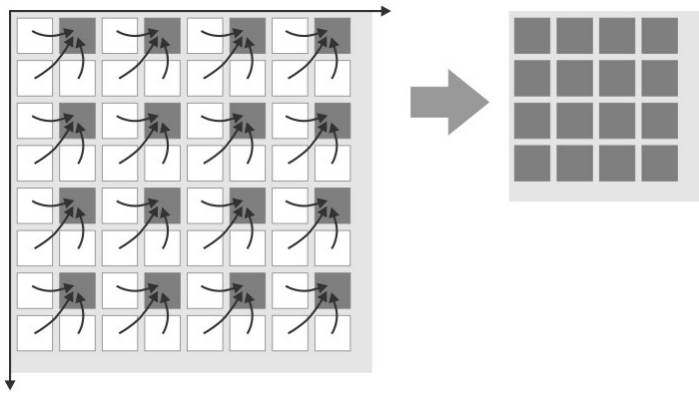
Color binning, as performed by most color sensors, combines only pixels of the same color (see also the Color filter (Bayer filter) chapter). For some monochrome sensors, the camera also performs color binning, resulting in slight artifacts.

Most monochrome sensors and some color sensors combine neighboring Bayer pattern pixels; in this case, the color information gets lost (mono binning).

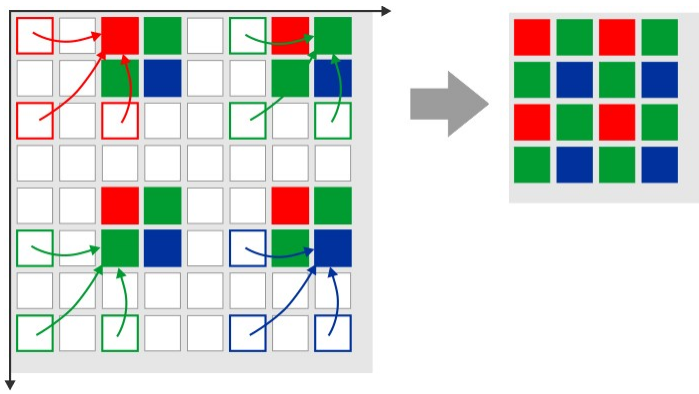
With CCD sensors, binning makes the images brighter because the pixel values are added up. With CMOS sensors, pixel values are usually averaged; this reduces image noise.

Depending on the model, DCx Cameras support different binning factors. Binning of horizontal and vertical pixels can be enabled independently.

The Camera and sensor data chapter lists the binning methods and factors the individual camera models support.



Binning on monochrome sensors



Binning on color sensors

2.5 Digitizing Images

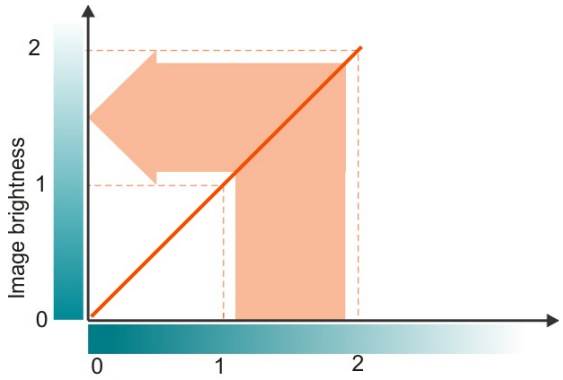
- Characteristics and LUT
- Bit depth and digital contrast adjustment

2.5.1 Characteristics and LUT

When perceiving or imaging a scene, the form of the imaging characteristic is crucial for displaying the differences in brightness. With image processing (e.g. applications such as edge detection and character recognition), linear characteristics are generally required. The human eye, on the other hand, perceives differences in brightness based on a logarithmic characteristic, which often approximates a gamma characteristic in practice. All three forms will be shown in the following.

Linear characteristic

If a system (e.g. a camera with a conventional CCD sensor) yields double the output value for double the brightness, the system features a linear characteristic:



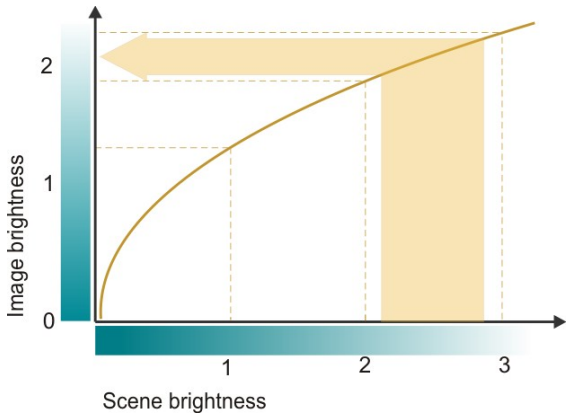
Imaging with linear characteristic

Gamma characteristic

Gamma characteristics (or gamma curves) are named after the Greek formula symbol γ. Gamma curves are power functions of the form

$$y = x^{\frac{1}{\gamma}}$$

and are often used in photography or image display on computer screens. A gamma value of 1 generates a linear characteristic again. A curve with the value γ = 2.2 used for computer screens is shown in the figure below.



Imaging with gamma characteristic

© 2015 Thorlabs Scientific Imaging

Such a gamma characteristic brightens dark areas of an image, which corresponds more to the perception of the human eye. In light areas of an image, the differences in brightness are condensed for this.

Image with linear characteristic

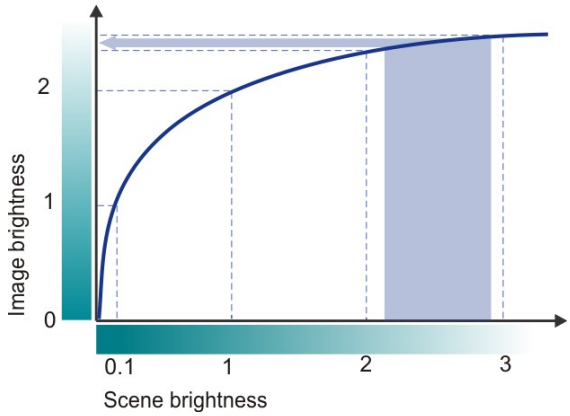
Image with gamma characteristic

Logarithmic characteristic

The effect of the logarithmic characteristic is even stronger. Here, the characteristic follows the function

$$y = \lg(x)$$

The following diagram illustrates how very large jumps in brightness in light areas of a scene only cause small changes in image brightness. This explains why image sensors with a logarithmic characteristic, in particular, are ideal for imaging scenes with very high dynamic range.

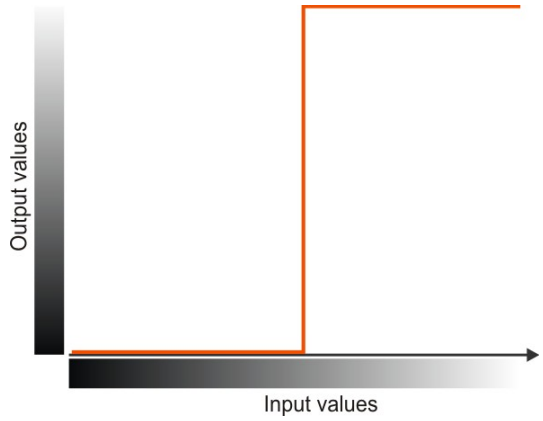


Imaging with logarithmic characteristic

Lookup table (LUT)

With a lookup table (LUT) it is easy to apply characteristic curves to digital images. A LUT is a table which assigns an output value to every possible input value. The figure below shows a LUT which would binarize an image: For an 8 bit image, for example, this LUT would replace all pixels with gray values 0...127 with value 0 and all pixels with gray values 128...255 with value 255.

Using LUTs has the advantage that calculations can be done very fast. Typical applications of LUTs are enhancing image contrast, or gamma characteristics.



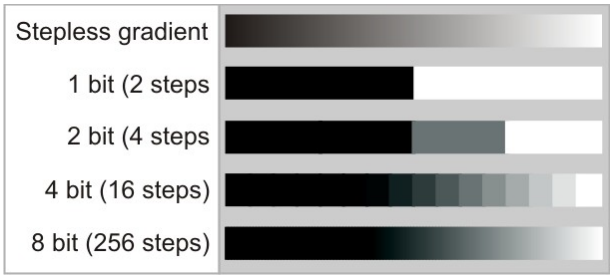
LUT characteristic for binarizing an image

© 2015 Thorlabs Scientific Imaging

2.5.2 Bit Depth and Digital Contrast Adjustment

Digitizing

Image sensor pixels first generate an analog voltage signal proportional to the amount of light that strikes them. The image is digitized for further processing, i.e. the stepless signal is converted to a digital numerical value. The following figure shows this using a gray gradient as an example



Various bit depths using a gray-scale gradient as an example

If the stepless gradient is imaged in a digital range in 2 bits, for example, the result is $2^2 = 4$ levels; for 4 bits, it is $2^4 = 16$ levels, and so on. The intermediate brightness values of the original gradient are irreversibly lost after digitization.

With around 200 levels or more, the jumps in brightness can no longer be discerned with the eye, which is why current monitors and digital cameras use 8 bits (256 levels) per color channel (fully adequate for visualization).

Bit depth in image processing

If digital image data undergoes further image processing, a bit depth greater than 8 may be necessary. The computer is able to differentiate between these very fine differences in brightness (no longer discernable by the eye) and process them. This is why industrial cameras often use 12 bits.

Bit depth	Brightness levels
8	$2^8 = 256$
10	$2^{10} = 1024$
12	$2^{12} = 4096$
14	$2^{14} = 16.384$

Note

Greater bit depths require extremely low-noise image sensors, however. As soon as the differences in brightness created by noise are greater than the digitization levels, no further data is gained.

Bit depth by sensors

Platform	USB 2.0	USB 3.0
CMOS sensors	8 bit	10 bit
CCD sensors	8 bit	-

Note

Color formats with a bit depth of more than 8 bits per channel are only supported by USB 3 DCC3240x camera models. Using color formats with higher bit depth increases the bandwidth used by a camera.

Histogram and contrast

The brightness distribution of digital images is represented in a histogram. If an image has optimum contrast, the histogram includes practically all brightness values between 0 and the highest value (255 in 8-bit images). If an image has low contrast, the histogram only includes a small number of the values; the image appears dull:

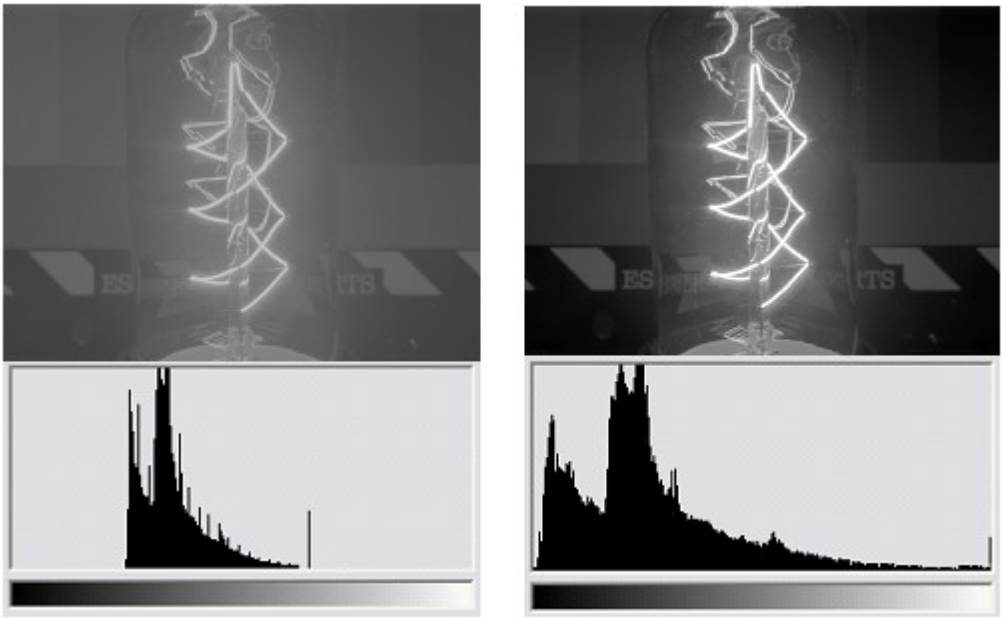


Image capture and histogram with minimal contrast (le.) and with optimum contrast after a contrast adjustment (ri.)

For improved display on the screen or when printed, the histogram can be spread to optimally utilize the possible brightness levels. For this purpose, the dark parts of the image are further darkened via an LUT characteristic and the light parts of the image are brightened. Thus the human eye can better differentiate between the different brightness levels; the image has more contrast.

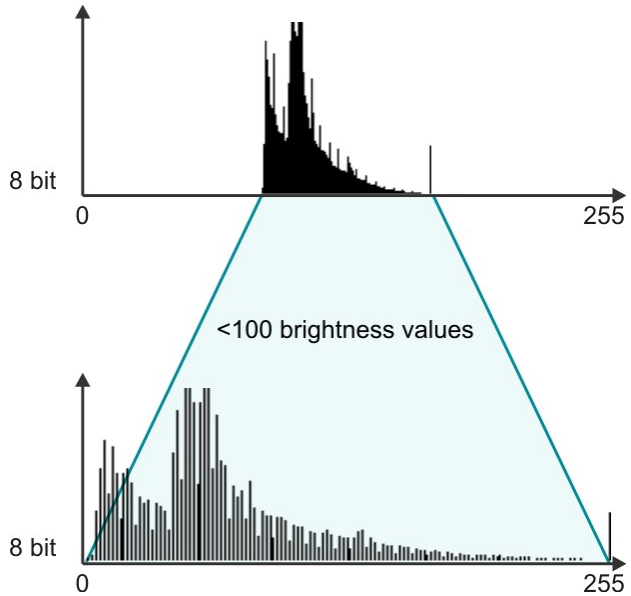
It must be noted, however, that subsequent processing with a computer will not yield more data. Therefore, subsequent contrast adjustment via software is not necessary for use in image processing. The computer can differentiate between the differences in brightness without contrast adjustment.

Advantage of greater bit depth with contrast adjustment

The bit depth in the output image is crucial for contrast adjustment. The following figures illustrate this. In the first example the 8 bit output image contains fewer than 100 brightness levels, as there are no dark or very bright parts. The image is low-contrast.

With a contrast adjustment, the values of the histogram are spread in such a way as to create a contrast-rich image. The fewer than 100 brightness values are now distributed across levels 0 to 255; gaps arise in the histogram and are visible as jumps in brightness in the resulting image.

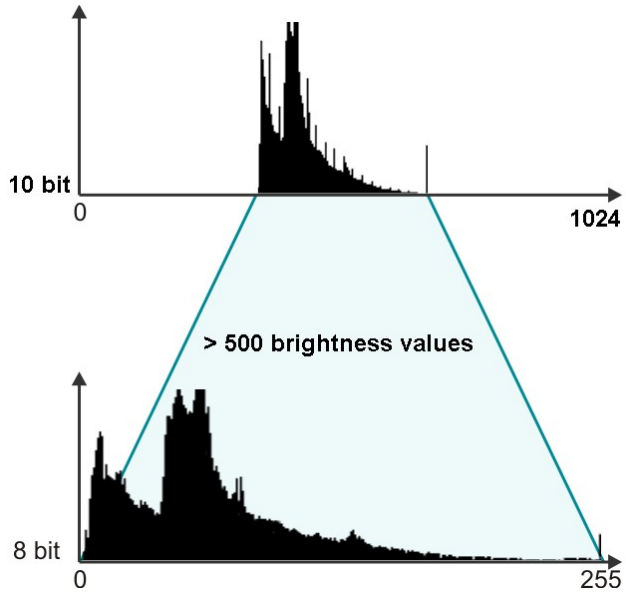
 © 2015 Thorlabs Scientific Imaging



Contrast adjustment with 8 bit output data

The second example shows the same output image with a 10 bit bit depth right at the time of capture. This image also has low contrast, as it features only average brightness values. The greater bit depth allows the brightness values of the image to be imaged over 500 different digital levels, however. The entire histogram includes 1,024 values in the 10 bit image (in contrast to 256 values with 8 bits).

This means that a contrast adjustment can now be made for screen display without a reduction in quality. The 500 values of the output image are distributed over the 256 values of the 8-bit target image in such a way that optimum contrast is the result. The large number of output values means that there are no gaps in the histogram.



Contrast adjustment with 10 bit output data

Note

This type of contrast adjustment can already be done in the camera when an image is digitized in 10 bits and transferred in 8 bit. In this case, optimum utilization of the 8 bit data is also important for image processing.

2.6 Camera Parameters

- Pixel clock, frame rate, exposure time
- Gain and offset
- Automatic image control
- Applying new parameters

2.6.1 Pixel Clock, Frame Rate, Exposure Time

Pixel clock

The basic parameter for camera timing is the pixel clock. It determines the speed at which the sensor cells can be read out.

Attention

We recommend not setting the pixel clock any higher than necessary to achieve the desired frame rate.

An excessive pixel clock can cause delays or transmission errors. If the data is read from the sensor at a higher speed (high pixel clock), you will also need a faster transmission over the data connection. Thus, by controlling the pixel clock, you can also influence the bandwidth required for a camera.

The pixel clock influences the connected load and consequently the temperature inside the camera.

Frame rate

The possible range of settings for the frame rate depends on the currently selected pixel clock. You can select a lower frame rate without changing the pixel clock. To set a higher frame rate, however, you need to increase the pixel clock.

Exposure time

The exposure time depends on the currently selected frame rate and is preset to its reciprocal value. You can select a shorter exposure time without changing the frame rate. To set a longer exposure time, however, you need to reduce the frame rate.

Note

The increments for setting the exposure time depend on the sensor's current timing settings (pixel clock, frame rate). The exposure time values are rounded down to the nearest valid value, if required. For this reason, the actual exposure time can deviate slightly from the exposure time you have selected.

See also:

- ThorCam: Settings > Camera
- `is_PixelClock()`
- `is_SetFrameRate()`
- `is_Exposure()`

© 2015 Thorlabs Scientific Imaging

2.6.2 Gain and Offset

Gain

In digital imaging, a voltage proportional to the amount of incident light is output by the sensor. To increase image brightness and contrast, this signal can be amplified by an analog gain and offset before the digitizing process. The results of analog signal processing are usually better than the results of digital post-processing.

Analog amplification of the read-out pixel values increases overall image brightness and contrast. Depending on the sensor type, a global gain value for all pixels (master gain) or a separate gain value for each color (RGB gain) can be set.

Note

Using Sensor Gain: A signal gain will also result in a noise gain. High gain settings are therefore not recommended.

We suggest the following gain settings:

1. Enable the Gain boost function (`is_SetGainBoost()`).
2. If required, adjust the gain setting with the master gain control.

Note

Linearity of sensor gain: You can set the gain factor in increments from 0 to 100.

- For CCD sensors the gain increases usually not linear but disproportionate.
- For CMOS sensors the gain increases linear. Some sensors have only 32 or fewer levels, so not each step is assigned to a level.

The maximum gain factor settings also vary from sensor to sensor (see Camera- and sensor data).

Offset

Every digital image sensor has light-insensitive cells next to the active image area. These dark pixels are used to measure a reference voltage (black level) which is subtracted from the image signal. This compensates thermally generated voltages on the sensor which would otherwise falsify the signals.

Normally, the sensor adjusts the black level automatically. If the environment is very bright or if exposure times are very long, it may be necessary to adjust the black level manually.

2.6.3 Automatic Image Control

The uc480 driver provides various options to automatically adjust the image capture parameters to the lighting situation. These include:

- Auto exposure shutter (AES)
- Auto gain control (AGC)
- Auto white balance (AWB)
- Auto frame rate (AFR)

The auto functions are used to adjust the average brightness and color rendering of the camera image to their setpoint values, while trying to keep the frame rate at the highest possible value.

All controls are configured using the is_SetAutoParameter() SDK function.

Auto exposure shutter (AES)

The control of the average brightness is preferably achieved by adjusting the exposure, i.e. you set the highest possible exposure time before gain is controlled. The auto exposure feature always

uses the current exposure range which results from the selected pixel clock frequency and the frame rate. You can set separate control range limits for exposure and gain.

Auto gain control (AGC)

The auto gain feature controls the camera master gain in a range from 0-100 %. You can set separate control range limits for exposure and gain.

Auto frame rate (AFR)

With the exposure control function enabled, you can still change the frame rate manually or automatically to maintain a dynamic exposure control range. A lower frame rate allows for longer exposure times, but then the live image display may exhibit jitter. The objective of the automatic frame rate control is to set the frame rate to an optimum value. This way, in all situations, the automatic exposure control can use the required control range at the highest possible frame rate.

Auto white balance (AWB)

Depending on the lighting source, light can have different color temperatures so that the images may have a color cast. At low color temperatures (e.g. light from incandescent lamps), the white content is offset towards a red hue. At high color temperatures (e.g. light from fluorescent lamps), the white content is offset towards a blue hue.

The white balance control feature uses the RGB gain settings of the camera to correct the white level. This is achieved by adjusting the gain controls within the 0-100 % range until the red or blue channel matches the average brightness of the green channel. In order to manually influence the color rendering, you can adjust the setpoint values for the red and blue channels relative to the green channel by using an offset value (see also ThorCam > Histogram).

Automatically disabling the control function

You can disable the control functionality automatically once the target value has been reached (approximately) and after 3 regulations no improvement has been reached (API parameters `IS_SET_AUTO_WB_ONCE` and `IS_SET_AUTO_BRIGHTNESS_ONCE`). An event/a message notifies the system of this (see also `is_InitEvent()`). Alternatively, you can keep the control feature enabled so that it responds to deviations from the target value.

Control speed

You can set the auto function speeds in a 0–100 % range. This influences the control increments. High speed (100 %) causes a little attenuation of a fast-responding control and vice versa. The control functions for average brightness and for color rendering use separate speeds.

In trigger mode, every frame is evaluated for automatic control. The freerun mode skips a number of frames by default because in that mode, changes to the image parameters only become effective after one or more image captures (see also Applying new parameters). With the "Skip Frames" parameter (API parameter `IS_SET_AUTO_SKIPFRAMES`), you can select how many frames should be skipped in freerun mode (default: 4). This parameter strongly influences the control speed. Choosing small values can destabilize the automatic control.

Note

For higher frame rates select for the "Skip frames" parameter a bigger value. This reduces the number of automatic adjustments that must be done by the camera.

Hysteresis

The automatic control feature uses a hysteresis function for stabilization. Automatic control is stopped when the actual value lies in a range between (setpoint - hysteresis value) and (setpoint + hysteresis value). It is resumed when the actual value drops below (setpoint - hysteresis value) or exceeds (setpoint + hysteresis value). If the hysteresis value is increased, the control function will stop sooner. This can be useful in some situations.

 © 2015 Thorlabs Scientific Imaging

See also:

- `is_SetAutoParameter()`

2.6.4 Applying New Parameters

New capture parameters (such as exposure time or gain settings) can be transferred to the camera via software at any time. Depending on the operating mode, these settings will not always be immediately effective for next image, however.

- Freerun mode
  In freerun mode, the camera is internally busy with capturing the next image while new parameters are transmitted to the camera. Depending on the exact time of transmission, new parameters might only come into effect two or even three images later.
- Trigger mode
  In this mode, the camera reverts to idle state between two images. When you change the camera parameters, the new settings will be applied immediately to the next image.

2.7 Firmware and Camera Start

Every DCx camera has its own firmware that handles internal processes in the camera. The camera firmware varies from model to model.

USB DCx Cameras have a two-tier firmware that is uploaded to the camera each to you connect it to a PC:

1. Common firmware (uc480 boot)
   The general firmware identifies what camera model you have connected, and uploads the corresponding firmware.
2. Model-specific firmware (e.g.: uc480 DC1240x series)
   The model-specific firmware is named after the camera type and provides the functions of the relevant model.

Note

When you connect a USB DCxCamera with a Windows PC or a new USB port for the first time, it is detected as a new device. This is normal standard behavior of the operating system.

The USB DCx Cameras firmware is part of the driver. The automatic upload always loads the firmware that matches the driver installed in the camera.

2.8 Digital Inputs / Outputs

All DCx cameras (see Model comparison), except DCC1545M and DCC1645C, come with opto-isolated inputs/outputs that can be used for triggering the camera and for flash control. DC3240x cameras have in addition general purpose I/Os (GPIO). Use of the GPIOs for flash control is possible to a certain degree. External triggering via the GPIO is not supported.

See also:

- Basics: Trigger mode
- ThorCam: Settings > Input/output
- Specification: Electrical specifications

Programming:

- `is_IO()`

2.8.1 Using Digital Inputs/Outputs

Digital input (trigger)

Models with optocoupler input can use the digital input for externally triggering the image capture, or query the applied signal level.

In trigger mode, a digital signal is applied to the camera's input. You can determine whether the camera will respond to the rising or falling edge of the digital signal. After an internal delay, the sensor is exposed for the defined exposure time. The captured image is then transferred to the PC.

On models with general purpose I/Os (GPIO), you can query a voltage level at these inputs (TTL compatible).

Digital output (flash)

The digital outputs can be used in both freerun mode and trigger mode. You can synchronize the output level to the exposure time or set it statically.

Models with optocoupler output allow control of a DC voltage applied to the output. This allows controlling a flash, either directly or via a separate flash controller unit. Models with general purpose I/Os (GPIO) can output a voltage at these outputs (TTL compatible).

Note

Please read the notes on I/O wiring for your camera model in the Electrical specifications chapter.

Note

The settings specified for the digital output will be reset when the camera is disconnected from the PC or the PC is powered down.

2.8.2 Flash Timing (Trigger Mode)

When using the digital output for flash control, you can set the delay and the duration of the flash. The flash timing can be adjusted manually or automatically by the camera driver.

Note

Sensor latency and delay times The sensor latency is due to a number of technical factors, including sensor type, image geometry, pixels clock and, with CCD sensors, the exposure time. The latency is constant for a specific combination of parameters.

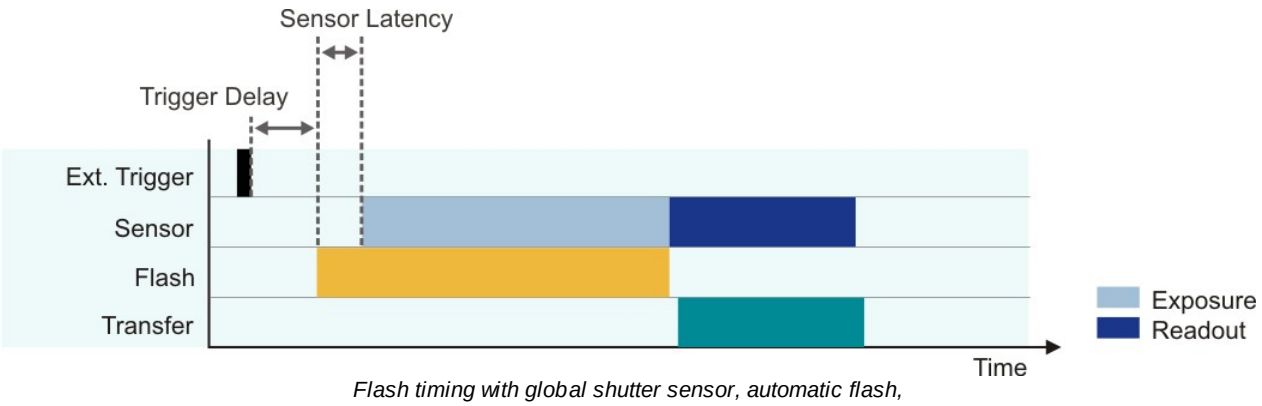
Trigger delay, flash delay and flash duration are optional and can be set by software.

The following illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera model and the current parameter settings.

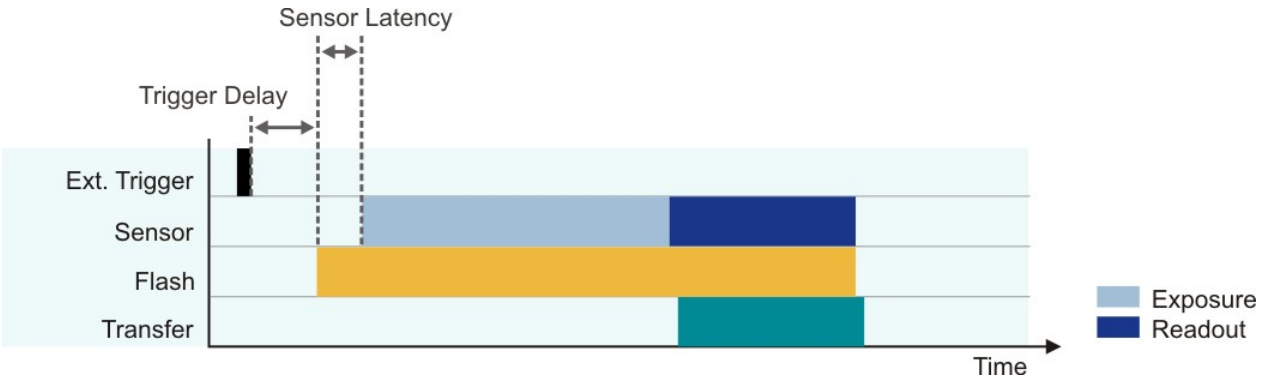
Automatic flash

If flash delay = 0 and flash duration = 0, the flash signal is automatically synchronized to the exposure time. The automatic flash feature has the advantage that the flash is synchronized automatically if the settings for image geometry or camera timing are changed. The disadvantage is that the flash signal is active slightly longer than the exposure time. The flash duration with automatic flash is longer for rolling shutter sensors than for global shutter sensors.

 © 2015 Thorlabs Scientific Imaging



Flash timing with global shutter sensor, automatic flash, optional trigger delay



Flash timing with rolling shutter sensor, automatic flash, optional trigger delay

Manual flash synchronization

If one of the flash delay or flash duration parameters is set to a value greater than 0, you can shift the flash signal to any point in the exposure time or change its duration. In this case, the flash delay will be calculated exactly from the start of the exposure time (after the sensor latency time). When manually synchronizing the flash signal to the exposure time, you can use the `is_IO()` function to query the data you need.

The advantage of manual flash synchronization is that the flash can be precisely controlled based on the start of exposure. This applies to both rolling and global shutter sensors. You can thus achieve a higher accuracy with the manual flash synchronization than with the automatic flash feature.

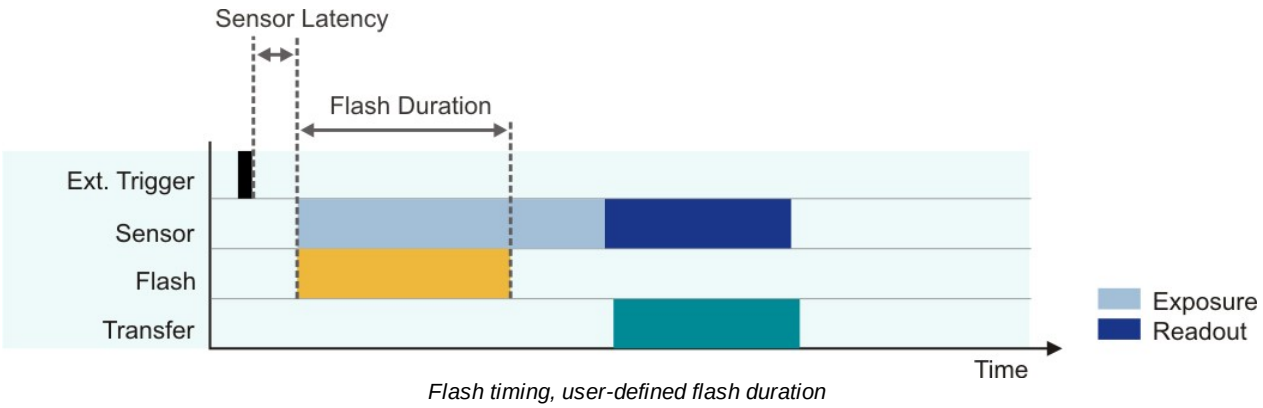
The disadvantage is that the flash signal has to be resynchronized whenever any settings for image geometry or camera timing change.

Note

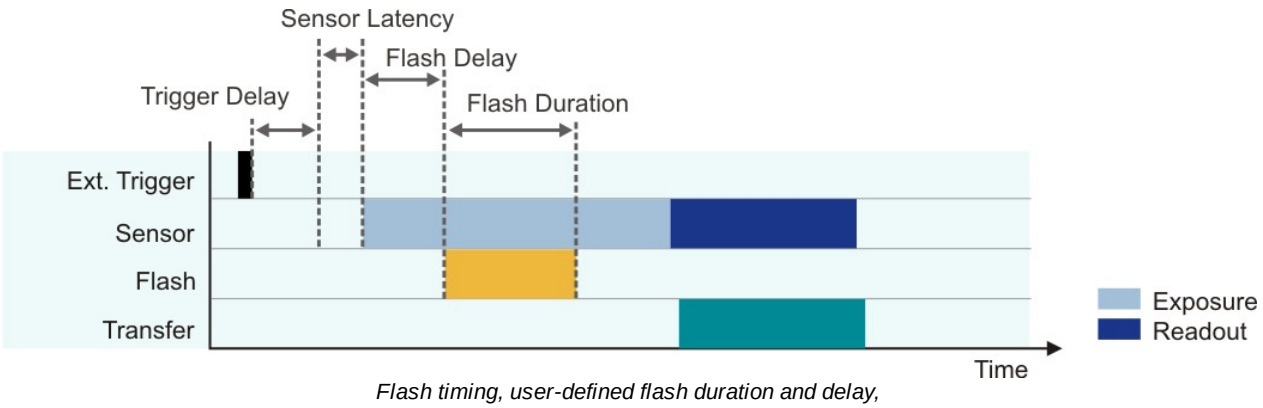
With rolling shutter sensors, you can avoid the rolling shutter effect by selecting suitable delay and duration settings (global flash function). Using `is_IO()`, you can query the appropriate values.

Note

The flash output is reset with the start of the next image capture. This also applies if you have set a longer flash duration.



Flash timing, user-defined flash duration

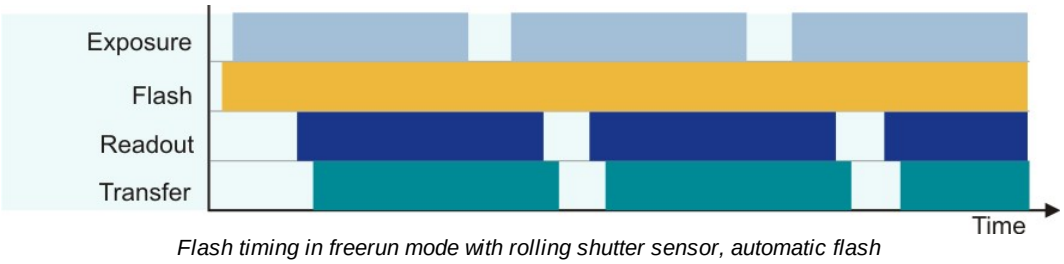


Flash timing, user-defined flash duration and delay, optional trigger delay

2.8.3 Flash Timing (Freerun Mode)

Automatic flash

In freerun mode, the automatic flash feature works in the same way as in trigger mode. As a result, the flash output is continuously or almost continuously active (see illustration below).



Flash timing in freerun mode with rolling shutter sensor, automatic flash

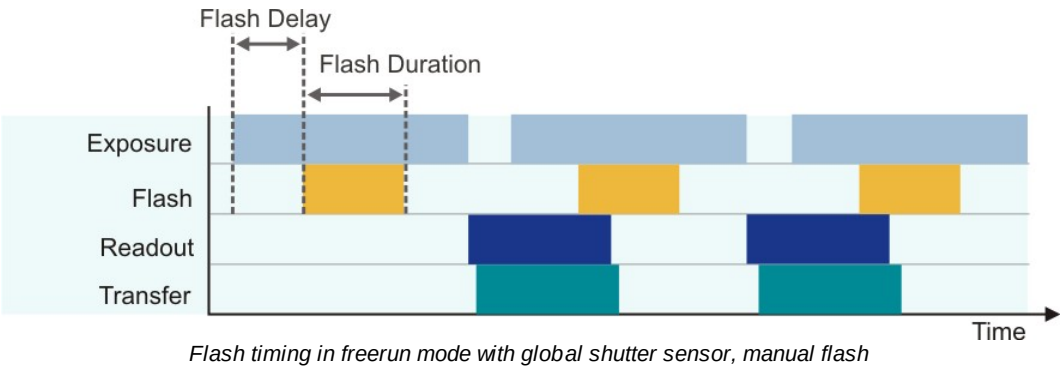
Manual flash synchronization

Note

It is recommended to synchronize the flash manually in freerun mode. This applies to both rolling and global shutter sensors.

In freerun mode, the manual flash synchronization works in the same way as in trigger mode.

© 2015 Thorlabs Scientific Imaging



Flash timing in freerun mode with global shutter sensor, manual flash

2.8.4 Serial Interface RS-232 (DC3240x only)

DC3240x cameras are equipped with a serial interface (RS-232). It provides functionality for communication with peripheral devices (e.g. lighting controller, lens controller or the serial port of a PC). Before you can send data through the camera's serial interface, one or more virtual COM ports have to be defined on the PC. Once defined, they can be used for data communication with appropriate software just like any physical COM port.

To set up and use the serial interface, the "Additional functions" dialog box is provided in the uc480 Camera Manager. For the serial interface specifications, please refer to the DC3240x Serial Interface Wiring (RS-232) chapters.

2.9 USB Interface

- History and development
- Structure and topology
- USB 2.0 cabling and connection
- USB 3.0 cabling
- Data transmission and bandwidth

2.9.1 History and Development

The **Universal Serial Bus** (USB) is an interface which enables you to easily connect various devices to a PC. As all data exchange is controlled by the PC, no additional interface controller is needed. Further advantages of USB are:

- The PC does not have to be shut down when connecting USB devices (hot plugging).
- USB devices can be supplied with power from the PC.
- High bandwidth for data transmission.

The USB standard was developed by a group of companies including Compaq, IBM, Intel, and Microsoft. Version 1.0 was presented in 1995. The slightly faster USB 1.1 standard followed in 1998.

At first, the USB interface was designed to connect peripheral devices such as printers, mice, or keyboards. With the introduction of USB 2.0 in 2000, the transfer rate increased to 480 Mbit/s, making USB 2.0 suitable for connecting devices with higher data volumes (such as mass storage devices, scanners, or cameras).

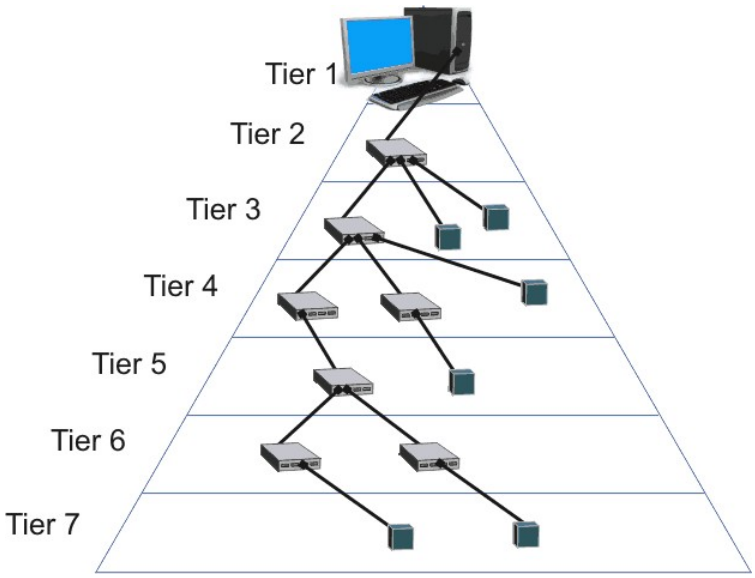
In 2008, with USB 3.0 a new version of the interface has been published, which is significantly faster than USB 2.0 (400 MByte/s).

2.9.2 Structure and Topology

USB uses a tree topology and is host-controlled. That means that a PC with host functionality is mandatory for using USB. Therefore, it is not possible to directly connect two USB devices (with the exception of USB on-the-go compliant devices). Neither is it possible to connect a camera to a PDA device.

Theoretically, 127 devices can be connected to a host controller. Using external hubs or repeaters, even more devices can be connected, and from a greater distance. Provided that a maximum of 5 hubs/repeaters may be daisy-chained, USB devices can be connected in up to seven levels.

 © 2015 Thorlabs Scientific Imaging



USB topology

Note

The maximum bandwidth of 480 Mbit/s per USB 2.0 host or 400 MByte/s per USB 3.0 host cannot be exceeded. Therefore, the maximum possible frame rate will be reduced if image data from multiple USB cameras is transferred simultaneously.

The available bandwidth might also be decreased when you use hubs or repeaters. You can reduce the bandwidth required for each camera by lowering the frame rate or the image size.

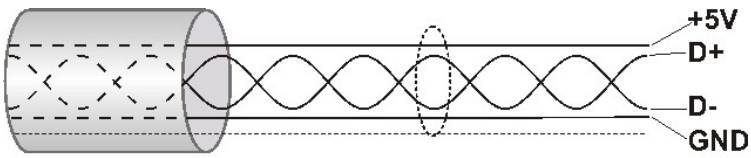
2.9.3 USB 2.0 Cabling and Connectors

In order to comply with the specifications, the maximum length of USB 2.0 cables is limited to 5 m. Longer cables may be connected if you use high-quality material. The USB bus provides power supply with 5 V and 500 mA max. Many USB devices use the bus power and do not need external power supply (bus-powered devices).

Cable design

The following illustration shows the basic design of a shielded USB cable:

- D+/D-: data transfer
- +5 V/GND: power supply



Basic design of a USB cable

Connector types

On the PC side, USB 2.0 cables are equipped with a standard A type plug (four pins) and on the device side either with a standard B plug (four pins) or a mini-B plug (five pins).

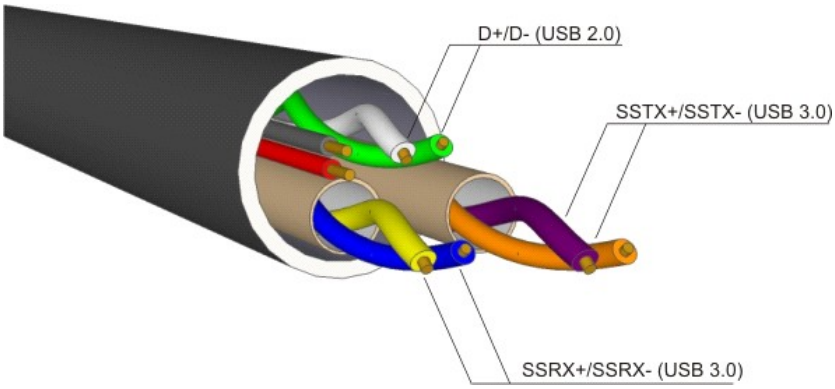
2.9.4 USB 3.0 Cabling and Connectors

In order to comply with the specifications, the maximum length of USB 3.0 cables is limited to 3-8 m. With the use of repeaters cable lengths up to 20 m are possible. With signal conversion into optical signals cable lengths up to 100 m are possible. The USB bus provides power supply with 5 V and 900 mA max.

Cable design

The following illustration shows the basic design of a shielded USB cable:

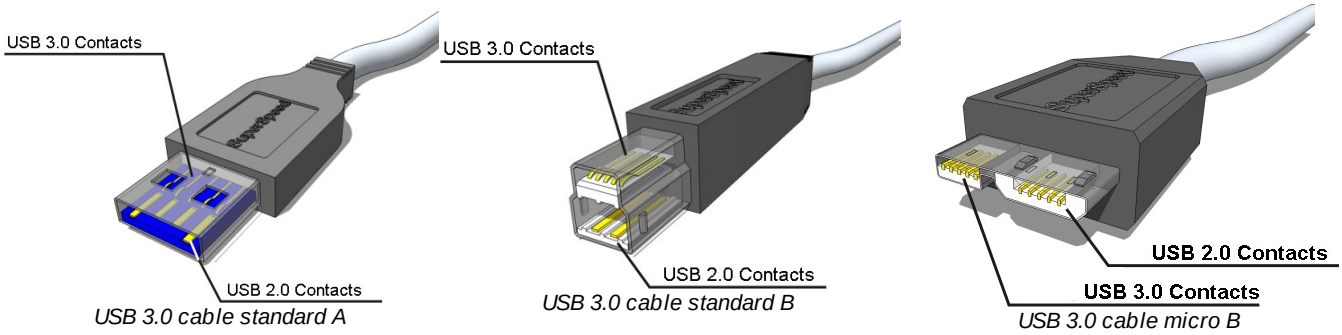
- SSTX+/-: SuperSpeed Transmit (data transfer from host to the device)
- SSRX+/-: SuperSpeed Receive (data ransfer from device to the host)
- D+/D-: data transfer (USB 2.0)
- +5 V/GND: power supply



Basic design of a USB 3.0 cable

Connector types

On the PC side, USB 3.0 cables are equipped with a standard A type plug (8 pins) and on the device side either with a standard B plug or a micro-B plug.



USB 3.0 cable standard A *USB 3.0 cable standard B* *USB 3.0 cable micro B*

While a USB 3.0 standard A plug and a USB 2.0 port can be used together (with the restriction that only USB 2.0 speed is possible), the standard B and micro B plug are no longer compatible with USB 2.0 ports.

2.9.5 Data Transmission and Bandwidth

USB 2.0

The USB 2.0 standard specifies an overall bandwidth of 480 Mbit/s shared between different transmission modes. DCx Cameras use the USB 2.0 bulk mode for transmitting images. This mode uses error correction to ensure correct delivery of the image data, but does not guarantee a

 © 2015 Thorlabs Scientific Imaging

fixed bandwidth. To ensure error-free communication with all connected devices at all times, the maximum bandwidth for payload data is limited to 416 Mbit/s.

Theoretically, up to 50 MByte/s of data can be transmitted in this mode, but in practice, this value is hardly ever reached. A high-performance desktop PC can transmit about 40 MByte/s, most notebooks or embedded PC systems even less than that.

The overall bandwidth can be increased by the use of USB 2.0 expansion cards. These cards are available for the PCI and PCIe buses and have their own host controller chip.

Note on hardware selection

To achieve optimum USB bandwidth, it is important to use a powerful mainboard chipset. The mainboard chipsets from e.g. Intel® or NVIDIA® provide very good results.

Note on image content and bandwidth usage

For USB cameras, you can use a white test image to check the camera's maximum load on the USB bus. Due to the transmission process, completely white camera images require a somewhat more bandwidth on the USB bus than completely black images.

USB 3.0

Compared to USB 2.0, USB 3.0 offers a tenfold increased bandwidth of 5 Gbit/s, i.e. images can be transmitted with a bandwidth of 400 Mbytes/s.

3 Operation

This chapter explains how to connect the DCx camera and how to use the applications contained in the uc480 software package.

- Quick start
- Installation and connection
- Installed uc480 programs
  - o ThorCam: A comprehensive viewer for exploring the camera functionality (See separate ThorCam User's Manual)
  - o uc480 Camera Manager: The central tool for managing all connected DCx Cameras.
  - o uc480 Hotpixel Editor: A tool to edit the sensor hot pixel list stored in the camera.

3.1 uc480 Quick Start

This chapter show how to quickly get started with your DCx camera. You will learn how easy it is to connect the camera and explore important functions. For further steps of integrating the DCx camera into your own applications please also see the First steps to uc480 programming chapter.

Connect the camera

Install the latest version of the DCx software. Then connect the DCx camera with the PC. USB cameras are automatically detected as new hardware under Windows. Check the status LEDs on your camera to see if the camera has been correctly identified.

See also:

- Installation and connection
- Troubleshooting
- Connection - Status LED

Configure the camera

USB DCx Cameras are ready for use right out of the box. You can assign a unique ID to your camera with the uc480 Camera Manager.

See also:

- uc480 Camera Manager
- Assigning a camera ID in the camera manager
- Firmware and camera start

Capture images

The uc480 software package includes many sample programs that you can use to try out the extensive functionality of your DCx camera. We recommend starting off with the ThorCam application. To run the application, simply double-click the corresponding icon on your Windows desktop.

See also:

- ThorCam Software User Manual (Separate Document)
- Camera basics: Operating modes

Customize the key camera properties

Select the Settings icon on the menu bar to open the dialog box for modifying the camera properties.

© 2015 Thorlabs Scientific Imaging

The "Camera" tab provides all the parameters for adjusting the camera's speed. You can increase the pixel clock to run the camera at a higher frame rate. Reduce the pixel clock if transmission errors occur too often.

On the "Image" tab, you find various sensor gain controllers. Use the "Master gain" controller to increase image brightness if no longer exposure time setting is possible. Switch to the "AES/AGC" tab to enable the Auto Exposure Shutter (AES) and Auto Gain Control (AGC) features.

Tip: Select a low sensor gain to minimize visible noise.

If you are using a color camera, you should activate sensor color correction on the "Color" tab in order to achieve rich vibrant colors for on-screen display. To adapt a color camera to the ambient light conditions, it is essential to carry out Auto White Balance (AWB). Aim the camera at a surface of a uniform gray color, then enable the "Image white balance: Enable" and "Run once" check boxes on the "AWB" tab.

See also:

- ThorCam Software User Manual (Separate Document)
- Camera basics: Camera parameters

Activate trigger and flash modes

DCx Cameras provide the possibility to trigger the image capture and to have the flash controlled by the camera. To switch the camera to trigger mode, go to the camera properties as described above, select the "Trigger" tab and enable the desired mode. To trigger on "falling edges" or "rising edges", a digital signal has to be applied to the camera. When you are finished with the trigger settings, select "Start Continuous Trigger" or "Triggered Snapshot" icons on the menu bar to start the triggered image capture.

If you have connected the digital output on your DCx camera to a flash controller, you can configure the flash function on "Input/Output" tab. Enable "Flash high active" and "Global exposure window". This way, the DCx camera automatically activates the flash during the exposure time.

See also:

- ThorCam Software User Manual (Separate Document)
- Camera basics: Digital input/output
- Specifications: Electrical specifications

Save the camera settings and images

With ThorCam, saving single frames or videos is very easy to do. Just choose the relevant option on the "File" menu. If you have recorded AVI videos, you can play them through Windows Media Player, ImageJ, or other AVI-compliant viewer with Motion-JPEG decoding capability.

See also:

- ThorCam Software User Manual (Separate Document)

3.2 Installation and Connection

- System requirements
- Installing uc480 software under Linux
- Connecting a USB DCx camera

3.2.1 System Requirements

For operating the DCx cameras, the following system requirements must be met:

	Recommended
CPU speed	>2.0 GHz Intel Core i5 or Core i7
Memory (RAM)	8 GByte
For USB DCx cameras: USB host controller	USB 3.0 Super Speed Intel® motherboard chipset
Graphics card	Dedicated AGP/PCIe graphics card Latest version of Microsoft DirectX Runtime 9.0c
Operating system	Windows 8.1 32 or 64 bit Windows 7 32 or 64 bit

Drivers for network cards

To ensure optimum performance of the network connection, you need to install the latest drivers for your network card. We recommend using the drivers of the following versions:

- Intel® chipsets: version 8.8 or higher
- Realtek chipsets: version 5.7 or higher

USB interface

- Onboard USB 2.0 ports usually provide significantly better performance than PCI and PCMCIA USB adapters.
- Current generation CPUs with energy saving technologies can cause bandwidth problems on the USB bus. See section on PCs With Energy Saving CPU Technology.

Large multi-camera systems

Connecting a large number of cameras to a single PC may require a large working memory (RAM). This is especially the case when many cameras with high sensor resolution are used.

If you want to set up such a system we recommend to use PCs with 64 bit operating systems and more than 4 GB of RAM.

Note on color cameras with high frame rates

For uc480 color cameras, the color conversion is done by software in the PC. When you use a color camera with a high frame rate, the conversion might lead to a high CPU load. Depending on the PC hardware used you might not be able to reach the camera's maximum frame rate.

© 2015 Thorlabs Scientific Imaging

Direct3D graphics functions

The uc480 driver can use Direct3D to display the camera image with overlay information (Microsoft DirectX Runtime had to be installed). On Windows systems, you can use the supplied "DXDiag" diagnostic tool to check whether your graphics card supports Direct3D functions. To start the diagnostic tool, click "Run…" on the Windows start menu (shortcut: Windows+R) and enter "DXDiag" in the input box.

On the "Display" page of the diagnostic tool, click the button for testing the Direct3D functions.

OpenGL graphics functions

For OpenGL version 1.4 or higher must be installed. The OpenGL graphics functions do not work with QT under Linux.

3.2.2 DCx Driver Compatibility

Attention

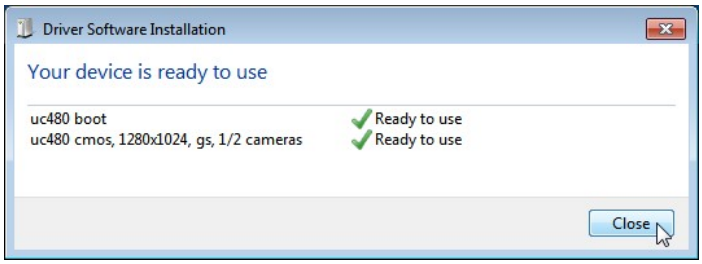
Support of older DCC1545M cameras by driver versions 3.10 and higher

From driver version 3.10 onwards, only USB board revision 2.0 or higher are supported. To operate a camera with an earlier USB board revision, you will need the *uc480* driver version 2.40. Please contact Thorlabs

The LED on the back of the camera housing also indicates the USB board version (see DCx Status LED). In addition, the *uc480 Camera Manager* version 3.10 or higher provides information about the compatibility (see Camera Manager). An incompatible camera will be displayed as *free* and *not available*.

3.2.3 Connecting a DCx Camera

Please install the software first as described in the Quick Start Guide. Connect the DCx camera to the PC, using the USB cable. The camera will be recognized automatically and the necessary driver software is being installed:

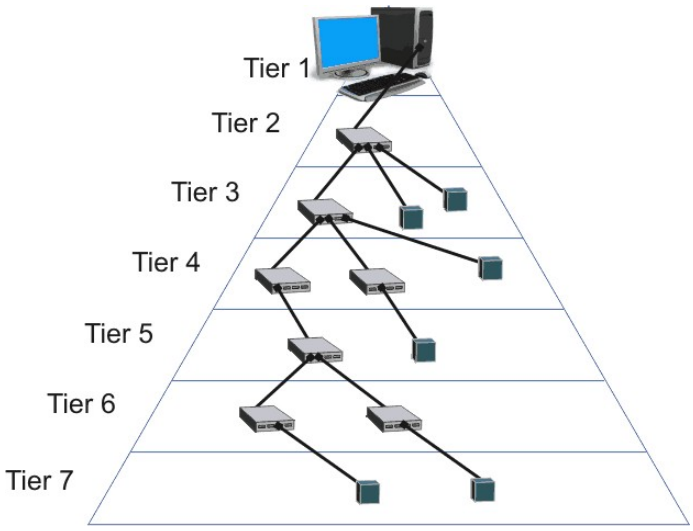


When the camera has been correctly installed, the LED on the back of the camera lights up green.

Note

The first time you connect a USB DCx camera to a USB port under Windows, two driver files will be registered. The first file (uc480 boot) contains the generic driver, the second file the model-specific driver.

The model will be immediately recognized whenever you connect the camera to this port again. If you use a different port, the registration will be repeated. Under Windows the camera will show up in the uc480 Camera Manager's camera list.



The DCx Cameras can be connected to a USB port either directly or via hubs and repeaters. A wide range of different hubs and repeaters are available commercially. The USB 2.0 hubs being used must be "full powered" hubs that are able to provide 500 mA per USB port. "Low Powered" hubs, in comparison, only supply 100 mA per port, which is not sufficient for DCx Cameras.

Note

To use maximum bandwidth, we recommend connecting the cameras directly to the USB ports on the mainboard. Many USB ports on PCI/PCIe cards and the USB ports on the front of the PC often supply lower bandwidth.

Attention

USB cables with non-standard connectors must be connected to the camera first and then to the PC. Otherwise the camera might not be recognized correctly.

© 2015 Thorlabs Scientific Imaging

3.3 Application Notes by Camera Model

Cameras with CMOS sensors

- DCC1240x / DCC3240x Application Notes
- DCC1545M Application Note
- DCC1645C Application Notes

Cameras with CCD sensors

- DC223x Application Notes
- DC224x Application Notes

3.3.1 DCC1240x / DCC3240x Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCC1240x / DCC3240x.

Shutter modes

The following table displays the four shutter modes and their advantages and disadvantages in different situations:

	Global shutter (default)	Global shutter (alternative timing)	Rolling shutter	Rolling shutter (global start)
Black level constancy	-	+	++	++
Capturing of moving objects	++	+	-	+
Hotpixels	-	-	+	+
Image quality with high gain	-	+	++	+

- The rolling shutter mode offers a better signal/noise ratio and a more consistent black level compared to the global shutter mode.
- If the sensor is used in global shutter mode at a low pixel clock frequency and a high gain, the bottom pixel rows might become brighter for technical reasons. Color distortion will occur for the color sensor with active white balance. In this case, use a higher pixel clock frequency, less gain or the rolling shutter mode.
- The "Global shutter (alternative timing)" mode offers a more consistent black level compared to the global shutter mode. This mode should not be used with a frame rate below 2 fps. This mode is also not suitable for bright, moving image contents in combination with long exposure times.
- The rolling shutter mode with global start is suitable for capturing moving objects with flash.
- When using flash in rolling shutter mode make sure to set the flash duration accordingly ((1/maxFramerate) + exposure) or that the global time window is available by a long exposure time (2 * (1/MaxFramerate) + FlashDuration). For flashing into this time window use the flash delay (1/MaxFramerate).
- The hardware sensor gamma curve is piecewise linear with three sections. This allows evaluating four times more details per pixel for lower gray level values and in 8 bit per pixel mode in dark image areas.
- In global shutter mode the shutter efficiency of 1:3000 have a negative impact with bright conditions and the usage of exposure times under 100 µs. In this case, set the pixel clock to the

maximum possible value and close the aperture a little bit. Also enabling the Log mode with low values achieve huge improvements.

Black level

- The black level can also be set to negative values. Therefore, the factory setting of the offset control is nearly in the middle of the range.
- Use of the gain functions can lead to slight fluctuations of the black level. In global shutter mode the black level can also vary slightly.
- In global shutter mode the black level can also vary slightly between two image captures.
- When enabling the rolling or global shutter mode the black level is set to a fixed factory-provided value. Therefore, the black level can individually adjusted after switching the shutter mode.
- Depending on the internal black level and shutter mode the offset control shows no additional cumulative effect at the top.
- The factory setting of the offset control are so selected that the black level is always slightly increased to avoid losing image information by cutting underneath the origin. For linearity measurements the black level must be adjusted to the origin with the offset control before the measurement is done.

Color sensor

- The color sensor's black level cannot be adjusted manually, as the RGB gains are downstreamed and an adjustment would cause color errors.
- Automatic black level correction is always enabled.
- The RGB gains work analog.
- The fast line scan mode is disabled.

NIR sensor

- It is recommended to use a IR-coated and IR-corrected high-quality lens, especially for non-monochromatic light.
- In the high IR wavelength range picture blur can occur with strong contrasts. This reduced MTF (modulation transfer function) is a characteristic of the sensor pixels.
- The master gain of the NIR sensor is adapted in comparison to the monochrome sensor. If both sensors are compared directly, the gain of the NIR sensor must be set to the double factor of the monochrome sensor. This can be done via the master gain or the gain boost.
- In the Log mode the guaranteed dynamic range of the NIR sensor is reached with a gain value of at least 1. For the monochrome sensor you need at least a value of 3. With very short exposure times, lower values can result in even higher dynamics.

Gain, pixel clock

- Master gain uses a combination of coarsely scaled analog gain factors and finer digital scaling. To achieve optimum homogeneity of the gray level, use only the gain factors 0, 33, 66, and 100.
- The gain boost has the factor 2. When using the master gain a maximum factor of 8 is possible caused by the sensor.
- For global shutter mode the pixel clock should be set to the maximum possible value to increase the image quality.
- In the 10 bit mode the usage of the digital gain intermediate level produces missing pixel values as the sensor works internally with maximally 10 bit.

 © 2015 Thorlabs Scientific Imaging

Hot pixel

- In the rolling shutter mode, there are less hot pixels, as the pixel charges are not buffered in the sensor.
- The sensor corrects hot pixels dynamically. Neighboured hot pixels in diagonal direction cannot be corrected effectively. These positions are covered by the factory-made hot pixel correction and are eliminated by the software hot pixel correction. Therefore, the hardware hot pixel correction is a prerequisite and should not be deactivated.
- On the color sensor the hot pixel correction works with the appropriate color neighbours.
- In the global shutter mode, increased hot pixels can become visible in the lower image area with log exposure time, a high gain and disabled hot pixel correction.
- Both hot pixel corrections should be disabled if extremely fine structures are captured with a high-quality lenses.
- For the measurements of noise characteristics both hot pixel corrections should be disabled.
- The activation of the factory-made hot pixel correction reduces the frame rate slightly. Here, the desired frame rate has to set **after** the activation or deactivation of the factory-made hot pixel correction.
- Increased hot pixels can occur In the entire image border area.

Binning

- 2x binning makes the image brighter by a factor of about two. It also reduces image noise.
- Binning does not result in a higher frame rate. Using binning allows higher pixel clock frequencies for USB DCx cameras with the USB 2.0 and GigE interface. To achieve the maximum frame rate, activate first the binning and then change the maximum pixel clock frequency.
- When using binning the frame is slightly shifted horizontally.
- Binning can only be enabled for both horizontal and vertical pixels. For this reason, the parameters of the `is_SetBinning()` function have to be passed together (`IS_BINNING_2X_VERTICAL | IS_BINNING_2X_HORIZONTAL`) to enable binning.

Scaling, AOI

- The digital scaling functions result in a higher possible frame rate. The maximum frame rate is increased approximately proportionally to the scaling factor. When using the scaling functions with USB 2.0 and GigE cameras, you can set higher pixel clock frequencies. To achieve the maximum frame rate, select first the scaling factor and then the maximum pixel clock frequency.
- For improved image quality without undersampling artifacts, the digital scaling feature permanently uses an anti-aliasing filter.
- On color sensors the scaler works in consideration the color information.
- Reducing the horizontal resolution does not result in a higher frame rate.
- The available step widths for the position and size of image AOIs depend on the sensor. The values defining the position and size of an AOI have to be integer multiples of the allowed step widths. For detailed information on the AOI grid see the DCC1240x / DCC3240x chapter.

Multi AOI

- When the Multi AOI function is enabled, no changes can be made to the image size settings (e.g. binning, subsampling, scaling).
- If sensor hotpixel correction and Multi AOI are enabled, the sensor displays a four pixel wide black line between the AOIs.

Line scan mode

- Fast line scan mode: The exposure time is fixed to the readout time of one sensor line. Exposure time cannot be changed in this line scan mode.
- Fast line scan mode: The time stamp is generated for the complete image.
- Fast line scan mode: There is a time gap between two frames with the line scan information. This corresponds to the duration of 15 lines at a frame rate set to maximum.
- In the fast line scan mode color images are not possible as Bayer color sensors needs at least two neighboring lines for color calculating. Therefore, only monochrome models support the line scan mode.

Log mode

- The Log mode shows visible effect only for short exposure times (< 5 ms)
- In global shutter mode the Log mode can help to increase the shutter efficiency for extremely short exposure time (< 100 µs).
- To find the right operating point of the Log mode use the following procedure:
  1. Set the Log mode gain to the minimum value.
  2. Find the operating point via the Log mode value. The higher the value, the more bright image areas are damped and more details become visible.
  3. Set the image as bright as possible via the Log mode gain. A typical value is 2 or 3 for monochrome sensors and 0 or 1 for NIR sensors.
- The master gain is disabled in Log mode.
- See ThorCam: Settings > Shutter > Log Mode

Anti blooming

- Activation: With exposure times over 10 ms and no use of gain or gain boost it can occur that bright image areas do not reach saturation and so no white level which is caused by the sensor. A visible, firm pattern is formed in bright image areas. For color sensors with enabled white balancing bright image areas gets purple. In this case, disable the anti blooming mode or increase the master gain from 1x to 1.5x.
- The anti blooming mode should not be enabled for short exposure times (< 5 ms). Depending on the shutter mode a "Black Sun" effect or overexposure occurs.

Micro lenses

- The sensor has non-removable micro lenses on each pixel. These micro lenses focus the incoming light for the subjacent smaller photodiode. This lens has a directive efficiency.
- To the corners of the active image area the micro lenses are slightly shifted to the photodiode. So the unavoidable non-vertical light incidence of C mount lenses is compensated. The shift is constant from the center to the corner and has a maximum of 12 degrees. When using a telecentric lens or parallel light incidence the shift must be considered as little shading effects may occur.

3.3.2 DCC1545M Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCC1545M.

Sensor

- Sensor speed does not increase for AOI width <240 pixels.
- Extreme overexposure shifts the black level. Please deactivate the Auto offset function in this

 © 2015 Thorlabs Scientific Imaging

case.

- At very long exposure times and minimum gain, the white level may not be reached. The gain should be increased by one step in this case.
- Monochrome version only: The sensor internally works like the color version. This might lead to artifacts when subsampling is used.
- The brightness of the first and last line might deviate due to the sensor.
- Gain values between 59 and 99 may lead to image inhomogeneity.
- When using very narrow AOIs, the sensor may not be able to calculate the correct black level. Use manual black level offset when problems with the black level occur.

Calibration

- Cameras with a date of manufacture after Dec. 9, 2008: The offset control has been calibrated internally. The calibration corrects offset errors when gain is used. In calibrated cameras, automatic black level correction is disabled by default. The calibration can only be used with uc480 driver version 3.31 or higher.
- Cameras with a date of manufacture before Dec. 9, 2008: If manual offset control is used, fixed pattern noise and horizontal lines may become visible. High gain values may shift the black level and therefore should be avoided.
  Offset increases the black level every 7th step. The steps in-between change the appearance of fixed pattern noise.

3.3.3 DCC1645C Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCC1645C.

Sensor

- At very long exposure times and minimum gain, the white level may not be reached. The gain should be increased by one step in this case.
- The RGB gain controls have no effect for values >90.

3.3.4 DCU223x Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCU223x.

Sensor

- Long exposure times will increase the number of hotpixels.
- High temperatures will increase the black level of individual pixels.

3.3.5 DCU224x Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCU224x.

Sensor

- Long exposure times will increase the number of hotpixels.
- High temperatures will increase the black level of individual pixels.
- When vertical 4x binning is activated, the minimum image width increases to 640 pixels.

3.4 Installed uc480 Programs

- uc480 Camera Manager: The central tool for managing all connected DCx cameras.
- ThorCam: A comprehensive viewer for exploring the camera functionality.
- uc480 Hotpixel Editor: A tool to edit the sensor hot pixel list stored in the camera.

3.4.1 uc480 Camera Manager

The DCx Camera Manager is the central tool for managing all DCx cameras. It displays information on the connected USB DCx Cameras and provides options for configuring them.

On Windows systems the DCx Camera Manager can be accessed as follows:

- Start > Programs > Thorlabs > Scientific Imaging > DCx Camera Manager

Click in the figure to get help on the functions.

- Camera list
  The camera list displays information on the connected DCx Cameras.

Attention

Under Windows XP (64 bit) and Windows Vista DC3240x Cameras are displayed in the DCx Camera Manager, but they cannot be opened, because they are supported by Windows 7 and Windows XP (32 bit) only

- Control center

© 2015 Thorlabs Scientific Imaging

In the control center, you can access the configuration and display detailed information on the connected DCx Cameras.

- In the English drop down box, you can choose the language for the DCx Camera Manager. This setting is saved and remains effective even after you close and reopen the program.
  For proper display of Asian languages on Windows systems, the support package for East Asian languages has to be installed on your system (in "Control Panel > Regional and Language Options").
- Click Close to close the application; any settings you have made are saved.
- The status box at the bottom indicates the current status of the selected camera. If it is available, the status message is shown in black. Otherwise, the status message is shown in red.
  If an error has occurred in a camera, a black exclamation mark on a yellow background is shown next the camera. The status box then indicates the cause of the error and suggests remedies.

3.4.1.1 Camera List

When a camera is activated (switched on or connected to the PC), it appears in the camera list of the uc480 Camera Manager after a few seconds.

Free	Avail.	Type	Cam.ID	Dev.ID	Model	SerNo.

The data shown in the camera list can be sorted in ascending or descending order by left-clicking on the respective column header.

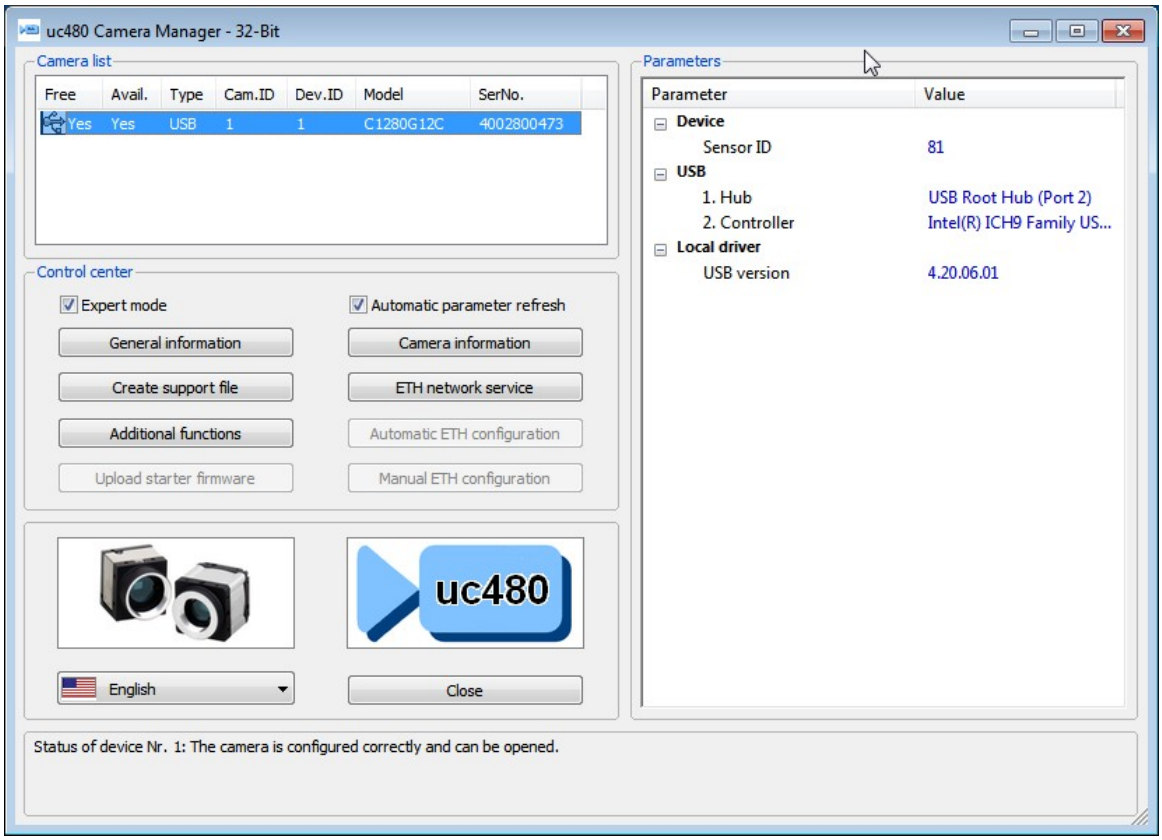
- Free/Avail.
  Free: indicates whether a camera is currently in use.
  Avail. (Available): indicates whether a camera can be opened by this computer with the current setup (computer and camera).
  Cameras shown with a red x are currently in use (Free = No) and are not available (Avail. = No).
  Cameras shown with an exclamation mark are not in use, but are currently unavailable for various reasons, such as:
  - o The camera is not compatible with the driver. Please update the uc480 driver.
  - o The driver has not properly detected (initialized) the camera. Please disconnect the camera from the PC and then reconnect it.
  - o The camera is currently being removed from the Manager.
  - o The camera reports that it is "Not operational".
- Type
  This column indicates the USB camera type.
- Cam.ID
  The camera ID assigned by the user.
- Dev.ID
  Unique device identifier sequentially assigned by the system. DCx cameras are assigned device IDs from 1 upwards. After deactivating a DCx camera (switching it off or disconnecting it from the network), the device ID is no longer valid and can be assigned again by the system.
- Model
  Model name of the camera
- SerNo.
  Serial number of the camera.

3.4.1.2 Control Center

- Expert mode

  When you select the ☑ Expert mode check box, the uc480 Camera Manager additionally displays the Parameters box on the right. There you will find detailed information on the DCx camera selected in the camera list.

Click in the figure to get help on the functions.



- Automatic parameter refresh

  If you select the ☑ Automatic parameter refresh check box, the data shown in the tree structure is updated periodically. If the option is disabled, the data in the tree structure is only updated when a different camera is selected.

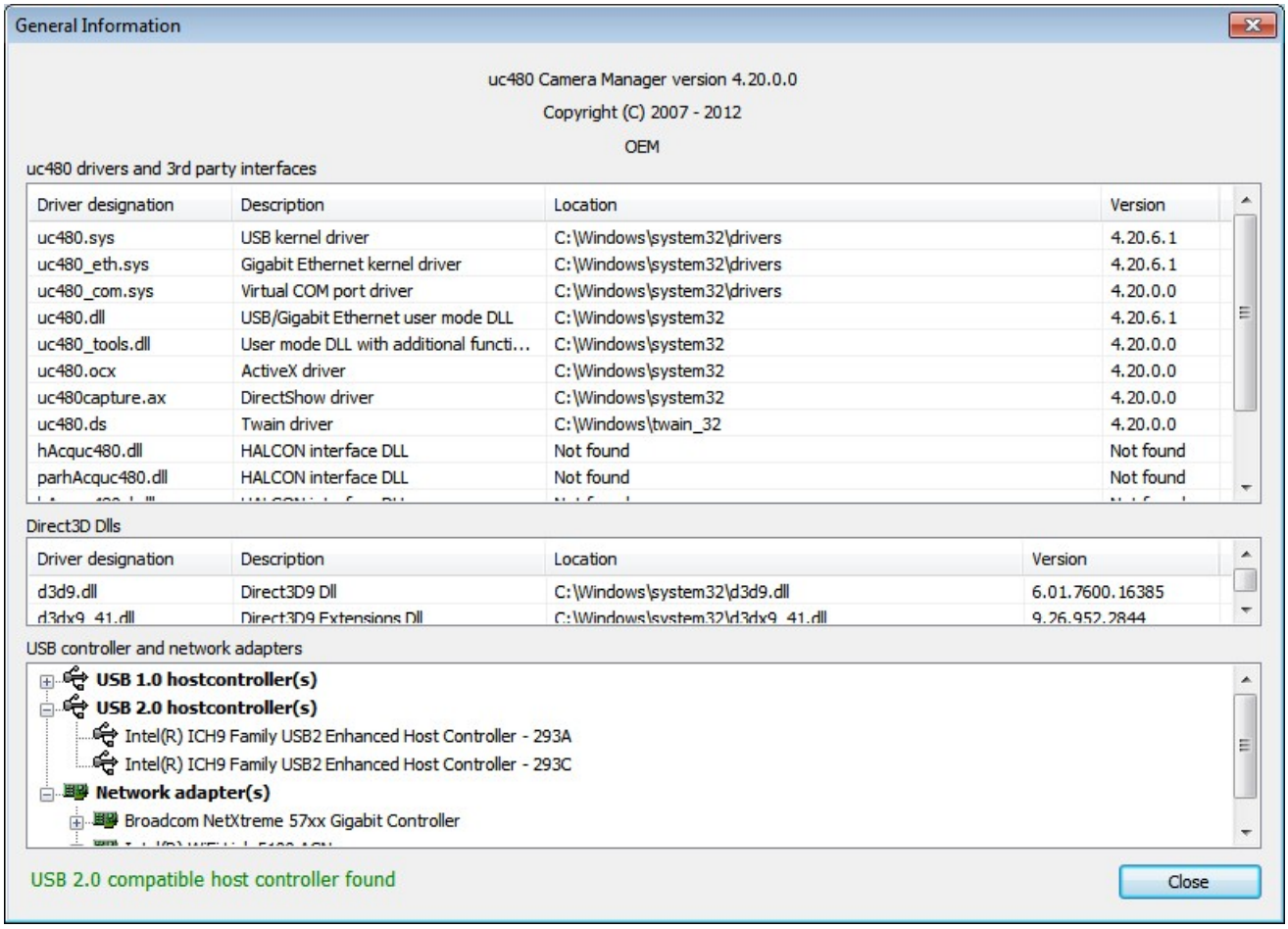
All other "Control Center" buttons are described in detail in the following sections:

- General information
- Camera information
- Creating a support file
- Additional functions (COM port)

© 2015 Thorlabs Scientific Imaging

3.4.1.3 General Information

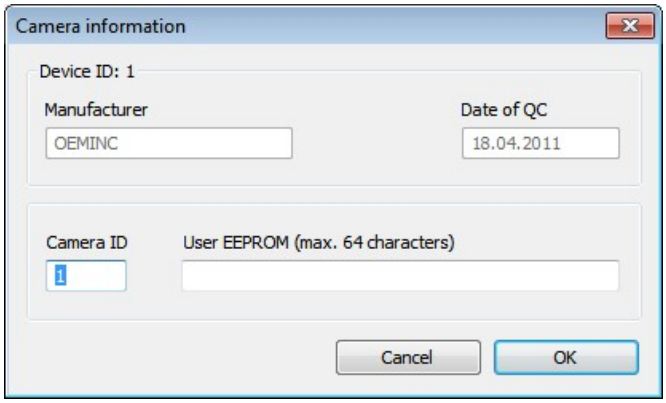
This dialog box provides information on the installed uc480 drivers and the available USB controllers and network adapters.



- uc480 drivers
  This list shows the location and version of the uc480 driver files installed on your system.
- 3rd party drivers
  This list shows the location and version of the uc480 interface files that have been installed on your system for third-party software.
- USB controller and network adapters
  All USB controllers and network adapters that are available in your system are shown in a tree structure.

3.4.1.4 Camera Information

In the "Camera information" dialog box, you can assign a unique ID to the selected camera and write to the user area of the EEPROM. The data you enter is retained in the camera memory even when the camera is disconnected from the PC or power supply.



- Camera ID
  The camera ID identifies a camera in multi-camera operation. The ID can range from 1 to 254. The default value for the camera ID is 1. The same ID can be assigned to multiple cameras. You do not have to assign sequential ID numbers to all connected cameras.
- User EEPROM (max. 64 characters)
  Every uc480 has a 64-byte user area in its EEPROM (Electrically Erasable and Programmable Read Only Memory) to which you can write text of your choice.

The "Camera information" dialog box displays two additional boxes that are for your information only and cannot be edited:

- Manufacturer (e.g.OEMINC)
- Date of QC (date of final camera quality test)

Notes

1. Setting a camera ID and writing to the EEPROM is possible only, if the camera is marked "Free" and "Available" in the Camera Manager (see also Camera list).
2. If software accesses cameras through the uc480 DirectShow interface, the camera IDs must be in a range from 1 to 24.
3. If software accesses cameras through the uc480 Cognex VisionPro interface, the camera IDs must be assigned consecutively beginning with 1.

3.4.1.5 Creating a Support File

A uc480 support file is a binary file with the extension `.bin`. The file contains camera and driver details that are required for diagnostics by our Technical Support team. No personal computer data or user data is stored in this file.

The [Create support file] button opens the "Save as" dialog box, where you can save the displayed camera information and additional driver information to a file.

3.4.1.6 Additional Functions

CPU idle states

Windows only: Processor operating states (idle states/C-states)
Modern processors have various operating states, so-called C-states, that are characterized by

 © 2015 Thorlabs Scientific Imaging

different power requirements. When the operating system selects an operating state with low power consumption (unequal C0), the USB transmission efficiency may be affected (see also `is_Configuration()` and section troubleshooting).

Camera parameters when camera is opened

Here, you can set whether to apply the parameters stored on the camera automatically when opening the camera. You must first store the camera parameters on the camera using the `is_ParameterSet()` function.

This setting applies to all connected cameras. If no parameters are stored on the camera, the standard parameters of this camera model are applied (see also `is_Configuration()`).

Boot boost

This mode is not related to DCx Cameras.

Bulk Transfer Size

Via "Bulk Transfer Size" the behavior of the USB sub-system can be set.

Warning

Contact our technical support before changing the value under "Bulk Transfer Size".

COM Ports

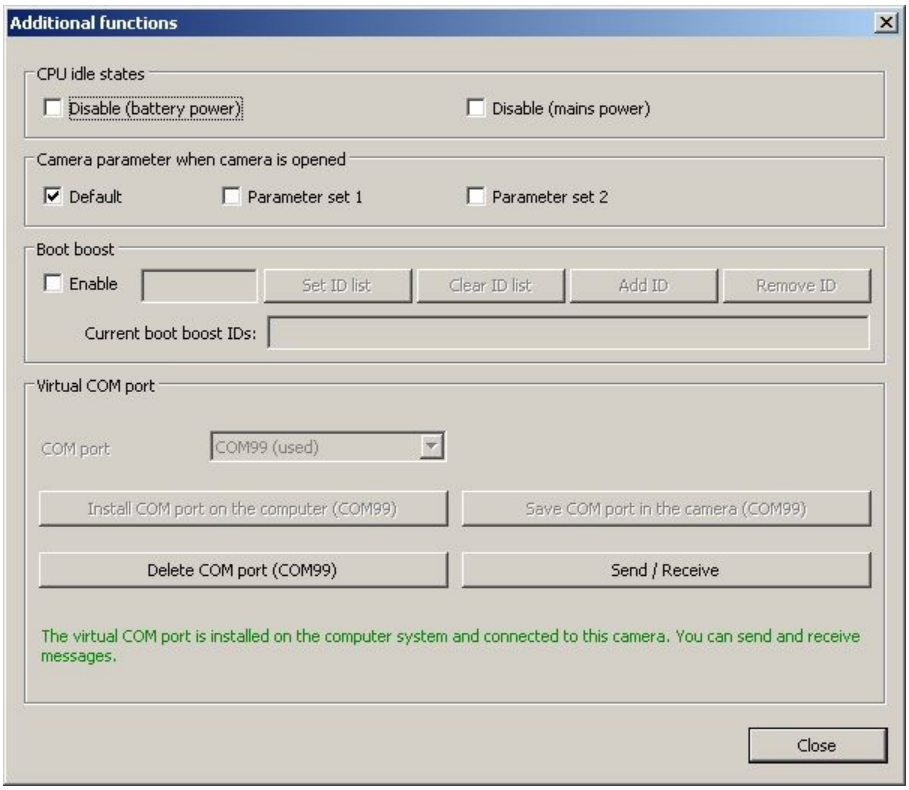
The "Additional functions" dialog box allows installing virtual COM ports for communication through the serial interface of a DCC3240x camera. The following sections show you how to set up and use the serial interface.

Note

This feature is only available for DCC3240x cameras.

You need administrator privileges to install a virtual COM port.

The selected in the uc480 Camera Manager DCC3240x camera has to be marked "Free" and "Available".



Setting up the serial interface on the DCC3240x camera

Before using the serial interface on the camera, one or more virtual COM ports have to be installed on the PC. Most systems support up to 255 COM ports; COM1 to COM8 are often assigned operating system functions by default. You can check the current port assignment in the Device Manager on your computer. Some older systems may not have more than eight ports; in that case you will need to assign the camera to one of these ports.

- COM port
  In the drop down box, select the number of the port you want to install (default: 100). COM ports in use are marked "(used)" in the list.
- Install COM port on the computer (COM100)
  Click this button to install the selected virtual COM port.
  During the first installation of a virtual COM port, an additional broadcast port with number 255 is installed. Data sent to this port will be forwarded to all paired cameras.
  You can install any number of virtual COM ports on a single system.
- Delete COM port (COM100)
  With this button, you can release a COM port that is marked "used." If the port number has been saved in that camera, it will be deleted in the camera, too. To release a COM port, select it in the drop down box and then click this button.
- Save COM port in the camera (COM100)
  Click this button to assign the selected port number to the camera. The port number is saved in the camera's non-volatile memory and retained even when the camera is switched off. You can look up the assigned port number in the Camera Manager's expert mode. A COM port number can also be saved in a camera without a virtual COM port installed on the PC.

Note

If you want to control more than one DCC3240x camera from a PC, each camera should be assigned a unique port number. If multiple cameras are assigned the same port number, only the port of the first camera that is opened will be used.

 © 2015 Thorlabs Scientific Imaging

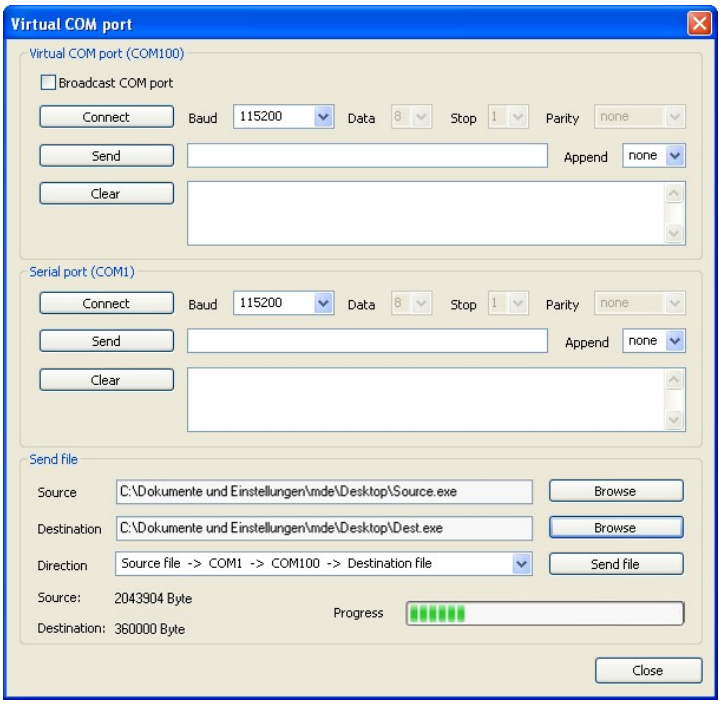
To send data via the serial interfaces of multiple cameras, you can use the broadcast port with number 255. Before connecting to the broadcast port, ensure that all the cameras that are to receive the broadcast have been opened.

Testing the serial interface on the DCC3240x camera

Note

To avoid transmission errors, please ensure that both the camera and the receiving end use the same communication parameters (baud rate, data bits, stop bits, parity). Further information on the communication parameters is provided in the Serial interface DCC3240x chapter.

- Send / Receive
  Clicking this button opens a dialog box for transferring data through the COM port. The dialog box is provided as the `uc480ComportDemo.exe` sample program together with the C++ source code and is included in the uc480 SDK.
  This program allows sending ASCII characters to the COM port assigned to a camera. The characters are output unchanged on the camera's serial port. To check the proper functionality, you can connect a PC to the camera's serial port and read the transmitted characters on the PC's COM port.



- Baud
  In this drop down box, you can change the data transfer rate of the serial interface.
- Append
  This drop down box allows appending the special characters "CR" (Carriage Return) and "LF" (Line Feed) to the ASCII text you want to transmit. Some devices with serial interface require ASCII strings to be terminated with CR/LF.
- Send file
  Using these functions, you can send a file in either direction (output on the camera's virtual COM port or output on the PC's COM port).

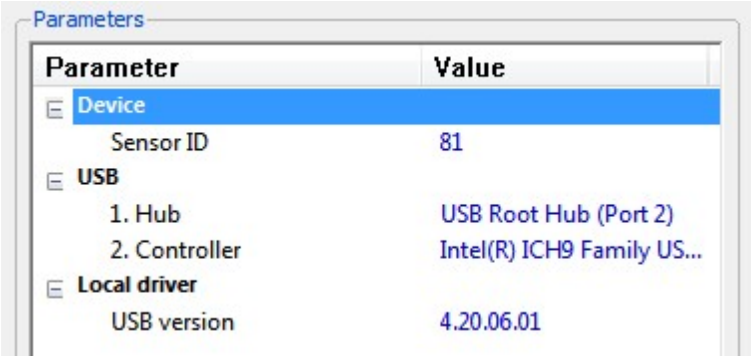
Note

Since the sample program has to open the camera, please make sure the selected camera is not used by other applications at the same time.

3.4.1.7 Parameters

This box displays the parameters of the camera you have selected in the camera list. The parameters box is only shown when Expert mode is active.

The parameters are organized in a tree structure. Only the information that applies to the selected camera is shown. The data displayed in the camera list is not repeated in the "Parameters" box. The data shown in the tree structure cannot be changed.



- Device
  - o Sensor ID
- USB
  - o Hub
    Indicates which hub and port a USB camera is connected to. In addition, the full path through all hubs to the USB controller on the computer is displayed.
  - o Controller
    Indicates the USB controller to which the camera is connected.
- Local driver
  - o Indicates the USB version of the camera driver

 © 2015 Thorlabs Scientific Imaging

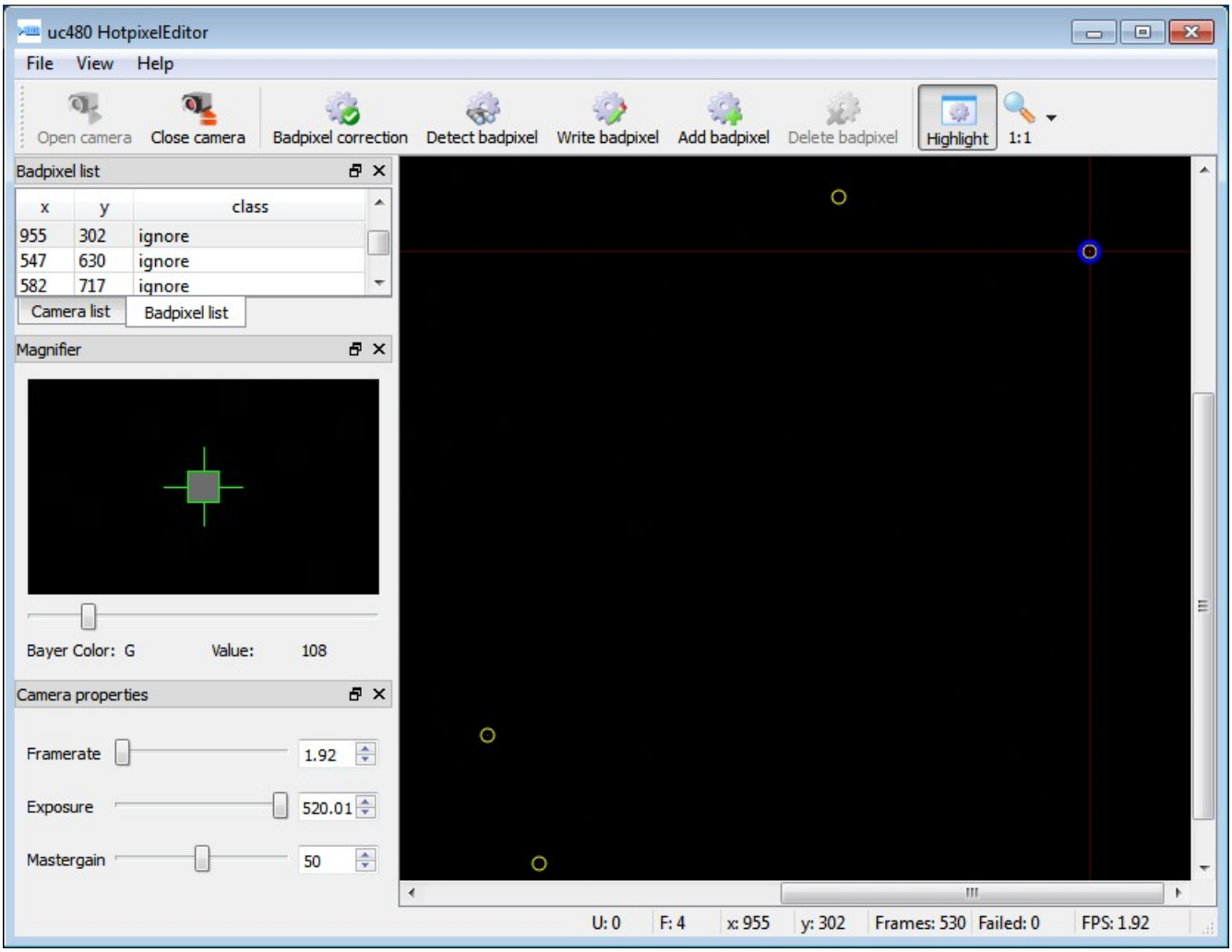
3.4.2 uc480 Hotpixel Editor

During manufacturing, each DCx camera is tested and calibrated for hot pixels (aka bad pixels), which are caused by technical reasons (see also Camera basics: Sensor: Hot pixels). In some cases, users may wish o extend this factory calibration. Using the uc480 Hotpixel Editor, you can now edit the sensor hot pixel list stored in the camera.

The uc480 Hotpixel Editor can be accessed as follows:

- `C:\Program Files\Thorlabs\Scientific Imaging\DCx Camera Support\Program\HotpixelEditor\uc480HotpixelEditor.exe`

After program start, the uc480 Hotpixel Editor shows the following window:



In the toolbar on the top the following functions are provided:

- Open camera
  Opens the camera that is marked in the camera list.
- Close camera
  Closes the camera that is marked in the camera list.
- Badpixel correction
  Turns the hot pixel correction on/off.
- Detect Badpixel
  Opens the dialog for the automatic hot pixel detection.
- Write Badpixel
  Writes the list of the hot pixel to the EEPROM.

- Add Badpixel
  Writes the marked hot pixel in the hot pixel list of the program.
- Delete Badpixel
  Deletes a hot pixel from the hot pixel list of the program.
- Highlight
  Mark bad pixel with a circle in order to improve localization. Yellow marker stands for a factory calibrated bad pixel, green marker - for user defined bad pixel.
- 1:1
  Zoom factor of the shown image.

Camera and bad pixel list

Tab Camera List:

Recognized cameras are shown with status in the camera list. Via the context menu (right click to the appropriate camera), a camera can be opened and closed. Cameras with the status "in use" cannot be opened.

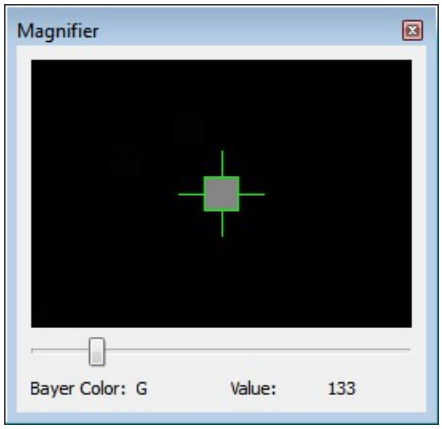
Tab Badpixel List:

In the bad pixel list the coordinates and the class of bad pixels are displayed.

Class "user": added by the user

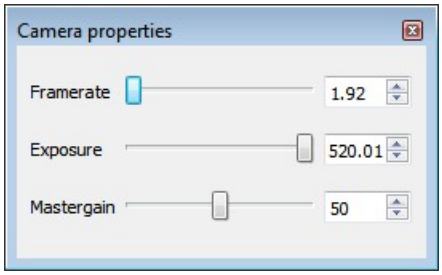
Class "ignore": factory calibrated hot pixels

Magnifier function



The magnifier allows to exactly mark hot pixels. "Value" displays the relative pixel intensity.

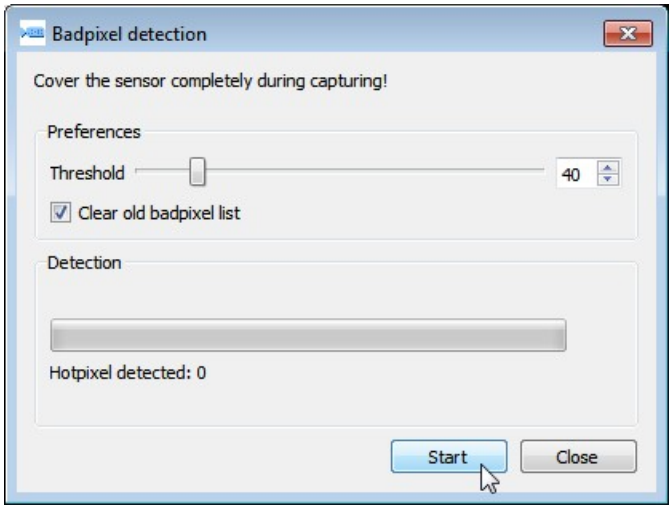
Camera properties



Here, the camera settings can be changed for a better hot pixels visualization. By default, exposure time is set to max. and master gain to 50. Note that the frame rate is limited to lower values.

© 2015 Thorlabs Scientific Imaging

Detecting hot pixel automatically



opens the "Badpixel detection" dialog. Bad pixels are detected and can be added to the "user" class list.

- "Threshold": Any pixel with an intensity exceeding that threshold will be recognized as a bad (hot) pixel, where the threshold is is the minimum intensity difference to the reference intensity. The reference is the averaged over the most recent 10 frames total pixel intensities. Threshold can be set between 5 and 255.
- The "Clear old badpixel list" option removes the old user list before the search.

Note

The number of bad pixels, stored to the EEPROM, is limited. If the number of detected bad pixels exceeds the memory capacity, please increase the bad pixel detection threshold and repeat the detection procedure.

4 Programming (SDK)

In addition to the drivers, the uc480 software Development Kit (SDK) includes the uc480 Camera Manager, the uc480 Viewer and the uc480 API programming interface for creating your own DCx programs under Windows and Linux. Numerous demo applications make it easy for you to get started with programming. The uc480 API offers you over 150 commands with which you can access all the parameters and functions of your DCx camera. This chapter contains all the information you need to integrate the DCx camera in your own applications using the uc480 API.

Note

Older Functions

We are continuously extending and enhancing the uc480 API. The resulting product upgrades sometimes require replacing obsolete functions with new ones. If it is necessary to continue working with the older functions, it is possible to add the `uc480_deprecated.h` header file additionally to the `ueye.h` header file. The `uc480_deprecated.h` header file contains all obsolete function definitions and constants which are no longer part of the `ueye.h` header file.

See also:

- First steps to uc480 programming
- How to proceed
- Function descriptions
- AVI function descriptions
- Obsolete functions
- Programming notes
- Lists

© 2015 Thorlabs Scientific Imaging

4.1 First Steps to uc480 Programming

This chapter shows the most important functions of the uc480 API for integrating your camera into your own applications. You will find comprehensive lists of the API functions, sorted by task, in the How to proceed chapter.

The **uc480 SimpleLive** and **uc480 SimpleAcquire C++** programming samples included in the SDK illustrate the steps described below.

For information on required include files (uc480 API and header) see Programming notes chapter.

Select a display mode

The uc480 API provides different modes you can use to display the camera's images on the PC. To quickly show a live image under Windows, it is easiest to use the Direct3D mode. Under Linux the OpenGL mode can be used.

This mode has the advantage that no image memory has to be allocated, and that image capture is handled by the driver. Call `is_SetDisplayMode()` to select the display mode. You can then customize the Direct3D mode by using `is_DirectRenderer()`.

For advanced users:

You can also access the image data directly by selecting the Bitmap (DIB) mode. To use DIB mode, you first have to allocate one or more memories by using `is_AllocImageMem()`, add them to a memory sequence, if required, and then activate a memory with `is_SetImageMem()` before each image capture. To show the image on-screen, call the `is_RenderBitmap()` function after each completed image capture. From the events or messages you can see when an image is available for display.

See also:

- How to proceed: Display mode selection

Capture images

Recording live images with the DCxCamera is very simple. Just call the `is_CaptureVideo()` function and the camera captures the live images at the default frame rate. To capture single frames, use the `is_FreezeVideo()` function. Every DCx camera of course also provides different trigger modes for image capture. Use `is_SetExternalTrigger()` to activate the desired mode before starting the image capture.

See also:

- How to proceed: Image capture

Adjust the frame rate, brightness and colors

To change the frame rate, for example, you call `is_SetFrameRate()`. With `is_SetColorMode()` you set the color mode. Image brightness is adjusted through the exposure time set with `is_Exposure()`. You can also implement automatic control of image brightness and other parameters by using `is_SetAutoParameter()`.

If you are using a color camera, you should activate color correction in order to achieve rich vibrant colors for on-screen display (`is_SetColorCorrection()`). To adapt a color camera to the ambient light conditions, it is essential to carry out white balancing. This is also done using the `is_SetAutoParameter()` function.

See also:

- How to proceed: Setting camera parameters

Save an image

Use the `is_ImageFile()` function to save the current image as a BMP or JPEG file. To save a specific image, it is better to use the Snap function (single frame mode) than the Live function (continuous mode).

See also:

- How to proceed: Saving images and videos

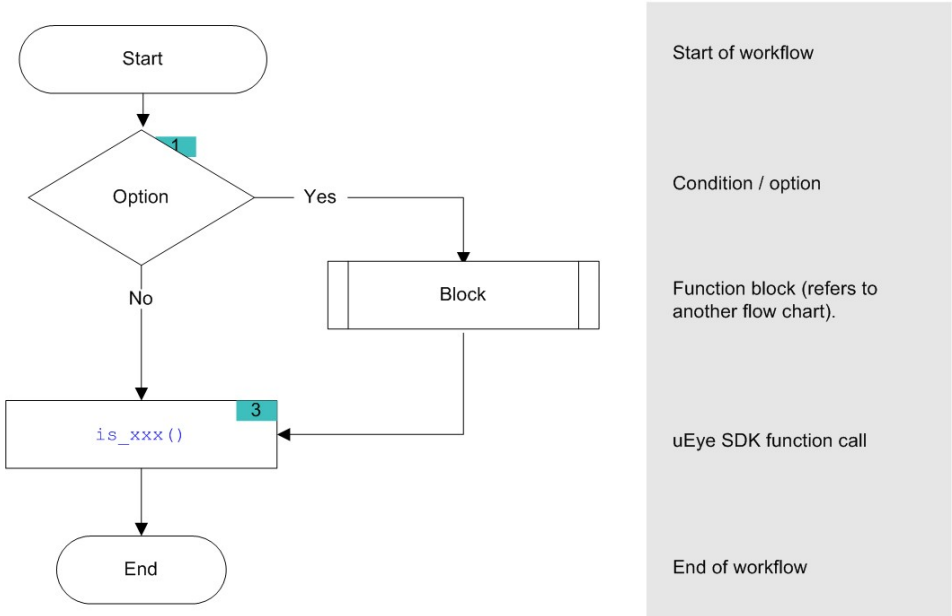
Close the camera

When you want to exit your application, close the camera with `is_ExitCamera()`. The camera and the allocated memory are automatically released. All previously set camera parameters will be lost, however. So, if you want to save specific settings, use the `is_ParameterSet()` function before closing the camera. The next time you start the application, you can simply load the settings again by using the same function.

© 2015 Thorlabs Scientific Imaging

4.2 How to Proceed

This chapter shows function blocks and workflows for important camera functions. The charts are structured as follows:



Hint

Click the function names in the flowcharts to open the corresponding function description!

4.2.1 Preparing Image Capture

- Querying information: Before you open one or more DCx Cameras, we recommend querying some key information.
- Opening and closing a camera
- Allocating an image memory: This is necessary if you want to access image memory contents or if you are not using Direct3D or OpenGL for image display.
- Creating an image memory sequence is required when capturing live images.

4.2.1.1 Querying Information

It is recommended to query the following important information before opening one or more DCx Cameras.

`is_GetNumberOfCameras()`	Determines the number of cameras connected to the system.
`is_GetCameraList()`	Returns information on all connected cameras.

It is also very useful to have the message boxes for error output enabled during the programming process

`is_SetErrorReport()`	Enables/disables dialog messages for error output.

With the following functions, you can read out additional information on cameras and software.

`is_CameraStatus()`	Returns the event counters and other information. Enables standby mode.

is_GetCameraInfo()	Returns the camera information of an opened camera.
is_GetDLLVersion()	Returns the version of the uc480.dll.
is_GetOsVersion()	Returns the operating system version.

4.2.1.2 Opening and Closing the Camera

The following functions are required to open and close a DCx camera.

is_InitCamera()	Hardware initialization
is_ExitCamera()	Closes the camera and releases the created image memory.

When multiple cameras are used on one system you should assign every camera a unique camera ID.

is_SetCameraID()	Sets a new camera ID.

4.2.1.3 Allocating Image Memory

When you are programming an application that

- requires direct access to the image data in stored in memory, or
- uses Bitmap mode (DIB) for display

use the following functions to allocate and manage image memories (see also Quick start: Image display).

is_AllocImageMem()	Allocates an image memory.
is_SetAllocatedImageMem()	The user provides pre-allocated memory for image capturing.
is_FreeImageMem()	Releases an allocated image memory.

An image memory has to be activated before each image capture:

is_SetImageMem()	Makes an image memory active.

To query image memory information and access the data in the image memories, you can use these functions:

is_CopyImageMem()	Copies the image to the user-defined memory.
is_CopyImageMemLines()	Copies selected image lines to the user-defined memory.
is_GetActiveImageMem()	Returns the number and address of the active image memory.
is_GetImageMem()	Returns the pointer to the starting address of the image memory.
is_GetImageMemPitch()	Returns the line offset used in the image memory.
is_InquireImageMem()	Returns the properties of an image memory.

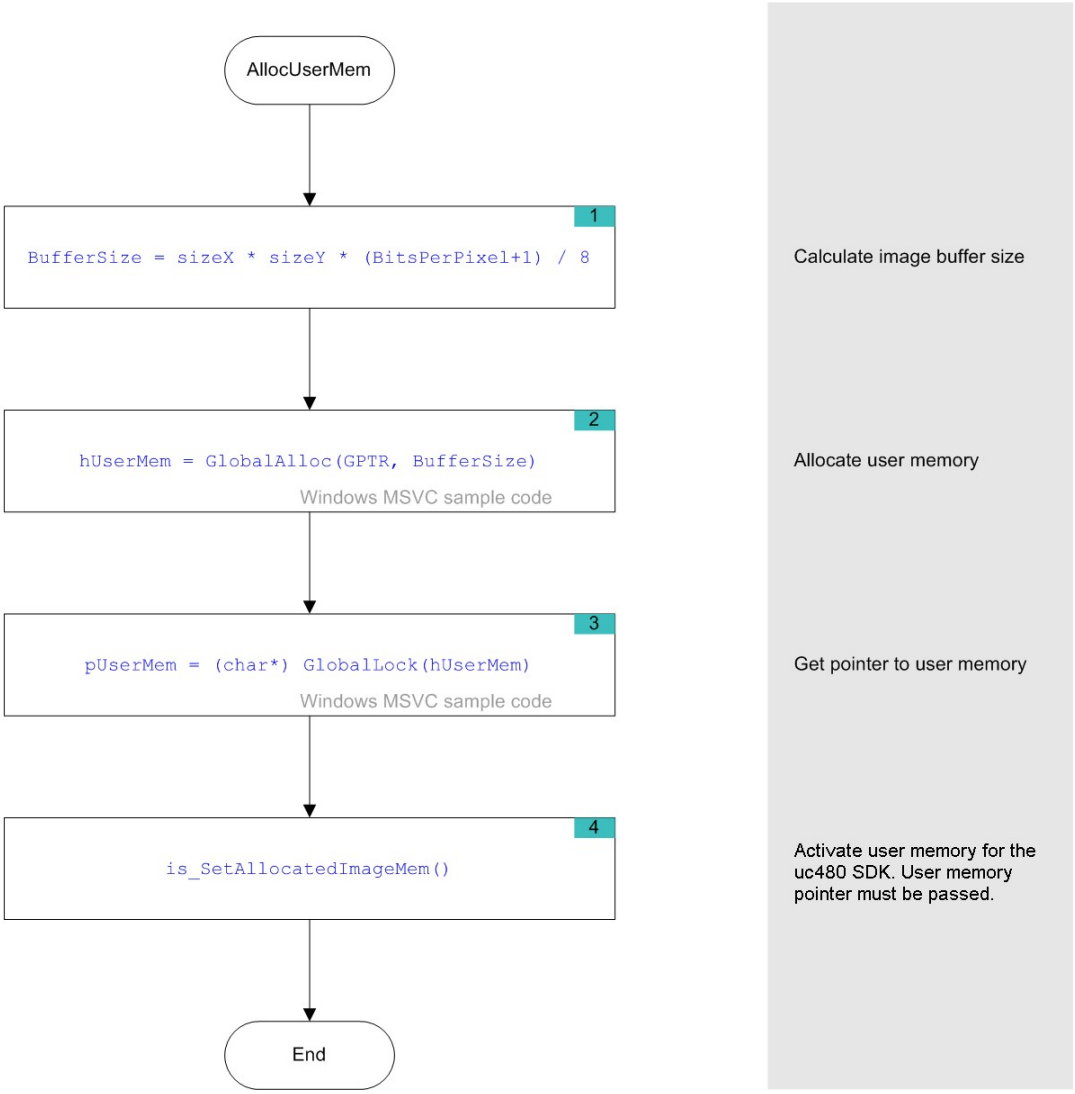
Note

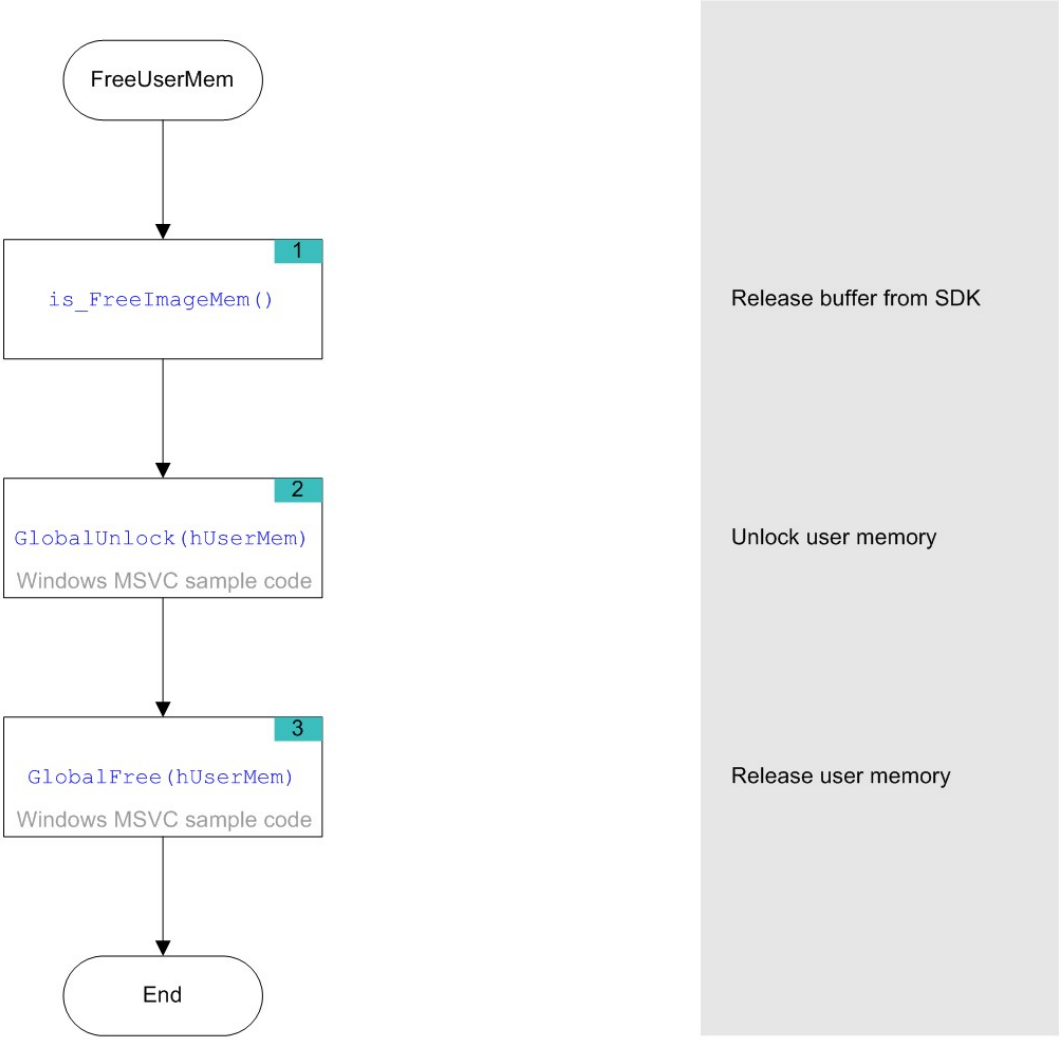
Image memory sequences should be used for frame sequence capture.

© 2015 Thorlabs Scientific Imaging

Flowchart: Allocating memory

Click in the figure to get help on the functions.





4.2.1.4 Image Memory Sequences

When you are capturing and displaying frame sequences (e.g. live display), it is advisable to provide appropriate image memory sequences. The uc480 driver offers a set of easy-to-use features for this purpose. For example, the system automatically cycles through the specified sequence of image memories and can generate an event when it reaches the end of a sequence cycle.

Before you can use a memory sequence, you have to allocate the relevant image memories (see Allocating image memory).

`is_AddToSequence()`	Adds image memory to the sequence list.
`is_ClearSequence()`	Deletes the entire sequence list.
`is_GetActSeqBuf()`	Determines the image memory currently used for the sequence.
`is_SetImageMem()`	Makes the indicated image memory the active memory.
`is_LockSeqBuf()`	Protects the sequence image memory from being overwritten.
`is_UnlockSeqBuf()`	Releases the sequence image memory for overwriting.

© 2015 Thorlabs Scientific Imaging

4.2.2 Selecting the Display Mode

Note

The uc480 driver provides different modes for displaying the captured images. We recommend using the Bitmap mode or the Direct3D functions (only Windows) or OpenGL functions, depending on your specific application.

For further information on the different display modes, see Basics: Image display modes.

Select the desired mode. The display mode has to be set before you start image capture.

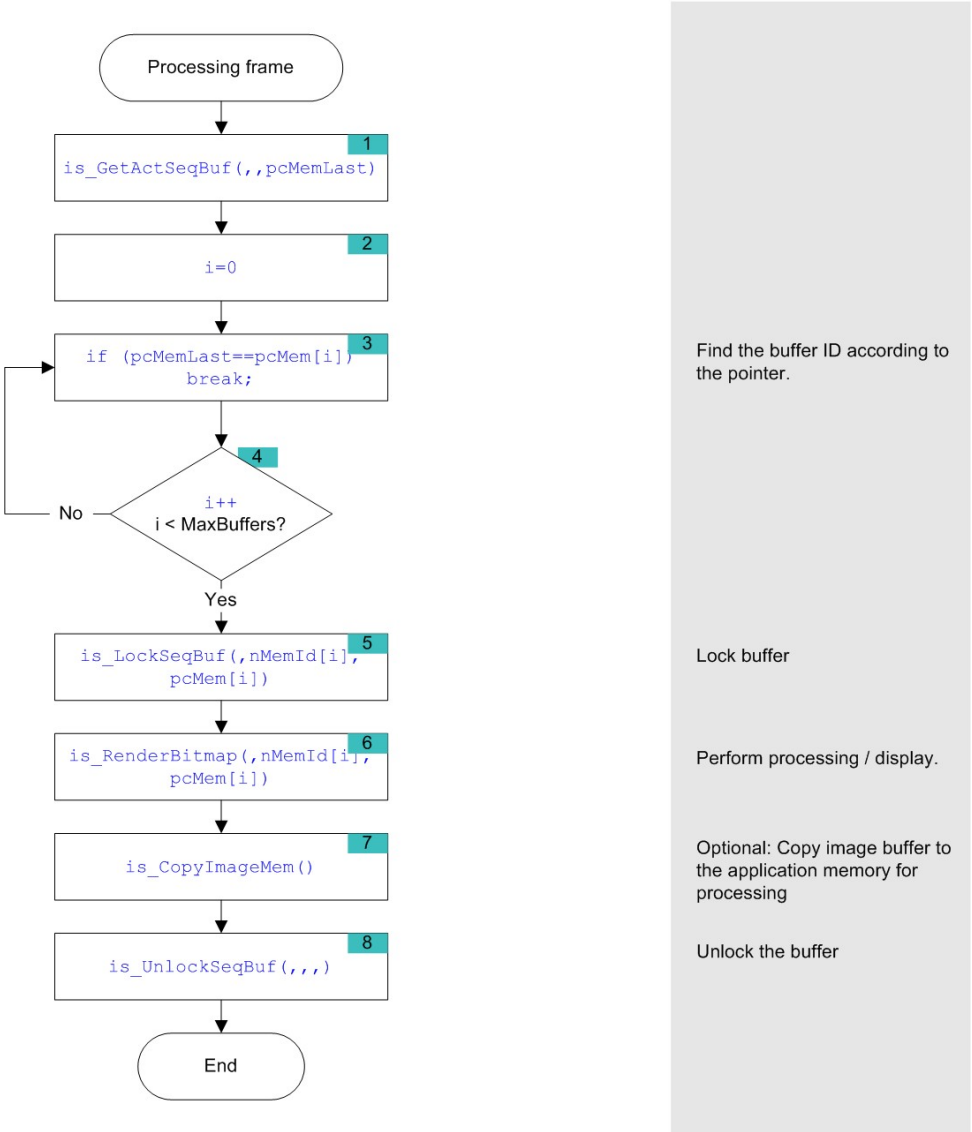
`is_SetDisplayMode()`	Selection of the display mode.

When Bitmap mode (DIB) is active, image display has to be called explicitly for each image.

`is_RenderBitmap()`	Outputs the contents of the active image memory to a window.
`is_SetDisplayPos()`	Enables offsetting the image output inside the window.

Flowchart: Image display in DIB mode

Click in the figure to get help on the functions.



4.2.3 Capturing Images

- DCx Cameras support the capture of single frames (snap) and frame sequences (live) in trigger mode and untriggered (freerun) mode. Select the image capture mode that best meets your application requirements.
- Using events or messages, the uc480 driver can provide information to an application, e.g. about the completion of image capture. You will need events and messages, for example, when you are using image memory sequences.

4.2.3.1 Image Capture Modes

For more information on the capture modes of the DCx Cameras see also Camera basics: Freerun and Camera basics: Trigger.

Freerun mode

In freerun mode, the camera sensor captures one image after another at the set frame rate. Exposure of the current image and readout/transfer of the previous image data are performed simultaneously. This allows the maximum camera frame rate to be achieved. The frame rate and the exposure time can be set separately. The captured images can be transferred one by one or continuously to the PC.

If trigger mode is active, you need to disable it with `is_SetExternalTrigger()` before activating freerun mode.

- Single frame mode (snap mode)
  When `is_FreezeVideo()` is called, the next image exposed by the sensor is transferred. You cannot use the DCxCamera flash outputs in this mode.
- Continuous mode (live mode)
  When `is_CaptureVideo()` is called, images are captured and transferred continuously. You can use the DCxCamera flash outputs.

Trigger mode

In trigger mode, the sensor is on standby and starts exposing on receipt of a trigger signal. A trigger event can be initiated by a software command (software trigger) or by an electrical signal via the camera's digital input (hardware trigger). For the specifications of the electrical trigger signals, see the Specifications: Electrical specifications chapter.

The trigger mode is selected using `is_SetExternalTrigger()`.

- Software trigger mode
  When this mode is enabled, calling `is_FreezeVideo()` immediately triggers the capture of an image and then transfers the image to the PC. If `is_CaptureVideo()` is called, the triggering of image capture and the transfer of images are performed continuously.
- Hardware trigger mode
  When this mode is enabled, calling `is_FreezeVideo()` makes the camera ready for triggering just once. When the camera receives an electrical trigger signal, one image is captured and transferred.
  If you call `is_CaptureVideo()`, the camera is made ready for triggering continuously. An image is captured and transferred each time an electrical trigger signal is received; the camera is then ready for triggering again (recommended procedure).
- Freerun synchronization
  In this mode, cameras running in freerun mode (live mode, see above) can be synchronized with an external trigger signal. The cameras still remain in freerun mode. The trigger signal stops and restarts the current image capture process. You can use this mode to synchronize multiple

 © 2015 Thorlabs Scientific Imaging

cameras that you are operating in the fast live mode. Not all camera models support this mode (see `is_SetExternalTrigger()`).

Notes

- The freerun synchronization mode is currently not supported by DCx Cameras.
- In trigger mode, the maximum frame rate is lower than in freerun mode because the sensors expose and transfer sequentially. The possible frame rate in trigger mode depends on the exposure time.

  Example: At the maximum exposure time, the frame rate is about half as high as in freerun mode; at the minimum exposure time, the frame rate is about the same.

Overview on image capture modes

Image capture	Trigger	Function calls	Allowed flash modes		Frame rate
			Standard	Global Start	
Continuous	Off	is_SetExternalTrigger (OFF) is_CaptureVideo()	X		Freely selectable
	Software	is_SetExternalTrigger (SOFTWARE) is_CaptureVideo()	X	X	Depending on exposure time and trigger delay
	Hardware	is_SetExternalTrigger (e.g. HI_LO) is_CaptureVideo()	X	X	Depending on exposure time and trigger delay
	Freerun sync.	is_SetExternalTrigger (e.g. HI_LO_SYNC) is_CaptureVideo()	X		Freely selectable
Single frame	Off	is_SetExternalTrigger (OFF) is_FreezeVideo()			Freely selectable
	Software	is_SetExternalTrigger (SOFTWARE) is_FreezeVideo()	X	X	Depending on exposure time and trigger delay
	Hardware	is_SetExternalTrigger (e.g. HI_LO) is_FreezeVideo()	X	X	Depending on exposure time and trigger delay

Timeout values for image capture

When you call `is_FreezeVideo()` or `is_CaptureVideo()`, the timeout value for the image capture is determined from the Wait parameter. If no image arrives within this timeout period, a timeout error message is issued. Under Windows, a dialog box is displayed if you have enabled error reports (see `is_SetErrorReport()`). Information on the error cause can be queried using `is_CaptureStatus()`.

© 2015 Thorlabs Scientific Imaging

The following table shows the effect of the Wait parameter depending on the image capture mode:

Parameter `Wait`	**Image capture mode**	**Function returns**	**Timeout for 1st image**	**Timeout for subsequent images**[*1]
`IS_DONT_WAIT`	HW trigger	Immediately	API default or user-defined value[*3]	API default or user-defined value[*3]
`IS_WAIT`	HW trigger	When 1st image in memory	API default or user-defined value[*3]	API default or user-defined value[*3]
Time `t` Value range [4… 429496729]	HW trigger	When 1st image in memory	Time t in steps of 10 ms (40 ms to approx. 1193 h)	API default or user-defined value[*3]
`IS_DONT_WAIT`	Freerun/SW trigger	Immediately	Calculated internally by API[*2]	Calculated internally by API[*2]
`IS_WAIT`	Freerun/SW trigger	When 1st image in memory	Calculated internally by API[*2]	Calculated internally by API[*2]
Time `t` Value range [4… 429496729]	Freerun/SW trigger	When 1st image in memory	Time t in steps of 10 ms (40 ms to approx. 1193 h)	Calculated internally by API[*2]

*1 Only with continuous image capture using is_CaptureVideo()

*2 The timeout is calculated from the exposure time setting, the image transfer time (depending on the pixel clock) and the optional trigger delay (see `is_SetTriggerDelay()`); it is at least 40 ms.

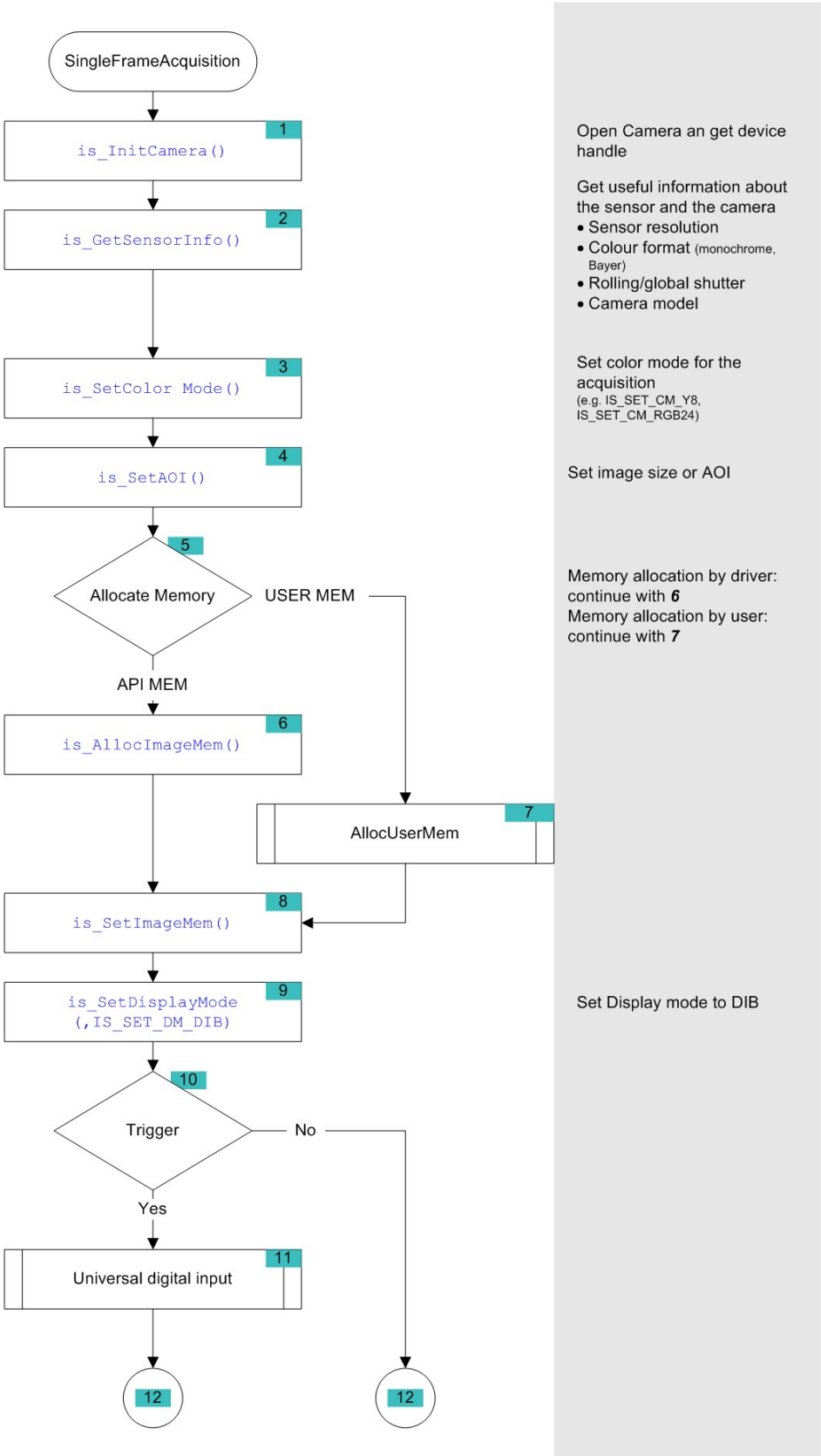
*3 The default value of the uc480 API is 60 s. User-defined values can be set using the `is_SetTimeout()` function.

Function list

`is_CaptureVideo()`	Captures a live video.
`is_FreezeVideo()`	Captures an image and writes it to the active image memory.
`is_ForceTrigger()`	Forces image capture in hardware trigger mode.
`is_HasVideoStarted()`	Returns whether the capture process has been started or not.
`is_IsVideoFinish()`	Returns whether the capture process has been terminated or not.
`is_SetSensorTestImage()`	Enables test image output from sensor (all cameras).
`is_StopLiveVideo()`	Terminates the capturing process (live video or single frame).

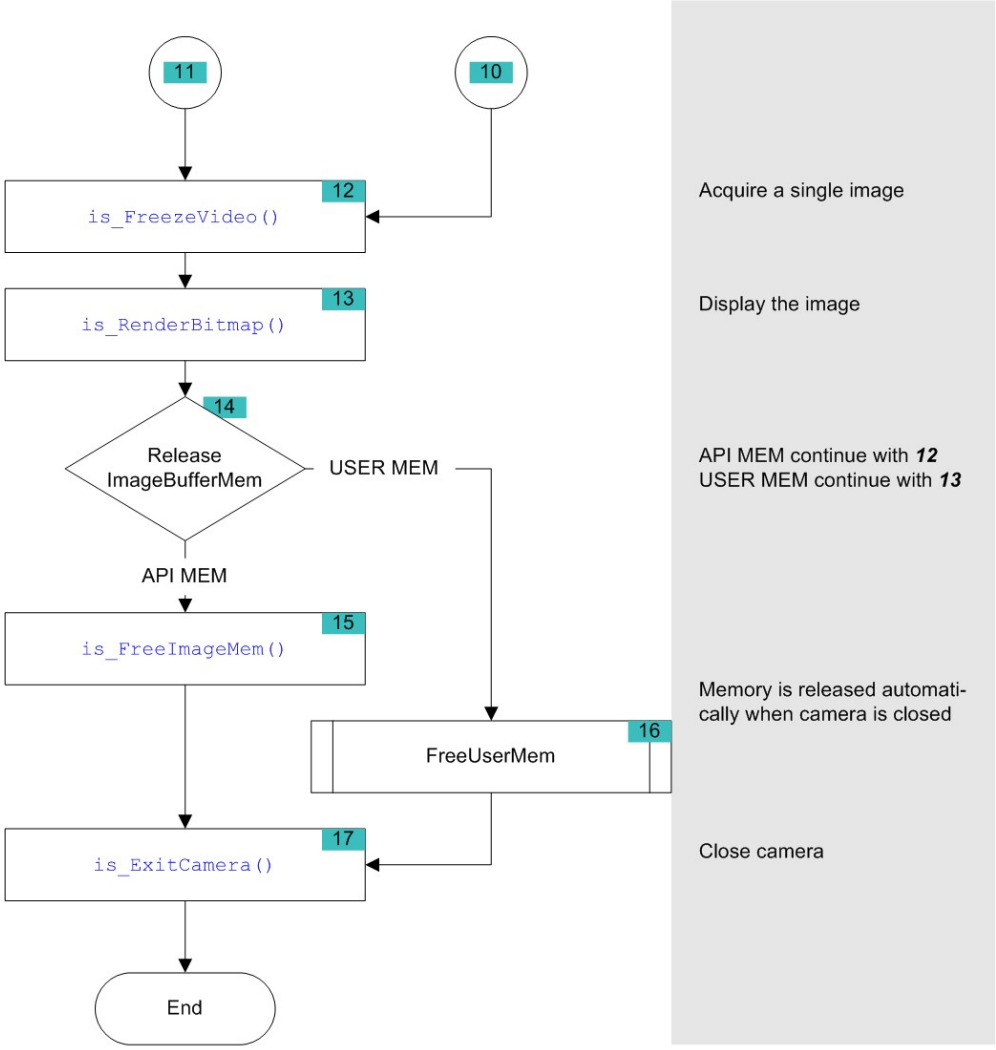
Flowchart: Single Capture

Click in the figure to get help on the functions.



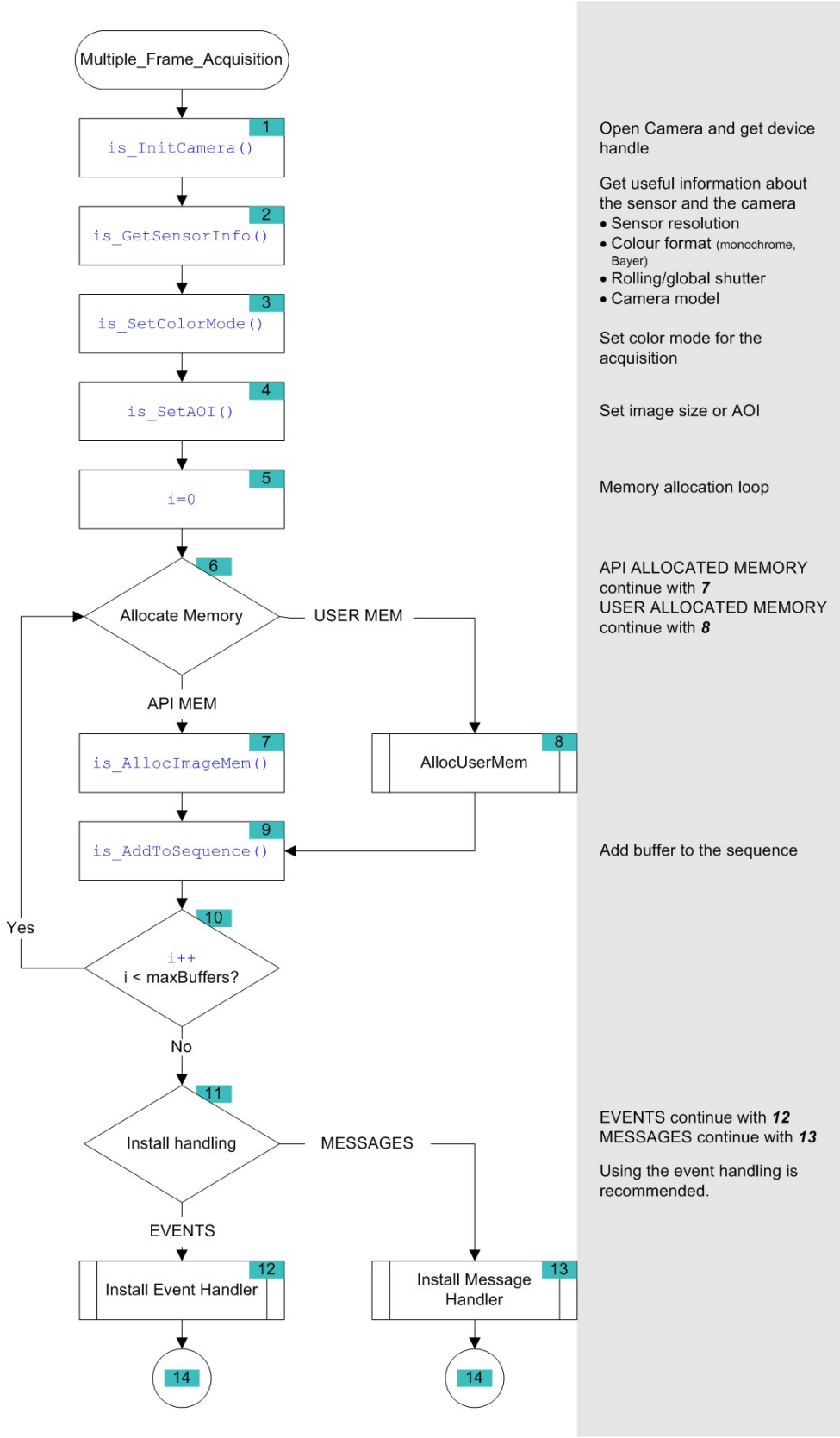
Flowchart - Single Capture (1 of 2)

© 2015 Thorlabs Scientific Imaging



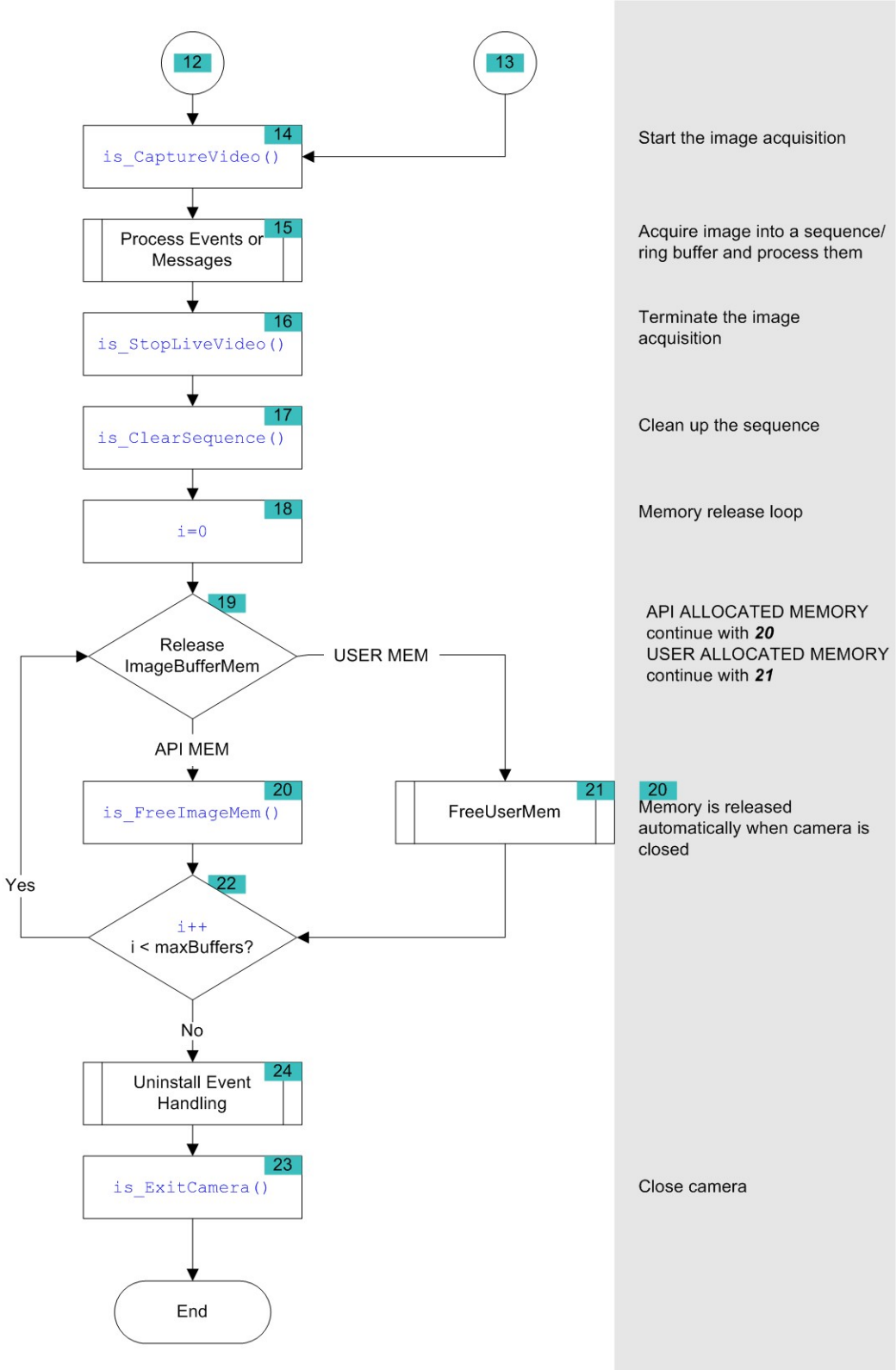
Flowchart - Single Capture (2 of 2)

Flowchart: Sequence Capture



Flowchart - Capturing a frame sequence (1 of 2)

© 2015 Thorlabs Scientific Imaging



Flowchart - Capturing a frame sequence (2 of 2)

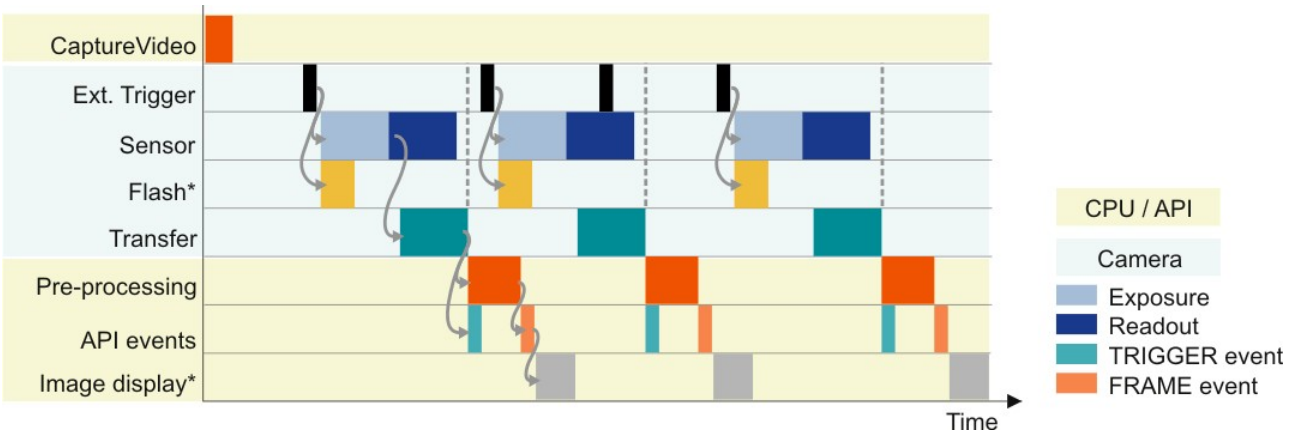
4.2.3.2 Event / Message Handling

Triggering events for single frame capture

The following figure shows the time sequence when triggering the `IS_SET_EVENT_EXTTRIG` and `IS_SET_EVENT_FRAME` events. The camera is prepared for triggered capture using the `is_SetExternalTrigger()` command. An incoming trigger signal at the camera starts the exposure and the subsequent image transfer. Upon completion of the data transfer, the `IS_SET_EVENT_EXTTRIG` event signals that the camera is ready for the next capture. The `IS_SET_EVENT_FRAME` event is set once pre-processing (e.g. color conversion) is complete and the finished image is available in the user memory.

Note

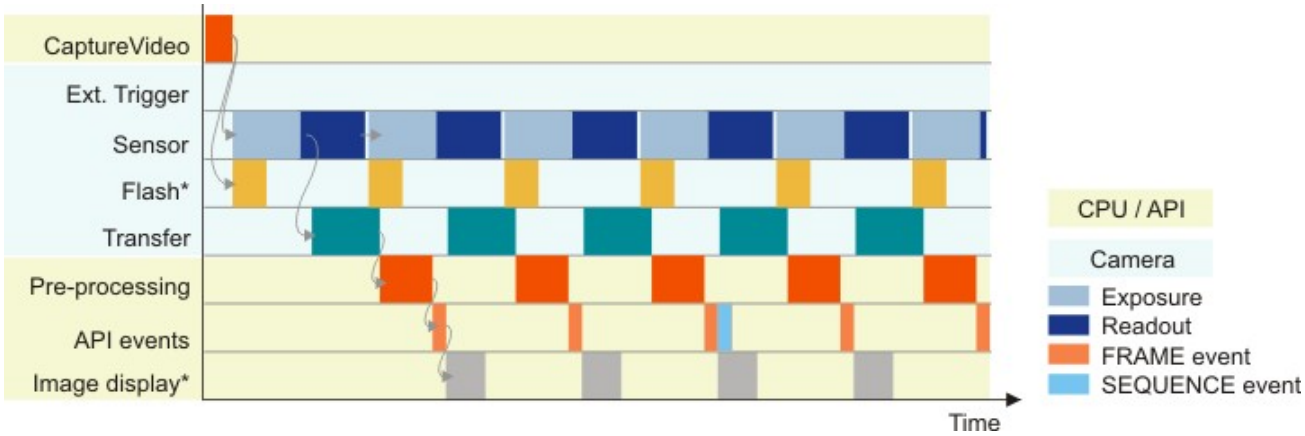
The following illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera settings. The pre-processing time depends on the API functions you are using (e.g. color conversion, edge enhancement).



* Optional function. The start time and duration of the flash signal are defined by the "Flash delay" and "Duration" parameters (see `is_IO()`).

Events in live mode (image sequence)

The following figure shows the time sequence when triggering the `IS_SET_EVENT_FRAME` and `IS_SET_EVENT_SEQ` events. The camera is set to live mode using `is_CaptureVideo()` so that it continuously captures frames. The `IS_SET_EVENT_FRAME` event is set once pre-processing (e.g. color conversion) is complete and a finished image is available in the user memory. The `IS_SET_EVENT_SEQ` event is set after one cycle of a storing sequence has been completed (see also `is_AddToSequence()`).



© 2015 Thorlabs Scientific Imaging

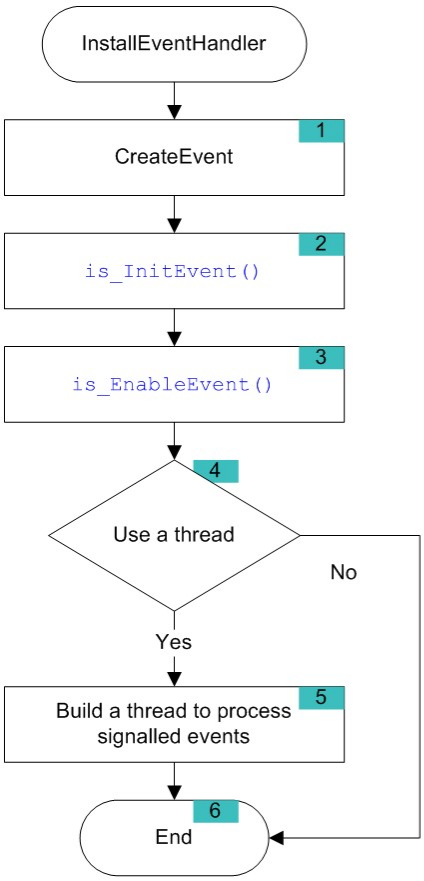
* Optional function. The start time and duration of the flash signal are defined by the "Flash delay" and "Duration" parameters (see `is_IO()`).

Function List

`is_DisableEvent()`	Disables a single event object.
`is_EnableEvent()`	Enables a single event object.
`is_EnableMessage()`	Turns the Windows messages on/off.
`is_ExitEvent()`	Closes the event handler (Windows only)
`is_InitEvent()`	Initializes the event handler (Windows only)
`is_EnableAutoExit()`	Automatically releases the camera resources when the camera is disconnected from the PC.
`is_WaitEvent()`	Waits for DCxCamera events (Linux only)

Flowchart: Enable events

Click in the figure to get help on the functions.



Create an event and receive the handle of the event object

Initialize the uc480 event handling. The event handle from CreateEvent is required.

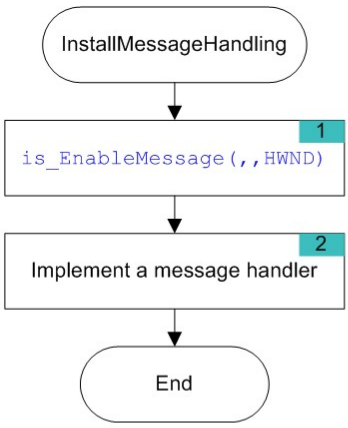
Enable a specific event.

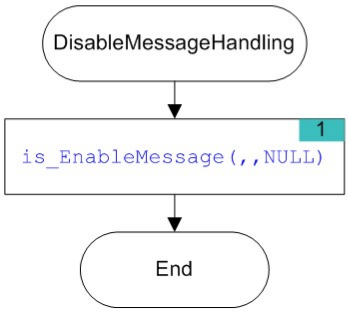
Using a thread (recommended) continue with ***5***
else continue with ***6***.

Windows functions:
WaitForSingleObject()
WaitForMultipleObjects()

© 2015 Thorlabs Scientific Imaging

Flowchart: Enabling Messages





MS Windows only

Enable the message handling.

This may be different in each programming language. Hook on the message map.

When the messages are sent to the applications main window and the GUI is locked for some reasons the uc480 messages cannot be processed.

Disable the message handling.

4.2.4 Setting Camera Parameters

- Setting and getting parameters: Using these functions, you can make settings for the camera and for image capture and preprocessing.
- The DCxCamera's automatic image control features allow automatically adjusting image brightness and image color to changing ambient conditions.
- Image preprocessing: These functions specify e.g. how color images are processed after image capture.
- Querying the camera status: With these functions, you can query additional useful information on the camera status.
- Using the camera EEPROM: All DCx Cameras have a non-volatile EEPROM where you can save the camera settings or any other information.

4.2.4.1 Setting and Getting Parameters

Capture parameters

This set of functions specifies the camera's image capture parameters, such as exposure, pixel clock and frame rate:

`is_ColorTemperature()`	Sets the color temperature
`is_Exposure()`	Returns the adjustable exposure range.
`is_GetFramesPerSecond()`	Returns the current frame rate in live mode.
`is_GetFrameTimeRange()`	Returns the adjustable frame rate range.
`is_PixelClock()`	Returns the adjustable pixel clock range.
`is_SetAutoParameter()`	Enables/disables automatic imaging functions.
`is_Blacklevel()`	Turns black level correction on / off.
`is_Exposure()`	Sets the exposure time.
`is_SetFrameRate()`	Sets the frame rate.
`is_SetGainBoost()`	Sets additional sensor hardware gain boost.
`is_SetGamma()`	Sets the gamma value (digital post-processing).
`is_SetHardwareGain()`	Enables the sensor hardware gain.
`is_SetHWGainFactor()`	Sets the sensor hardware gain factor.
`is_PixelClock()`	Sets the pixel clock frequency.
`is_ResetToDefault()`	Resets the camera parameters to its default values.

Image geometry

This set of functions lets you influence the image geometry for image capture, e.g. the area of interest:

`is_ImageFormat()`	Sets a predefined image size
`is_AOI()`	Sets the size and position of an area of interest (AOI) or of a reference AOI for auto imaging functions.
`is_SetBinning()`	Sets the binning modes.
`is_SetRopEffect()`	Makes real-time geometry changes to an image

 © 2015 Thorlabs Scientific Imaging

	(Rop = raster operation)
is_SetSensorScaler()	Scales the image in the camera
is_SetSubSampling()	Sets the subsampling modes.

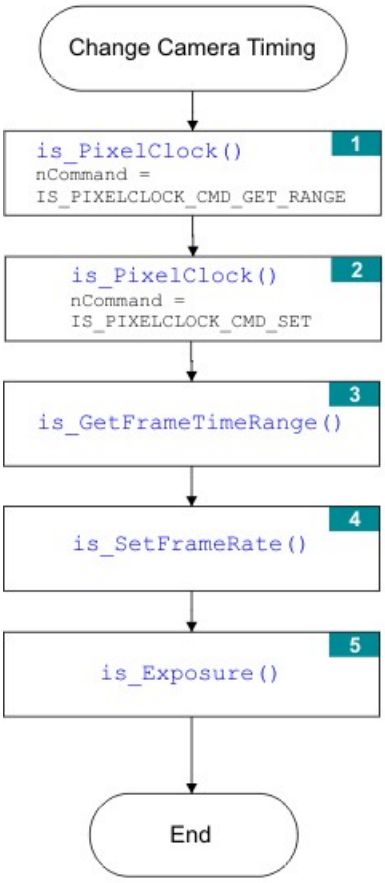
Processing image data

The following set of functions refers to the further processing of image data in the PC:

is_GetColorDepth()	Determines the desktop color mode set in the graphics card.
is_GetTimeout()	Returns the user-defined timeout values.
is_HotPixel()	Enables and configures the hot pixel correction.
is_SetColorConverter()	Selects Bayer conversion mode.
is_SetColorCorrection()	Sets color correction.
is_SetColorMode()	Selects a color mode.
is_Convert()	Conversion parameters for raw Bayer conversion.
is_EdgeEnhancement()	Sets edge enhancement.
is_SetSaturation()	Sets the image saturation (digital post-processing).
is_SetSensorTestImage()	Enables test image output from sensor.
is_SetTimeout()	Sets user-defined timeout values.

Flowchart: Changing camera timing

Click in the figure to get help on the functions.



Recommended order of function calls for timing changes.

Return the possible range for the pixel clock.

The pixel clock determines the range of frame rate settings.

Return the possible range of the frame rate.

The frame rate defines the range for the exposure time.

Return the possible exposure range and set the exposure time.

© 2015 Thorlabs Scientific Imaging

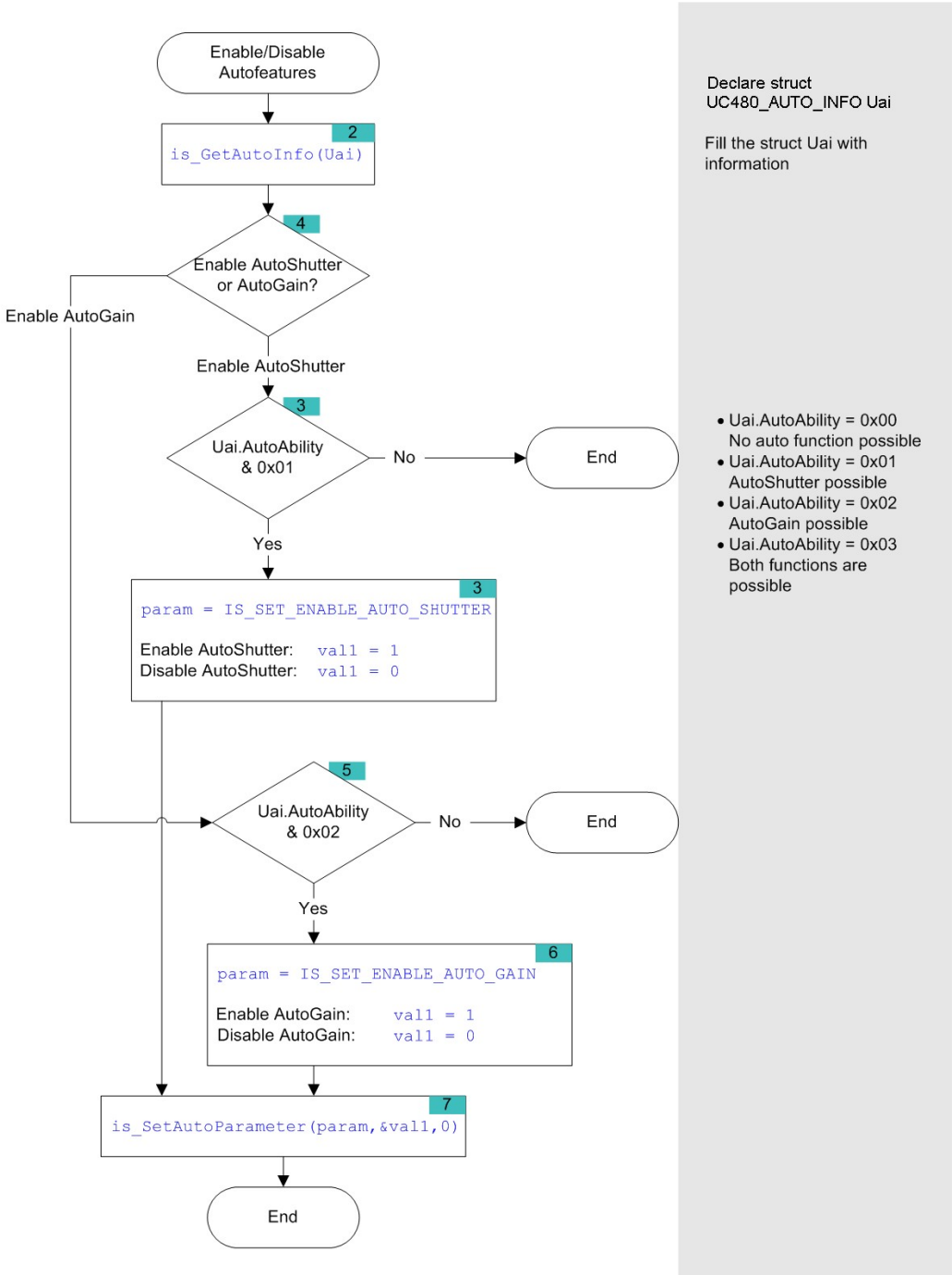
4.2.4.2 Automatic Image Control

The uc480 driver provides various options to automatically adjust the image capture parameters to the lighting situation. All controls are configured using the `is_SetAutoParameter()` SDK function.

For more information on the automatic image control see Camera basics: Automatic image control.

Flowchart: Enable Auto Brightness

Click in the figure to get help on the functions.



4.2.4.3 Image Pre-processing

Bayer conversion

The following functions enable and adjust the Bayer conversion (see Color filter (Bayer filter)).

is_Convert()	Converts a Bayer raw image into the desired output format
is_GetColorConverter()	Returns the currently set Bayer conversion mode
is_SetBayerConversion()	Sets the algorithm for Bayer conversion
is_SetColorConverter()	Sets the algorithm for Bayer conversion in the camera (not applicable for DCx Cameras)

Lookup table

Using lookup table (LUT) functions, you can e. g. adjust brightness or contrast after the acquisition.

is_GetCameraLUT()	Read out current hardware LUT
is_GetImageHistogram()	Computes a histogram for the image buffer passed to the function

4.2.4.4 Get Camera Status

Using these functions, you can read out additional useful information on the camera status.

is_CameraStatus()	Returns the event counters and other information. Enables standby mode.
is_GetAutoInfo()	Returns status information on the auto features.
is_GetCameraList()	Returns information on all connected cameras.
is_GetCameraType()	Returns the camera type.
is_CaptureStatus()	Displays information on errors that have occurred.
is_GetError()	Displays errors that have occurred.
is_GetUsedbandwidth()	Returns the bus bandwidth (in Mbyte/s) currently used by all initialized or selected cameras.
is_GetVsyncCount()	Returns the VSYNC counter. It will be incremented by 1 each time the sensor starts capturing an image.
is_SetErrorReport()	Enables/disables dialog messages for error output.

4.2.4.5 Using the Camera EEPROM

The non-volatile EEPROM of every DCx camera can hold user data or camera settings.

is_GetCameraInfo()	Returns the factory-set information (e.g. revision information for the individual DCxCamera components).
is_GetSensorInfo()	Returns the sensor information.
is_ReadEEPROM()	Reads out the writable data area of the EEPROM.
is_WriteEEPROM()	Writes user data to the EEPROM.

 © 2015 Thorlabs Scientific Imaging

4.2.5 Saving Images and Videos

Using the uc480 API, you can

- Save and load single frames
- Capture an AVI frame sequence

4.2.5.1 Saving and Loading Single Frames

With the `is_ImageFile()` function you can save the image data of the current image memory to a BMP, PNG or JPG file, and load saved image data into an image memory.

4.2.5.2 Capturing AVIs

The functions of the `uc480_tools.dll` enable you to save images captured with the DCxCamera as sequences to an AVI file. In order to reduce the file size, the single frames are stored in the AVI container using an adjustable JPEG compression. It is possible to extract single frames from the AVI file.

AVI Capture Workflow

First initialize the AVI interface and then create a empty AVI file.

`isavi_InitAVI()`	Initializes the AVI interface.
`isavi_ExitAVI()`	Terminates and closes the AVI interface.
`isavi_OpenAVI()`	Opens an AVI file for capturing.
`isavi_CloseAVI()`	Closes an AVI file.
`isavi_GetAVIFileName()`	Returns the name of the current AVI file.

The following settings should also be done prior to starting the recording.

`isavi_SetFrameRate()`	Sets the frame rate of the AVI video.
`isavi_SetImageQuality()`	Sets the compression level/image quality of the AVI video.
`isavi_SetImageSize()`	Sets the size and offset of the input image memory.

Once the AVI file has been created, captured images are placed in a buffer. Then, the images are compressed and added to the AVI file which is stored on the hard disk. These operations are not performed in the same thread as the capturing process. If you capture more images while a compression or write operation is in progress, the new images will be discarded.

`isavi_StartAVI()`	Starts AVI recording.
`isavi_AddFrame()`	Adds a compressed image to the AVI file.
`isavi_StopAVI()`	Stops AVI recording.

With these functions, you can query additional information on the ongoing recording.

`isavi_GetAVISize()`	Returns the size of the current AVI file.
`isavi_GetnCompressedFrames()`	Returns the number of frames in the current AVI file.
`isavi_GetnLostFrames()`	Returns the number of frames that have been discarded so far.
`isavi_ResetFrameCounters()`	Resets the counters for discarded and saved frames to 0.

Events can be used to get signalled when a frame was added.

`isavi_DisableEvent()`	Disables a AVI event.

isavi_EnableEvent()	Enables a AVI-Event.
isavi_ExitEvent()	Turns off AVI event handling.
isavi_InitEvent()	Turns on AVI event handling.

Supported color formats

The supported input color formats are RGB32, RGB24, Y8 and raw Bayer. The output file will always be in RGB24 format, regardless of the input data format. You can adjust the size of the images to be stored by defining a freely selectable area of interest (AOI).

Capture speed

The possible speed of capture depends on the selected color format, the image size and the compression level of the AVI file as well as the PC performance.

Playback in external applications

AVI files you have captured using the `uc480_tools.dll` can also be played back in external applications, such as Windows Media Player. To do this, you need to install the uc480 MJPEG codec on your system:

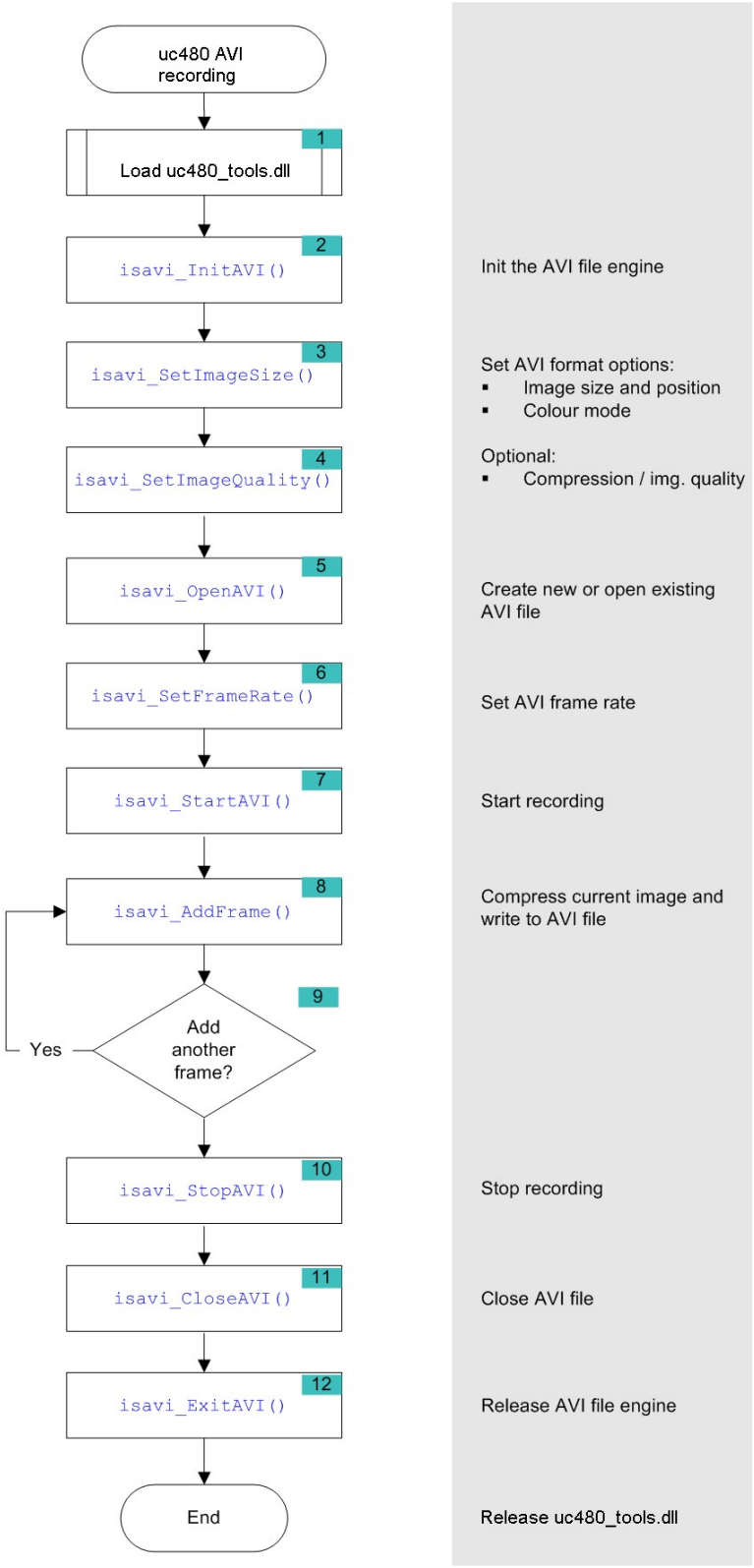
- Open the uc480 installation directory (default: `C:\Program Files\Thorlabs\DCx Cameras\Tools32 or C:\Program Files\Thorlabs\DCx Cameras\Tools64`).
- Right-click the `uc480Mjpeg.inf` (`uc480Mjpeg_64.inf)` file.
- Select "Install". The codec is installed automatically.

In player or recording software, the codec will show up as "Intermedia-X MJPEG Codec".

 © 2015 Thorlabs Scientific Imaging

Flowchart: AVI capture

Click in the figure to get help on the functions.



4.2.6 Using Inputs and Outputs

Depending on the model, DCx Cameras have one or more digital inputs and outputs designed for different purposes.

- Input/output control: Here, you will find functions for setting the DCxCamera's I/Os and for using the trigger and flash modes.

4.2.6.1 Input/Output Control

With these functions you can use the camera's digital in-/outputs for trigger and flash control.

`is_SetExternalTrigger()`	Enables the digital input for trigger operation or returns the applied signal level.
`is_IO()`	Sets the digital output for flash control or a static output level.
`is_IO()`	Sets the delay and power-on time of the flash output.
`is_SetTriggerDelay()`	Sets the trigger signal delay time.
`is_IO()`	Determines the delay and power-on times of the flash output to obtain a global shutter effect when using rolling shutter sensors.
`is_ForceTrigger()`	Simulates a trigger signal in hardware trigger mode.

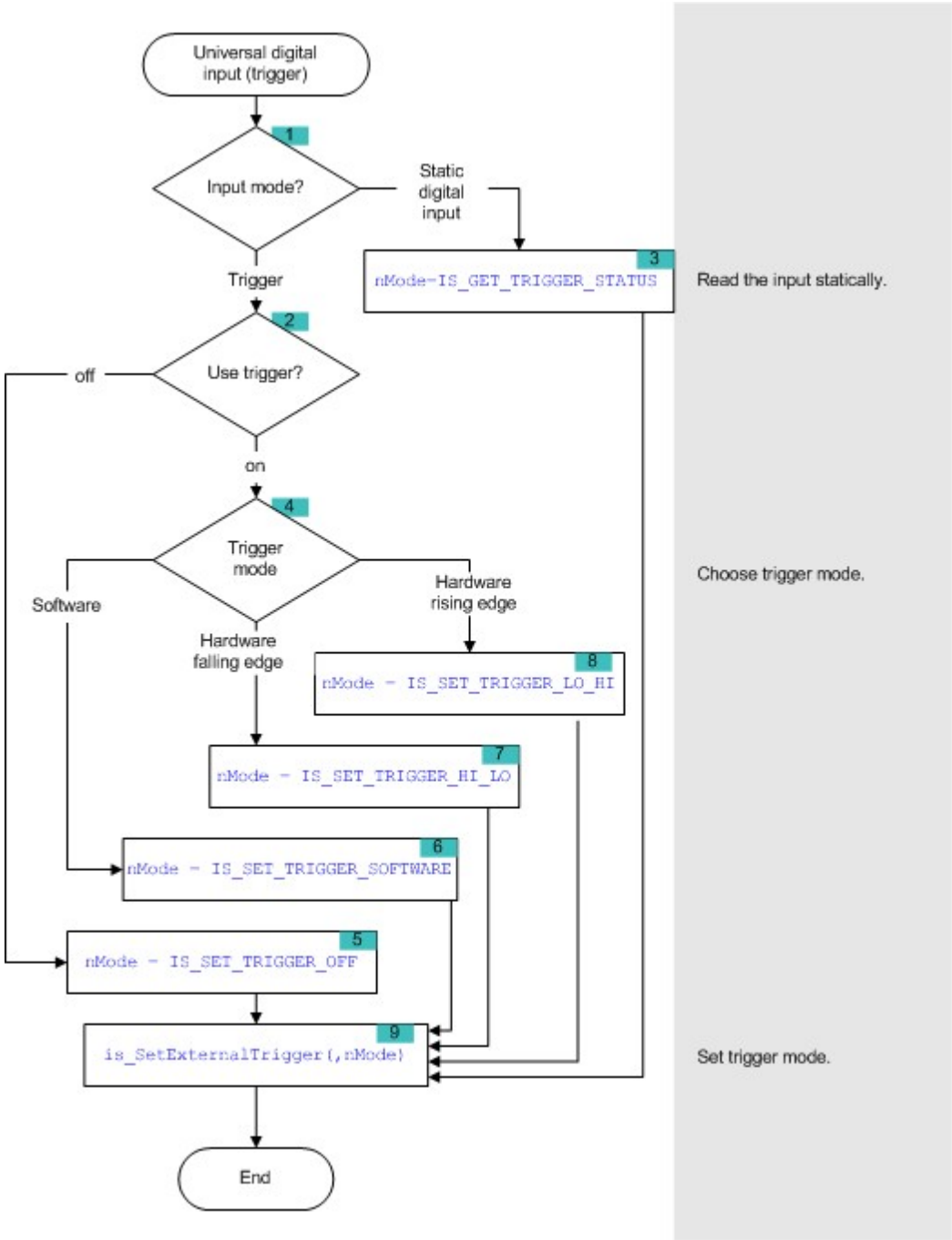
With these commands you can activate additional functions or use GPIOs on some DCx Cameras.

`is_IO()`	Sets the additional digital outputs (GPIO).
`is_IO()`	Defines each port as a digital input or output (GPIO).
`is_IO()`	Toggles the color of the status LED for DCU22x and DCC1240x cameras.

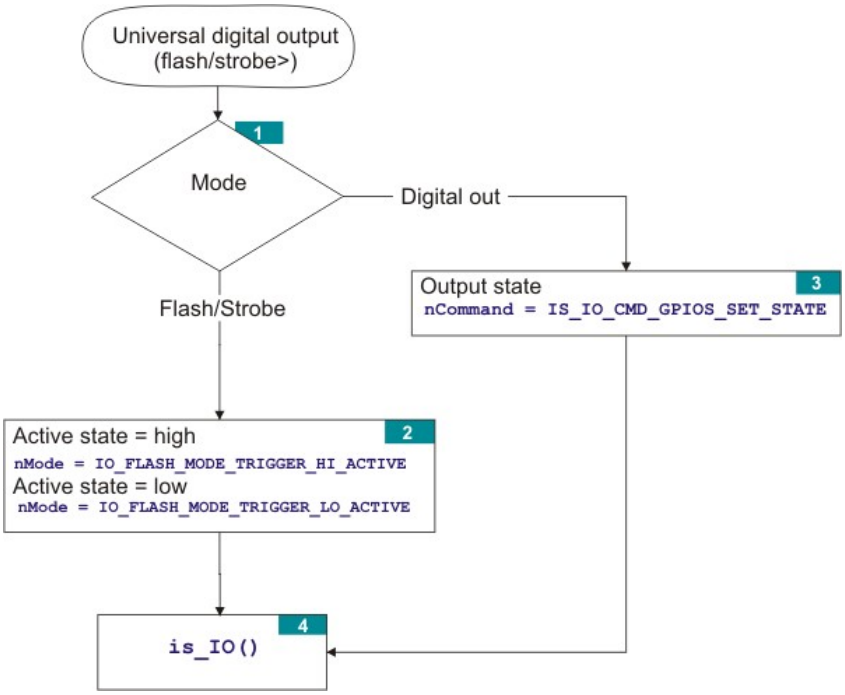
© 2015 Thorlabs Scientific Imaging

Flowchart: Digital input

Click in the figure to get help on the functions.



Flowchart: Digital output



© 2015 Thorlabs Scientific Imaging

4.3 Function Descriptions

To integrate the DCx Cameras into your own programs, you can use the functions and parameters provided by the uc480 SDK. These are described in this chapter. The descriptions are listed alphabetically by function and are structured as follows:

USB 2.0 USB 3.0	USB 2.0 USB 3.0

This table shows the availability of the function. For both Windows and Linux the table shows which DCx camera series supports the function.

Syntax

Prototype of the function from the `uc480.h` header file.

Description

Description of the function with cross-references to related functions.

Input parameters

Description of the function parameters including their value ranges.

Return value

Description and value range of the return value. If a function returns the `IS_NO_SUCCESS (-1)` value, you can get information on the error from the `is_GetError()` function.

Related functions

List with similar or related SDK functions.

Example

For some functions, C++ programming samples are have been added.

Sample programs

Some descriptions include references to uc480 SDK sample programs. When you install the uc480 software, the demo applications are copied to the `C:\Program Files\Thorlabs\DCx Cameras\Samples` directory. The associated source code can be found under `C:\Program Files \Thorlabs\DCx Cameras\Develop\Source`.

All sample programs are described in the **uc480 Samples Manual.**

4.3.1 is_AddToSequence

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_AddToSequence (HIDS hCam, char* pcImgMem, INT nID)
```

Description

`is_AddToSequence()` adds an image memory to the list of image memories used for ring buffering. The image memory must have been previously requested using `is_AllocImageMem()`. Using the `is_SetAllocatedImageMem()` function, you can set a memory that has been allocated before as image memory. Image memories that are used for ring buffering must all have been allocated with the same color depth (bits per pixel).

Input parameters

`hCam`	Camera handle
`pcMem`	Pointer to image memory
`nID`	Image memory ID

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`
- `is_InitImageQueue()`
- `is_SetImageMem()`
- `is_SetAllocatedImageMem()`

Sample programs

- uc480Sequence (C++)

© 2015 Thorlabs Scientific Imaging

4.3.2 is_AllocImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_AllocImageMem (HIDS hCam, INT width, INT height, INT bitspixel, char** ppcImgMem, INT* pid)
```

Description

`is_AllocImageMem()` allocates an image memory for an image having its dimensions defined by `width` and `height` and its color depth defined by `bitspixel`. The memory size is at least:

`size = [width * int((bitspixel + 7) / 8) + adjust] * height` (for details on `adjust`, see below)

The line increment is calculated as:

`line = width * int[(bitspixel + 7) / 8]`

`lineinc = line + adjust`

`adjust = 0`, if line can be divided by 4 without remainder

`adjust = 4 - rest(line / 4)`, if line cannot be divided by 4 without remainder

To read out the line increment, you can use the `is_GetImgMemPitch()` function.

The starting address of the memory area is returned in `ppcImgMem`.

pid returns an ID for the allocated memory. A newly allocated memory is not directly active, i.e. digitised images will not be stored immediately in this new memory. It must first be made active using `is_SetImageMem()`.

The returned pointer must be write-protected and may not be altered because it will be used for all further `ImageMem` functions. To release the memory, you can use `is_FreeImageMem()`.

Notes

- In the Direct3D or OpenGL modes, image memory allocation is not necessary.
- RGB16 and RGB15 require the same amount of memory, but can be distinguished by the `bitspixel` parameter. For information on the bit depths of different color formats please refer to the Appendix: Color and memory formats chapter.
- In case the operating system is short of physical memory, today's OS versions swap individual areas of the RAM that have not been used for some time out to the slower hard disk. This can slow down image capture if more image memory has been allocated than can be provided by the RAM at a time.

Input parameters

`hCam`	Camera handle
`width`	Image width
`height`	Image height
`bitspixel`	Image bit depth (bits per pixel).
`ppcImgMem`	Returns the pointer to the memory starting address
`pid`	Returns the ID of this memory

Return values

IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully

Related functions

- is_FreeImageMem()
- is_AddToSequence()
- is_SetImageMem()
- is_SetAllocatedImageMem()
- is_GetColorDepth()
- is_GetImgMemPitch()

© 2015 Thorlabs Scientific Imaging

4.3.3 is_AOI

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_AOI (HIDS hCam, UINT nCommand, void* pParam, UINT nSizeOfParam)
```

Description

`is_AOI()` can be used to set the size and position of an area of interest (AOI) within an image. The following AOIs can be defined:

- Image AOI – display of an image portion
- Auto Brightness AOI – reference area of interest for automatic brightness control
- Auto Whitebalance AOI – reference area of interest for automatic white balance control

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

Note

Previous AOI functions

The `is_AOI()` function comprises all the functions for setting and positioning an AOI. The following uc480 API commands are therefore obsolete:

- `is_SetAOI()`
- `is_SetImageAOI()`
- `is_SetImageSize()`
- `is_SetImagePos()`

See also Obsolete functions

AOI for automatic image control

The AOI for automatic brightness control (AES/AGC) and automatic white balance (AWB) defaults to the same size as the current image (i.e. the image AOI).

After changes to the image geometry (by resetting an image AOI, by binning or subsampling), the AOIs for automatic image control will always be reset to the image AOI value. This means that it might be necessary to set the AOIs for auto brightness/auto white balance again manually.

Fast changes of AOI position

Using the `IS_AOI_IMAGE_SET_POS_FAST` command, you can change the positions of AOIs very quickly. Executing this command takes just a few milliseconds. When using this command, a few special requirements have to be met:

- The command is currently not supported by all DCx Cameras. With the `IS_AOI_IMAGE_SET_POS_FAST_SUPPORTED` command, you can check whether your sensor supports fast position changes.
- Hot pixel correction has to be disabled (see `is_HotPixel()`).
- Image capture is not suspended for fast AOI position changes. As a result, when you call the command, a number of images might still be transferred with the old AOI position if they were in the driver buffer at that moment.

Notes

1. Changing the image size

- When changing the size of the AOI, please make sure that the selected image memory is large enough. If it isn't, allocate a new image memory (see `is_AllocImageMem()`).
- Changes to the image size affect the value ranges of the frame rate and exposure time. After executing `is_AOI()`, calling the following functions is recommended in order to keep the defined camera settings:
  - `is_SetFrameRate()`
  - `is_Exposure()`
  - If you are using the flash function: `is_IO()`

2. Step widths for AOI definition (position grid)

The available step widths for the position and size of image AOIs depend on the sensor. The values defining the position and size of an AOI have to be integer multiples of the allowed step widths.

For details on the AOI grids of the individual camera models, please see Camera and sensor data.

3. AOI in combination with high frame rates

With very small AOI and therefore high frame rate and maximum possible frame rate set, it is possible that the USB camera transfers in freerun mode only half frame rates. This is a signal for a camera-internal overload. In this case it is recommended to set the frame rate to maximum of 98 % .

Multi AOI function of the DCC1240x and DCC3240x models

The sensor of a.m. cameras supports multiple AOIs in one image capture. The AOIs are transferred together as one image. In this mode you can create 2 or 4 AOIs, which have either the same X axis or the same Y axis (see also uc480 Viewer: Multi AOI). The sensor is faster in this mode. It is possible to switch the AOI in the horizontal direction.

Sequence AOI mode for DCC1240x and DCC3240x camera models

A.m. camera models have a special AOI mode. In this mode you can define besides the normal AOI (AOI 1) up to 3 further AOI on the sensor (see uc480 Viewer: Sequence AOI). When activating the sequence mode, note that only the following combinations are possible:

1. All additional AOIs are off. AOI 1 is always active.
2. AOI 2 (+ AOI 1)
3. AOI 2 and 3 (+ AOI 1)
4. AOI 2, 3 and 4 (+ AOI 1)

It is not possible to have a combination e.g. of AOI 2 and AOI 4.

The parameters of AOI 2, 3 and 4 are defined by the `AOI_SEQUENCE_PARAMS` structure. In the version 4.60 binning, subsampling and scaler are not supported.

Input parameters

hCam	Camera handle
⊟ nCommand	
IS_AOI_IMAGE_SET_AOI	Sets the position and size of the image by using an object of the `IS_RECT` type. Sample 1 for AOI You can define the start position of the AOI in the memory by ORing `IS_AOI_IMAGE_POS_ABSOLUTE` with the X or Y position. Sample for setting the AOI position

 © 2015 Thorlabs Scientific Imaging

hCam	Camera handle
IS_AOI_IMAGE_GET_AOI	Returns the AOI in an IS_RECT object. Sample 2 for AOI
IS_AOI_IMAGE_SET_POS	Sets the AOI position by using an object of the IS_POINT_2D type. IS_AOI_IMAGE_POS_ABSOLUTE can be ORed here, as well. Sample for setting the AOI position
IS_AOI_IMAGE_GET_POS	Returns the position in an IS_POINT_2D object.
IS_AOI_IMAGE_SET_SIZE	Sets the AOI size by using an object of the IS_SIZE_2D type.
IS_AOI_IMAGE_GET_SIZE	Returns the size in an IS_SIZE_2D object.
IS_AOI_IMAGE_GET_POS_MIN	Returns the minimum possible position in an IS_POINT_2D object.
IS_AOI_IMAGE_GET_SIZE_MIN	Returns the smallest possible size in an IS_SIZE_2D object.
IS_AOI_IMAGE_GET_POS_MAX	Returns the maximum possible position in an IS_POINT_2D object.
IS_AOI_IMAGE_GET_SIZE_MAX	Returns the largest possible size in an IS_SIZE_2D object.
IS_AOI_IMAGE_GET_POS_INC	Returns the increment for the position in an IS_POINT_2D object.
IS_AOI_IMAGE_GET_SIZE_INC	Returns the increment for the size in an IS_SIZE_2D object.
IS_AOI_IMAGE_GET_POS_X_ABS	Returns an UINT object indicating whether IS_AOI_IMAGE_POS_ABSOLUTE is set for the X position. Sample 3 for AOI
IS_AOI_IMAGE_GET_POS_Y_ABS	Returns an UINT object indicating whether IS_AOI_IMAGE_POS_ABSOLUTE is set for the Y position.
IS_AOI_IMAGE_GET_ORIGINAL_AOI	Returns the AOI in an IS_RECT object without binning, subsampling or scaling.
IS_AOI_IMAGE_SET_POS_FAST	Allows changing the AOI position very quickly by using an IS_POINT_2D object. Hot pixel correction has to be disabled (see information above).
IS_AOI_IMAGE_SET_POS_FAST_SUPPOR	Returns an UINT object indicating whether fast AOI position changes are supported. The passed variable returns 0 if the function is not supported by the sensor.
IS_AOI_AUTO_BRIGHTNESS_SET_AOI	Sets the AOI for automatic brightness control (similar to IS_AOI_IMAGE_SET_AOI).
IS_AOI_AUTO_BRIGHTNESS_GET_AOI	Returns the AOI for automatic brightness control (similar to IS_AOI_IMAGE_GET_AOI).
IS_AOI_AUTO_WHITEBALANCE_SET_AOI	Sets the AOI for automatic white balance (similar to IS_AOI_IMAGE_SET_AOI).
IS_AOI_AUTO_WHITEBALANCE_GET_AOI	Returns the AOI for automatic white balance (similar to IS_AOI_IMAGE_GET_AOI).
IS_AOI_MULTI_GET_SUPPORTED_MODES	Returns the supported multi AOI modes in an UINT object.
IS_AOI_MULTI_SET_AOI	Sets the multi AOI mode. The mode you want to use has to be ORed with IS_AOI_MULTI_SET_AOI. The axes are passed in an UINT array: • Array[0] - Array[3] = X1…X4 • Array[4] - Array[8] = Y1…Y4 Sample 1 for multi AOI
IS_AOI_MULTI_GET_AOI	Returns the set multi AOI mode. The mode that is used has to be ORed with IS_AOI_MULTI_SET_AOI.

hCam	Camera handle
	Sample 2 fo multi AOI
IS_AOI_MULTI_MODE_X_Y_AXES	Multi AOI mode of the camera models DCC1240x/DCC3240x with up to AOIs (up to 4 x and y axes). The axes are passed by a UINT array: • array[0] - array[3] = X1…X4 • array[4] - array[8] = Y1…Y4 Attention: This parameter has been renamed in version 4.20. In formerly versions this parameter was named IS_AOI_MULTI_MODE_AXES.
IS_AOI_MULTI_DISABLE_AOI	Disables Multi AOI. The mode that is used has to be ORed with IS_AOI_MULTI_SET_AOI. Sample 3 for multi AOI
IS_AOI_SEQUENCE_GET_SUPPORTED	Returns a bitmask with the supported AOIs (only DCC1240x/DCC3240x camera models) Sample 1 for AOI sequence mode
IS_AOI_SEQUENCE_SET_PARAMS	Sets the parameters of AOI 2, 3 or 4 (only DCC1240x/DCC3240x camera models) Sample 2 for sequence AOI mode
IS_AOI_SEQUENCE_GET_PARAMS	Returns the parameters of AOI 2, 3 or 4 (only DCC1240x/DCC3240x camera models) Sample 2 for sequence AOI mode
IS_AOI_SEQUENCE_SET_ENABLE	Set a bitmask defining which AOIs should be active (only DCC1240x/DCC3240x camera models). **Note:** IS_AOI_SEQUENCE_SET_PARAMS must be called after IS_AOI_SEQUENCE_SET_ENABLE, with enabling the sequence AOI mode all AOIs are set to the same value and therefore the parameters are lost. Sample 3 for sequence AOI mode
IS_AOI_SEQUENCE_GET_ENABLE	Returns the bitmask (only DCC1240x/DCC3240x camera models) Sample 3 for sequence AOI mode
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the IS_RECT structure

INT	s32X	X position of the AOI
INT	s32Y	Y position of the AOI
INT	s32Width	Width of the AOI
INT	s32Height	Height of the AOI

Contents of the IS_POINT_2D structure

INT	s32X	X position of the AOI
INT	s32Y	Y position of the AOI

 © 2015 Thorlabs Scientific Imaging

Contents of the IS_SIZE_2D structure

INT	s32Width	Width of the AOI
INT	s32Height	Height of the AOI

Content of the AOI_SEQUENCE_PARAMS structure

INT	s32AOIIndex	Index of the AOI
INT	s32NumberOfCycleRepetitions	Number of readout cycles
INT	s32X	X position of the AOI
INT	x32Y	Y position of the AOI
Double	dblExposure	Exposure
INT	s32Gain	Gain
INT	s32BinningMode	Binning mode (not supported in version 4.60)
INT	s32SubsamplingMode	Subsampling mode (not supported in version 4.60)
INT	s32DetachImageParameters	• 0 = every change of the exposure time and the master gain is copied from AOI 1 to the additional AOIs (default). As a change of AOI 1 also reset the exposure time, this change is also transferred to AOI 2, 3 and 4. • 1 = a change of exposure time, gain or position of AOI 1 does not affect the parameters of AOI 2, 3 and 4.
Double	dblScalerFactor	Scaling factor (not supported in version 4.60)
BYTE	byReserved[64]	Reserved

Return values

IS_BAD_STRUCTURE_SIZE	An internal structure has an incorrect size.
IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

 © 2015 Thorlabs Scientific Imaging

IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- is_ImageFormat()
- is_SetBinning()
- is_SetSubSampling()

Sample 1 for AOI

```
// Sets the position and size of the image by using an object of the IS_RECT type.
IS_RECT rectAOI;

rectAOI.s32X      = 100;
rectAOI.s32Y      = 100;
rectAOI.s32Width  = 200;
rectAOI.s32Height = 100;

INT nRet = is_AOI( hCam, IS_AOI_IMAGE_SET_AOI, (void*)&rectAOI, sizeof(rectAOI));
```

Sample 2 for AOI

```
// Returns the AOI position and size by using an object of the IS_RECT type.
IS_RECT rectAOI;

INT nRet = is_AOI(hCam, IS_AOI_IMAGE_GET_AOI, (void*)&rectAOI, sizeof(rectAOI));
if (nRet == IS_SUCCESS)
{
    INT x      = rectAOI.s32X;
    INT y      = rectAOI.s32Y;
    INT width  = rectAOI.s32Width;
    INT height = rectAOI.s32Height;
}
```

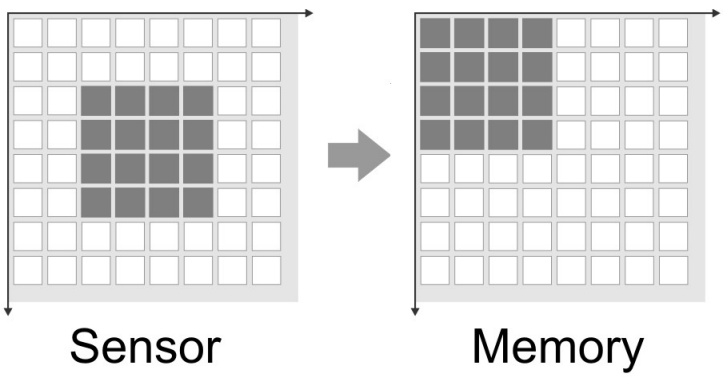
Sample 3 for AOI

```
// Returns an UINT object indicating whether IS_AOI_IMAGE_POS_ABSOLUTE is set for the X position.
UINT nAbsPos = 0;

INT nRet = is_AOI(hCam, IS_AOI_IMAGE_GET_POS_X_ABS, (void*)&nAbsPos , sizeof(nAbsPos ));
if (nRet == IS_SUCCESS)
{
    if (nAbsPos == IS_AOI_IMAGE_POS_ABSOLUTE)
    {
        // set
    }
    else if (nAbsPos == 0)
    {
        // not set
    }
}
```

Examples for setting absolute AOI positions in memory

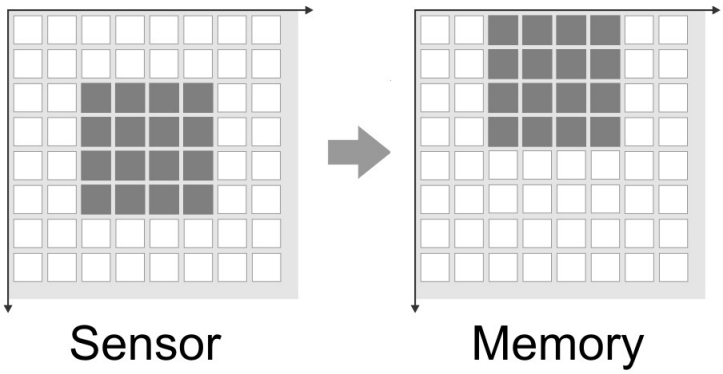
```
x = 100
y = 100
```



AOI without absolute memory positioning

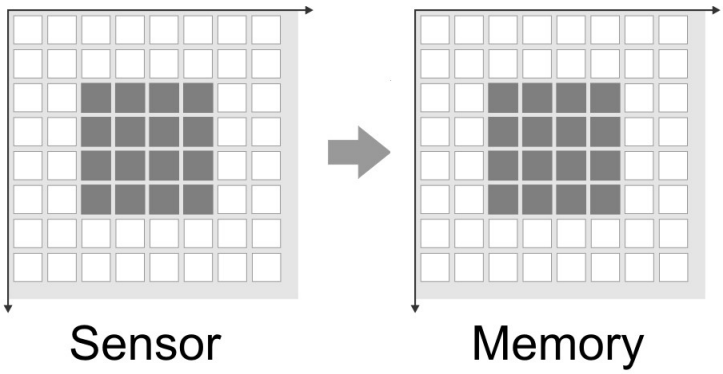
```
x = 100 | IS_AOI_IMAGE_POS_ABSOLUTE
y = 100
```


© 2015 Thorlabs Scientific Imaging



AOI with absolute memory positioning on x-axis

```
x = 100 | IS_AOI_IMAGE_POS_ABSOLUTE
y = 100 | IS_AOI_IMAGE_POS_ABSOLUTE
```



AOI with absolute memory positioning on x- and y-axis

Sample 1 for multi AOI

```
// Set Multi AOI. The axes are passed in an UINT array of length 8.
UINT nAxes[8];

nAxes[0] = 100;  // Set X1
nAxes[1] = 120;  // Set X2
...

INT nRet = is_AOI(hCam, IS_AOI_MULTI_SET_AOI | IS_AOI_MULTI_MODE_X_Y_AXES, (void*)nAxes, sizeof(nAxes))
```

Sample 2 for multi AOI

```
// Read Multi AOI
UINT nAxes[8];

INT nRet = is_AOI(hCam, IS_AOI_MULTI_GET_AOI | IS_AOI_MULTI_MODE_X_Y_AXES, (void*)nAxes, sizeof(nAxes))
```

Sample 3 for multi AOI

```
// Disable Multi AOI
UINT nAxes[8];

INT nRet = is_AOI(hCam, IS_AOI_MULTI_DISABLE_AOI | IS_AOI_MULTI_MODE_X_Y_AXES, NULL, NULL);
```

Sample 1 for sequence AOI mode

```
INT nSequenceAOI = 0;
if (is_AOI(m_hCam, IS_AOI_SEQUENCE_GET_SUPPORTED,
           (void*)&nSequenceAOI, sizeof(nSequenceAOI)) == IS_SUCCESS)
{
  // Sequence AOI 2 is supported
  if ((nSequenceAOI & IS_AOI_SEQUENCE_INDEX_AOI_2) != 0);
}
```

Sample 2 for sequence AOI mode

```
AOI_SEQUENCE_PARAMS Param;

// Set parameters of AOI 2
Param.s32AOIIndex = IS_AOI_SEQUENCE_INDEX_AOI_2;
Param.s32NumberOfCycleRepetitions = 1;
Param.s32X = 100;
Param.s32Y = 200;
...

INT nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_SET_PARAMS, (void*)&Param, sizeof(Param));

// Get parameters of AOI 2
Param.s32AOIIndex = IS_AOI_SEQUENCE_INDEX_AOI_2;

nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_GET_PARAMS, (void*)&Param, sizeof(Param));
```

Sample 3 for sequence AOI mode

```
INT nMask = 0;

// Enable AOI 1, Disable AOI 2, 3 and 4
nMask = IS_AOI_SEQUENCE_INDEX_AOI_1;

INT nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_SET_ENABLE, (void*)&nMask, sizeof(nMask));

// Enable AOI 1 and 2
nMask = IS_AOI_SEQUENCE_INDEX_AOI_1 |
        IS_AOI_SEQUENCE_INDEX_AOI_2;

nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_SET_ENABLE, (void*)&nMask, sizeof(nMask));

// Enable AOI 1, 2 and 3
nMask = IS_AOI_SEQUENCE_INDEX_AOI_1 |
        IS_AOI_SEQUENCE_INDEX_AOI_2 |
        IS_AOI_SEQUENCE_INDEX_AOI_3;

nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_SET_ENABLE, (void*)&nMask, sizeof(nMask));

// Enable AOI 1, 2, 3 and 4
nMask = IS_AOI_SEQUENCE_INDEX_AOI_1 |
        IS_AOI_SEQUENCE_INDEX_AOI_2 |
        IS_AOI_SEQUENCE_INDEX_AOI_3 |
        IS_AOI_SEQUENCE_INDEX_AOI_4;

nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_SET_ENABLE, (void*)&nMask, sizeof(nMask));

// Get current AOI mask
INT nRet = is_AOI(m_hCam, IS_AOI_SEQUENCE_GET_ENABLE, (void*)&nMask, sizeof(nMask));
```


© 2015 Thorlabs Scientific Imaging

4.3.4 is_AutoParameter

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_AutoParameter(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

This functions enables/diisables the auto white balance. With this function, you can require all supported types for white balance. In addition to the older white balance with the Gray-World algorithm, there is also a color temperature control according to Kelvin. In addition to the function the supported color spaces are queried and set.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

In a later version the `is_AutoParameter()` function will replace the `is_SetAutoParameter()` function. In version 4.60, it is only partly replaced.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_AWB_CMD_GET_SUPPORTED_TYPES`	Returns the supported types for auto white balance (Example 1) • `IS_AWB_GREYWORLD`: 0x0001 • `IS_AWB_COLOR_TEMPERATURE`: 0x0002
`IS_AWB_CMD_GET_TYPE`	Returns the current set type of the auto white balance (Example 2) • `IS_AWB_GREYWORLD`: 0x0001 • `IS_AWB_COLOR_TEMPERATURE`: 0x0002
`IS_AWB_CMD_SET_TYPE`	Sets the type of the auto white balance (Example 2) • `IS_AWB_GREYWORLD`: 0x0001 • `IS_AWB_COLOR_TEMPERATURE`: 0x0002
`IS_AWB_CMD_GET_ENABLE`	Returns if the auto white balance is enabled (Example 3) • `IS_AUTOPARAMETER_DISABLE`: 0 • `IS_AUTOPARAMETER_ENABLE`: 1 • `IS_AUTOPARAMETER_ENABLE_RUNONCE`: 2
`IS_AWB_CMD_SET_ENABLE`	Enables/Disables the auto white balance (Example 3) • `IS_AUTOPARAMETER_DISABLE`: 0 • `IS_AUTOPARAMETER_ENABLE`: 1 • `IS_AUTOPARAMETER_ENABLE_RUNONCE`: 2
`IS_AWB_CMD_GET_SUPPORTED_RGB_COLOR_M`	Returns the supported color spaces for the auto white

hCam	Camera handle
ODELS	balance (Example 4) • RGB_COLOR_MODEL_SRGB_D50: 0x0001 • RGB_COLOR_MODEL_SRGB_D65: 0x0002 • RGB_COLOR_MODEL_CIE_RGB_E: 0x0004 • RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008 • RGB_COLOR_MODEL_ADOBE_RGB_D65: 0x0010
IS_AWB_CMD_GET_RGB_COLOR_MODEL	Returns the current color space for the auto white balance (Example 5) • RGB_COLOR_MODEL_SRGB_D50: 0x0001 • RGB_COLOR_MODEL_SRGB_D65: 0x0002 • RGB_COLOR_MODEL_CIE_RGB_E: 0x0004 • RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008 • RGB_COLOR_MODEL_ADOBE_RGB_D65: 0x0010
IS_AWB_CMD_SET_RGB_COLOR_MODEL	Sets the color space for the auto white balance (Example 5) • RGB_COLOR_MODEL_SRGB_D50: 0x0001 • RGB_COLOR_MODEL_SRGB_D65: 0x0002 • RGB_COLOR_MODEL_CIE_RGB_E: 0x0004 • RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008 • RGB_COLOR_MODEL_ADOBE_RGB_D65: 0x0010
pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetAutoParameter()

Example 1

```
UINT nSupportedTypes = 0;
INT nRet = is_AutoParameter(m_hCam,
                            IS_AWB_CMD_GET_SUPPORTED_TYPES,
                            (void*)&nSupportedTypes,
                            sizeof(nSupportedTypes)
                           );

if (nRet == IS_SUCCESS)
{
    if ((nSupportedTypes & IS_AWB_COLOR_TEMPERATURE) != 0)
    {
        // AWB type "Color Temperature" is supported
    }
```

 © 2015 Thorlabs Scientific Imaging

```
    if ((nSupportedTypes & IS_AWB_GREYWORLD) != 0)
    {
        // AWB type "Greyworld" is supported
    }
}
```

Example 2

```
UINT nType = 0;

// Read current type
INT nRet = is_AutoParameter(m_hCam, IS_AWB_CMD_GET_TYPE, (void*)&nType, sizeof(nType));

// Write new type
nType = IS_AWB_GREYWORLD;
nRet = is_AutoParameter(m_hCam, IS_AWB_CMD_SET_TYPE, (void*)&nType, sizeof(nType));
```

Example 3

```
// Is AWB enabled?
UINT nEnable;
INT nRet = is_AutoParameter(m_hCam, IS_AWB_CMD_GET_ENABLE, (void*)&nEnable, sizeof(nEnable));

// Enable AWB (once)
nEnable = IS_AUTOPARAMETER_ENABLE_RUNONCE;
nRet = is_AutoParameter(m_hCam, IS_AWB_CMD_SET_ENABLE, (void*)&nEnable, sizeof(nEnable));
```

Example 4

```
UINT nSupportedRGBColorModels = 0;
nRet = is_AutoParameter(m_hCam,
                        IS_AWB_CMD_GET_SUPPORTED_RGB_COLOR_MODELS,
                        (void*)&nSupportedRGBColorModels,
                        sizeof(nSupportedRGBColorModels)
                       );

if (nRet == IS_SUCCESS)
{
    if ((nSupportedRGBColorModels & RGB_COLOR_MODEL_SRGB_D50) != 0)
    {
        // Color model SRGB D50 is supported. See uc480.h for color model defines
    }
}
```

Example 5

```
UINT nRGBColorModel = 0;
INT nRet = is_AutoParameter(m_hCam,
                            IS_AWB_CMD_GET_RGB_COLOR_MODEL,
                            (void*)&nRGBColorModel,
                            sizeof(nRGBColorModel)
                           );

nRGBColorModel = RGB_COLOR_MODEL_CIE_RGB_E;

nRet = is_AutoParameter(m_hCam,
                        IS_AWB_CMD_SET_RGB_COLOR_MODEL,
                        (void*)&nRGBColorModel,
                        sizeof(nRGBColorModel)
                       );
```

4.3.5 is_Blacklevel

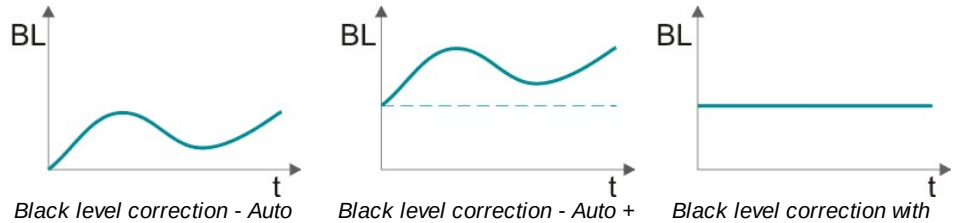
USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_Blacklevel(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Beschreibung

`is_Blacklevel()` controls the black level correction of the camera which might improve the image quality under certain circumstances. By default, the sensor adjusts the black level value for each pixel automatically. If the environment is very bright, it can be necessary to adjust the black level manually.



Black level correction - Auto *Black level correction - Auto + offset* *Black level correction with offset only*

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_BLACKLEVEL_CMD_GET_CAPS`	Returns the black level feature of the camera (Example 1): • `IS_BLACKLEVEL_CAP_SET_AUTO_BLACKLEVEL`: The state of the automatic black level can be changed. The flag does not indicate whether the camera is running with auto black level by default or not. For this purpose, use `IS_BLACKLEVEL_CMD_GET_MODE_DEFAULT`. • `IS_BLACKLEVEL_CAP_SET_OFFSET`: The offset can be changed. The flag does not indicate whether the camera has set an offset by default or not. For this purpose, use `IS_BLACKLEVEL_CMD_GET_OFFSET_DEFAULT`.
`IS_BLACKLEVEL_CMD_GET_MODE_DEFAULT`	Returns the default black level mode (Example2)
`IS_BLACKLEVEL_CMD_GET_MODE`	Returns the current black level mode (Example2)
`IS_BLACKLEVEL_CMD_SET_MODE`	Sets the black level mode (Example 2) • `IS_AUTO_BLACKLEVEL_OFF`: The automatic black level mode is switched off.

 © 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	• IS_AUTO_BLACKLEVEL_ON: The automatic black level mode is switched on.
IS_BLACKLEVEL_CMD_GET_OFFSET_DEFAULT	Returns the default offset (Example 3)
IS_BLACKLEVEL_CMD_GET_OFFSET_RANGE	Returns the range of the offset (Example 3)
IS_BLACKLEVEL_CMD_GET_OFFSET	Returns the current offset (Example 3)
IS_BLACKLEVEL_CMD_SET_OFFSET	Sets the offset (Example 3)
pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Example 1

```
INT nBlacklevelCaps;

nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_CAPS,
                     (void*)&nBlacklevelCaps, sizeof(nBlacklevelCaps) );
if (nRet == IS_SUCCESS) {

  // The user can changed the state of the auto blacklevel
  BOOL bSetAutoBlacklevel = (nBlacklevelCaps & IS_BLACKLEVEL_CAP_SET_AUTO_BLACKLEVEL) != 0;

  // The user can change the offset
  BOOL bSetBlacklevelOffset = (nBlacklevelCaps & IS_BLACKLEVEL_CAP_SET_OFFSET) != 0;
}
```

Example 2

```
INT nMode = IS_AUTO_BLACKLEVEL_OFF;

// Get default blacklevel mode
INT nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_MODE_DEFAULT, (void*)&nMode, sizeof(nMode));

// Get current blacklevel mode
nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_MODE, (void*)&nMode, sizeof(nMode));

// Set new mode (enable auto blacklevel)
nMode = IS_AUTO_BLACKLEVEL_ON;
nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_SET_MODE, (void*)&nMode , sizeof(nMode ));
```

Example 3

```
INT nOffset = 0;

// Get default blacklevel offset
INT nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_OFFSET_DEFAULT,
                         (void*)&nOffset, sizeof(nOffset));

// Get offset range
IS_RANGE_S32 nRange;
nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_OFFSET_RANGE, (void*)&nRange, sizeof(nRange));
INT nOffsetMin = nRange.s32Min;
INT nOffsetMax = nRange.s32Max;
```

```
INT nOffsetInc = nRange.s32Inc;

// Get current blacklevel offset
nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_GET_OFFSET, (void*)&nOffset, sizeof(nOffset));

// Set new offset
nOffset = 100;
nRet = is_Blacklevel(hCam, IS_BLACKLEVEL_CMD_SET_OFFSET, (void*)&nOffset, sizeof(nOffset));
```


© 2015 Thorlabs Scientific Imaging

4.3.6 is_CameraStatus

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
ULONG is_CameraStatus (HIDS hCam, INT nInfo, ULONG ulValue)
```

Description

Using `is_CameraStatus()`, you can query and partly set various status information and settings.

Input parameters

`hCam`	Camera handle
⊟ `nInfo`	
`IS_FIFO_OVR_CNT`	Number of FIFO overruns. Is increased if image data gets lost because the USB bus is congested.
`IS_SEQUENCE_CNT`	Returns the sequence count. For `is_CaptureVideo()`, this parameter is set to 0. Each time the sequence buffer (image counter) changes, the counter is increased by 1.
`IS_SEQUENCE_SIZE`	Returns the number of sequence buffers.
`IS_EXT_TRIGGER_EVENT_CNT`	Returns the camera's internal count of external trigger events.
`IS_TRIGGER_MISSED`	Returns the number of unprocessed trigger signals. Is reset to 0 after each call.
`IS_LAST_CAPTURE_ERROR`	Returns the last image capture error. For a list of all possible error events, see `is_CaptureStatus()`.
`IS_PARAMETER_SET_1`	Indicates whether parameter set 1 including camera settings is present on the camera (read-only). See also `is_ParameterSet()`. Return values: `TRUE` Parameter set 1 present `FALSE` Parameter set 1 not present
`IS_PARAMETER_SET_2`	Indicates whether parameter set 2 including camera settings is present on the camera (read-only). See also `is_ParameterSet()`. Return values: `TRUE` Parameter set 2 present `FALSE` Parameter set 2 not present
`IS_STANDBY`	Sets the camera to standby mode. Return values: `TRUE` Camera changes to standby mode `FALSE` The camera changes to freerun mode
`IS_STANDBY_SUPPORTED`	Queries whether the camera supports standby mode (read-only). Return values: `TRUE` The camera supports standby mode `FALSE` The camera does not support standby mode
⊟ `ulValue`	
`IS_GET_STATUS`	Returns the information specified by `nInfo`.

Return values

Only if `ulValue = IS_GET_STATUS`	Returns the information specified by `nInfo`
When used with `IS_LAST_CAPTURE_ERROR`	Returns the last image capture error. For a list of all possible error events, see `is_GetCaptureErrorInfo()`.
`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_EXPOSURE_TIME`	This setting is not available for the currently set exposure time.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_OUT_OF_MEMORY`	No memory could be allocated.
`IS_SUCCESS`	Function executed successfully
`IS_TIMED_OUT`	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- `is_GetCameraInfo()`
- `is_GetError()`
- `is_SetErrorReport()`
- `is_SetTriggerCounter()`

© 2015 Thorlabs Scientific Imaging

4.3.7 is_CaptureStatus

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_CaptureStatus (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

The function returns information on errors that occurred during an image capture. All errors are listed that occurred since the last reset of the function.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

The following functions are obsolete by the `is_CaptureStatus()` function:

- `is_GetCaptureErrorInfo()`
- `is_ResetCaptureErrorInfo()`

See also Obsolete functions.

Input parameters

hCam	Camera handle
nCommand	
`IS_CAPTURE_STATUS_INFO_CMD_GET`	Returns the CaptureStatus information (Example 1)
`IS_CAPTURE_STATUS_INFO_CMD_RESET`	Resets the CaptureStatus infomation (Example 2)
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Content of the uc480_CAPTURE_STATUS enumeration

`IS_CAP_STATUS_API_NO_DEST_MEM`	There is no destination memory for copying the finished image. Possible cause/remedy Not enough destination memory allocated or all destination buffers locked by the application. • Release locked destination memory • Allocate more destination memory • Reduce the frame rate so that there is more time to process the filled destination memory
`IS_CAP_STATUS_API_CONVERSION_FAILED`	The current image could not be processed correctly.

	⊟ Possible cause Internal error during internal processing of the image
IS_CAP_STATUS_API_IMAGE_LOCKED	The destination buffers are locked and could not be written to. ⊟ Possible cause/remedy All destination buffers locked by the application • Release locked destination memory • Allocate more destination memory • Reduce the frame rate so that there is more time to process the filled destination memory
IS_CAP_STATUS_DRV_OUT_OF_BUFFERS	No free internal image memory is available to the driver. The image was discarded. ⊟ Possible cause/remedy The computer takes too long to process the images in the uc480 API (e.g. color conversion) • Reduce the frame rate so that there is more time to process the filled image memory of the driver • Disable resource-intensive API image pre-processing functions (e.g. edge enhancement, color correction, choose smaller filter mask for software color conversion)
IS_CAP_STATUS_DRV_DEVICE_NOT_READY	The camera is no longer available. It is not possible to access images that have already been transferred. ⊟ Possible cause The camera has been disconnected or closed.
IS_CAP_STATUS_USB_TRANSFER_FAILED	The image was not transferred over the USB bus. ⊟ Possible cause/remedy Not enough free bandwidth on the USB bus for transferring the image • Reduce the pixel clock frequency • Operate fewer cameras simultaneously on a USB bus • Check the quality of the USB cabling and components
IS_CAP_STATUS_DEV_TIMEOUT	The maximum allowable time for image capturing in the camera was exceeded. ⊟ Possible cause/remedy The selected timeout value is too low for image capture • Reduce the exposure time • Increase the timeout

Contents of the uc480_CAPTURE_STATUS_INFO structure

DWORD	dwCapStatusCnt_Total	Returns the total number of errors occurred since the last reset.
BYTE	reserved[60]	Reserved for an internal function
DWORD	adwCapStatusCnt_Detail[256]	This array returns the current count for each possible error. The possible errors are listed above.

Return values

© 2015 Thorlabs Scientific Imaging

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_GetError()
- is_CameraStatus()
- is_SetErrorReport()

Example 1

```
uc480_CAPTURE_STATUS_INFO CaptureStatusInfo;
INT nRet = is_CaptureStatus(m_hCam,
                            IS_CAPTURE_STATUS_INFO_CMD_GET,
                            (void*)&CaptureStatusInfo,
                            sizeof(CaptureStatusInfo));

if (nRet == IS_SUCCESS)
{
    UINT nConversionFailed = CaptureStatusInfo.adwCapStatusCnt_Detail[IS_CAP_STATUS_API_CONVERSION_FAIL
    UINT nTotalInfos = CaptureStatusInfo.dwCapStatusCnt_Total;
}
```

Example 2

```
INT nRet = is_CaptureStatus(m_hCam,
                            IS_CAPTURE_STATUS_INFO_CMD_RESET,
                            NULL,
                            0);
```

4.3.8 is_CaptureVideo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_CaptureVideo (HIDS hCam, INT Wait)
```

Description

`is_CaptureVideo()` activates the camera's live video mode (free run mode). The driver transfers the images to an allocated image memory or, if Direct3D/OpenGL is used, to the graphics card. The image data (DIB mode) is stored in the memory created using `is_AllocImageMem()` and designated as active image memory using `is_SetImageMem()`. Using `is_GetImageMem()`, you can query the memory address.

If ring buffering is used, the image capturing function cycles through all image memories used for storing the images of a capture sequence in an endless loop. Sequence memories locked by `is_LockSeqBuf()` will be skipped. If the last available sequence memory has been filled, the sequence event or message will be triggered. Capturing always starts with the first element of the sequence.

For further information on the image capture modes, see the How to proceed: Image capture section.

Input parameters

`hCam`	Camera handle
⊟ `Wait`	
`IS_DONT_WAIT`	Timeout value for image capture (see also the How to proceed: Timeout values for image capture section)
`IS_WAIT`	
Time `t`	
`IS_GET_LIVE`	Returns if live capture is enabled.

Return values

When used with `IS_GET_LIVE`	`TRUE` if live capture is enabled
`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_CAPTURE_RUNNING`	A capturing operation is in progress and must be terminated before you can start another one.
`IS_INVALID_BUFFER_SIZE`	The image memory has an inappropriate size to store the image in the desired format.
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_EXPOSURE_TIME`	This setting is not available for the currently set exposure time.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle

© 2015 Thorlabs Scientific Imaging

IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the is_SetImageMem() function or create a sequence using the is_AddToSequence() function.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- is_FreezeVideo()
- is_StopLiveVideo()
- is_SetExternalTrigger()
- is_ForceTrigger()
- is_SetTimeout()
- is_CaptureStatus()

Sample programs

- SimpleLive (C++)

4.3.9 is_ClearSequence

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ClearSequence (HIDS hCam)
```

Description

`is_ClearSequence()` removes all image memories from the sequence list that were added using `is_AddToSequence()`. After a call of `is_ClearSequence()`, there is no more active image memory. To make an image memory the active memory, call `is_SetImageMem()`.

Input parameters

`hCam`	Camera handle

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SEQUENCE_LIST_EMPTY`	The sequence list is empty and cannot be deleted.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AddToSequence()`
- `is_FreeImageMem()`
- `is_SetImageMem()`

© 2015 Thorlabs Scientific Imaging

4.3.10 is_ColorTemperature

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ColorTemperature (HIDS hCam, UINT nCommand,
                         void *pParam, UINT nSizeOfParam)
```

Description

Using `is_ColorTemperature()` you can fix a setting (in kelvins) for the color temperature of an image when you are using a color camera. The function will use the sensor's hardware gain controls for the setting, as far as possible. In addition, you can choose between different color spaces. A specific color temperature will result in slightly differing RGB values, depending on the selected color space.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

The color temperature is the temperature to which a black body radiator has to be heated to glow and give off light in the corresponding color. Warm light (reddish) has a lower color temperature than cold light (bluish). The following table lists a few example values:

Light source	Color temperature
Light bulb (100 W)	2800
Halogen lamp	3200
Fluorescent lamp (cold white)	4000
Morning and evening sunlight	5000
Noon sunlight	5500-5800
Flash strobe	6000
Overcast daylight	6500-7500
Fog	8000

Note

The `is_ColorTemperature()` function cannot be used simultaneously with the automatic white balance function in `is_SetAutoParameter()`/`is_AutoParameter()`.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
Setting the color space	
`COLOR_TEMPERATURE_CMD_GET_SUPPORTED_RGB_COLOR_MODELS`	Returns the supported color spaces. ⊟ More details `pParam`: Pointer to a bit mask of type `UINT` The bit mask returns the supported modes,

hCam	Camera handle
	linked by logical ORs (see Color spaces table). nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_SET_RGB_COLOR_MODEL	Sets a color space. More details pParam: Pointer to variable of type UINT that passes the value to be set. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_RGB_COLOR_MODEL	Returns the set color space. More details pParam: Pointer to variable of type UINT returning the current value. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_RGB_COLOR_MODEL_DEFAULT	Returns the default color space. More details pParam: Pointer to variable of type UINT returning the default value. nSizeOfParam: 4
Setting the color temperature	
COLOR_TEMPERATURE_CMD_SET_TEMPERATURE	Sets a color temperature. More details pParam: Pointer to variable of type UINT that passes the value to be set. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_TEMPERATURE	Returns the set color temperature. More details pParam: Pointer to variable of type UINT returning the current value. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_MIN	Returns the minimum value for the color temperature. More details pParam: Pointer to variable of type UINT returning the minimum value. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_MAX	Returns the maximum value for the color temperature. More details pParam: Pointer to variable of type UINT returning the maximum value. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_INC	Returns the increment for setting the color temperature. More details pParam: Pointer to variable of type UINT

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	returning the increment. nSizeOfParam: 4
COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_DEFAULT	Returns the default value for the color temperature. More details pParam: Pointer to variable of type UINT returning the default value. nSizeOfParam: 4
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Color Spaces

RGB_COLOR_MODEL_SRGB_D50	sRGB (standard RGB) color space with a white point of 5000 kelvins (warm light)
RGB_COLOR_MODEL_SRGB_D65	sRGB (standard RGB) color space with a white point of 6500 kelvins (mid daylight)
RGB_COLOR_MODEL_CIE_RGB_E	CIE-RGB color space with standard illumination E
RGB_COLOR_MODEL_ECI_RGB_D50	ECI-RGB color space with a white point of 5000 kelvins (warm light)
RGB_COLOR_MODEL_ADOBE_RGB_D65	Adobe RGB color space with a white point of 6500 kelvins (mid daylight). The Adobe RGB color space is larger than the sRGB color space, but not all devices can render it.

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetHardwareGain()
- is_AutoParameter()
- is_SetAutoParameter()

© 2015 Thorlabs Scientific Imaging

4.3.11 is_Configuration

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_Configuration (UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

Use `is_Configuration()` to set various system-wide options:

- **Windows only: Processor operating states (idle states/C-states)**
  Modern processors have various operating states, so-called C-states, that are characterized by different power requirements. When the operating system selects an operating state with low power consumption (unequal C0), the USB transmission efficiency may be affected.
  Use the function parameters `IS_CONFIG_CPU_IDLE_STATES_CMD…` to disable these low power consumption operating states and improve USB transmission efficiency. The uc480 driver changes the current energy settings of the operating system when the first USB DCx camera is opened. After the last USB DCx camera is closed, the uc480 driver restores the original settings. The settings are valid for the current user only.
- **Windows only: Activate OpenMP (Open Multi-Processing)**
  OpenMP is a programming interface that supports distributed computing on multi-core processors. When you activate OpenMP support, intensive computing operations, such as the Bayer conversion, are distributed across several processor cores to accelerate execution. The use of OpenMP, however, increases CPU load.
- **Load camera parameters during installation**
  Use the function parameters `IS_CONFIG_INITIAL_PARAMETERSET…` to indicate whether to apply the parameters stored on the camera automatically when opening the camera. You must first store the camera parameters on the camera using the `is_ParameterSet()` function or via the corresponding function in the uc480 demo. This setting applies to all connected cameras. If no parameters are stored on the camera, the standard parameters of this camera model are applied.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

Settings for processor operating states: The settings for processor operating states are available only on Windows operating systems.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_CONFIG_CMD_GET_CAPABILITIES`	Returns the configuration options supported by the system. ⊟ Additional information • `pParam`: Pointer to a `UINT` bitmask. The status flags from `CONFIGURATION_CAPS` are returned in the bitmask.

hCam	Camera handle
	• nSizeOfParam: 4 Example 1
IS_CONFIG_CPU_IDLE_STATES_CMD_GET_ENABLE	Returns whether the current settings allow low power consumption operating states (unequal C0). ⊟ Additional information • pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL are returned in the bitmask. • nSizeOfParam: 4 Example 2
IS_CONFIG_CPU_IDLE_STATES_CMD_SET_DISABLE_ON_OPEN	Changes the energy settings of the operating system so that low power consumption operating states (unequal C0) are disabled. ⊟ Additional information • pParam: Pointer to a UINT variable, see CONFIGURATION_SEL. • nSizeOfParam: 4 **Note:** To apply a new setting, you must close all open DCx Cameras and then reopen at least one camera. Example 3
IS_CONFIG_CPU_IDLE_STATES_CMD_GET_DISABLE_ON_OPEN	Returns the current setting for IS_CONFIG_CPU_IDLE_STATES_CMD_SET_DISABLE_ON_OPEN. ⊟ Additional information • pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL are returned in the bitmask. • nSizeOfParam: 4 Example 3
IS_CONFIG_OPEN_MP_CMD_GET_ENABLE	Returns whether OpenMP support is enabled. ⊟ Additional information • pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL are returned in the bitmask. • nSizeOfParam: 4 Example 4
IS_CONFIG_OPEN_MP_CMD_SET_ENABLE	Enables OpenMP support. ⊟ Additional information • pParam: Pointer to a UINT variable, to enable: IS_CONFIG_OPEN_MP_ENABLE to disable: IS_CONFIG_OPEN_MP_DISABLE • nSizeOfParam: 4 **Note:** The settings are lost after the application is closed and must be set again the next time the

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	camera is started. Example 4
IS_CONFIG_OPEN_MP_CMD_GET_ENABLE_DEFAULT	Returns the default setting for OpenMP support. Additional information • pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL are returned in the bitmask. • nSizeOfParam: 4 Example 4
IS_CONFIG_INITIAL_PARAMETERSET_CMD_SET	Sets the parameter set to read and apply from the camera EEPROM when the camera is opened. Additional information • pParam: Pointer to a UINT variable, IS_CONFIG_INITIAL_PARAMETERSET_NONE IS_CONFIG_INITIAL_PARAMETERSET_1 IS_CONFIG_INITIAL_PARAMETERSET_2 • nSizeOfParam: 4 Example 5
IS_CONFIG_INITIAL_PARAMETERSET_CMD_GET	Returns which parameter set will be read and applied from the camera EEPROM when the camera is opened. Additional information • pParam: Pointer to a UINT variable, IS_CONFIG_INITIAL_PARAMETERSET_NONE IS_CONFIG_INITIAL_PARAMETERSET_1 IS_CONFIG_INITIAL_PARAMETERSET_2 • nSizeOfParam: 4 Example 5
pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the CONFIGURATION_CAPS Structure

INT	IS_CONFIG_CPU_IDLE_STATES_CAP_SUPPORTED	Function parameters for setting the processor operating states are supported.
INT	IS_CONFIG_OPEN_MP_CAP_SUPPORTED	Function parameters to configure OpenMP are supported.
INT	IS_CONFIG_INITIAL_PARAMETERSET_CAP_SUPPORTED	Function parameters to load camera parameters during initialization are supported.

Contents of the CONFIGURATION_SEL Structure

INT	IS_CONFIG_CPU_IDLE_STATES_BIT_AC_VALUE	Set/recover processor operating states for power supply unit operation
INT	IS_CONFIG_CPU_IDLE_STATES_BIT_DC_VALU	Set/recover processor operating states for

	E	battery operation
INT	IS_CONFIG_OPEN_MP_DISABLE	OpenMP support disabled
INT	IS_CONFIG_OPEN_MP_ENABLE	OpenMP support enabled
INT	IS_CONFIG_INITIAL_PARAMETERSET_NONE	Load camera parameters during initialization disabled
INT	IS_CONFIG_INITIAL_PARAMETERSET_1	Load camera parameter set 1 during initialization
INT	IS_CONFIG_INITIAL_PARAMETERSET_2	Load camera parameter set 2 during initialization

Return values

IS_CANT_OPEN_REGISTRY	Error opening a Windows registry key
IS_CANT_READ_REGISTRY	Error reading settings from the Windows registry
IS_ERROR_CPU_IDLE_STATES_CONFIGURATION	The configuration of the CPU idle has failed.
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_IMAGE_MEM_ALLOCATED	The driver could not allocate memory.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OPERATING_SYSTEM_NOT_SUPPORTED	Operating system not supported
IS_SUCCESS	Function executed successfully

Related functions

- is_ParameterSet()

© 2015 Thorlabs Scientific Imaging

Example 1

```
UINT nCaps = 0;
INT nRet = is_Configuration(IS_CONFIG_CMD_GET_CAPABILITIES, (void*)&nCaps, sizeof(UINT));
if (nRet == IS_SUCCESS)
{
    if (nCaps & IS_CONFIG_CPU_IDLE_STATES_CAP_SUPPORTED)
    {
        // CPU idle states supported
    }

    if (nCaps & IS_CONFIG_OPEN_MP_CAP_SUPPORTED)
    {
        // OpenMP supported
    }

    if (nCaps & IS_CONFIG_INITIAL_PARAMETERSET_CAP_SUPPORTED)
    {
        // Initial parameter set supported
    }
}
```

Example 2

```
INT nCurrentCpuStates = 0;
INT nRet = is_Configuration(IS_CONFIG_CPU_IDLE_STATES_CMD_GET_ENABLE,
                            (void*)&nCurrentCpuStates,
                            sizeof(nCurrentCpuStates)
                           );

if (nRet == IS_SUCCESS)
{
    if ((nCurrentCpuStates & IS_CONFIG_CPU_IDLE_STATES_BIT_AC_VALUE) == 0)
    {
        // The CPU idle states for mains power is already deactivated
    }

    if ((nCurrentCpuStates & IS_CONFIG_CPU_IDLE_STATES_BIT_DC_VALUE) == 0)
    {
        // The CPU idle states for battery power is already deactivated
    }
}
```

Example 3

```
UINT nCpuStates = IS_CONFIG_CPU_IDLE_STATES_BIT_AC_VALUE | IS_CONFIG_CPU_IDLE_STATES_BIT_DC_VALUE;

INT nRet = is_Configuration(IS_CONFIG_CPU_IDLE_STATES_CMD_SET_DISABLE_ON_OPEN,
                            (void*)&nCpuStates,
                            sizeof(nCpuStates)
                           );

if (nRet == IS_SUCCESS)
{
    nCpuStates = 0;
    nRet = is_Configuration(IS_CONFIG_CPU_IDLE_STATES_CMD_GET_DISABLE_ON_OPEN,
                            (void*)&nCpuStates,
                            sizeof(nCpuStates)
                           );

    if (nRet == IS_SUCCESS)
    {
        if (nCpuStates & IS_CONFIG_CPU_IDLE_STATES_BIT_AC_VALUE)
        {
            // CPU idle states for mains power are deactivated when camera is opened
        }

        if (nCpuStates & IS_CONFIG_CPU_IDLE_STATES_BIT_DC_VALUE)
        {
            // CPU idle states for battery power are deactivated when camera is opened
        }
```

```
    }
}
```


© 2015 Thorlabs Scientific Imaging

Example 4

```
UINT nEnabled = 0;
INT nRet = is_Configuration(IS_CONFIG_OPEN_MP_CMD_GET_ENABLE, (void*)&nEnabled, sizeof(nEnabled));
if (nRet == IS_SUCCESS)
{
    if (nEnabled == IS_CONFIG_OPEN_MP_ENABLE)
    {
        // OpenMP enabled
    }
}

nEnabled = 0;
nRet = is_Configuration(IS_CONFIG_OPEN_MP_CMD_GET_ENABLE_DEFAULT, (void*)&nEnabled, sizeof(nEnabled));
if (nRet == IS_SUCCESS)
{
    nRet = is_Configuration(IS_CONFIG_OPEN_MP_CMD_SET_ENABLE, (void*)&nEnabled, sizeof(nEnabled));
    if (nRet == IS_SUCCESS)
    {
        // Default value set
    }
}
```

Example 5

```
UINT nNumber = 0;
INT nRet = is_Configuration(IS_CONFIG_INITIAL_PARAMETERSET_CMD_GET, (void*)&nNumber, sizeof(nNumber));
if (nRet == IS_SUCCESS)
{
    if (nNumber == IS_CONFIG_INITIAL_PARAMETERSET_NONE)
    {
        // No parameter set specified
    }
    else if (nNumber == IS_CONFIG_INITIAL_PARAMETERSET_1)
    {
        // Parameter set 1
    }
    else if (nNumber == IS_CONFIG_INITIAL_PARAMETERSET_2)
    {
        // Parameter set 2
    }
}

nNumber = IS_CONFIG_INITIAL_PARAMETERSET_2;
is_Configuration(IS_CONFIG_INITIAL_PARAMETERSET_CMD_SET, (void*)&nNumber, sizeof(nNumber));
if (nRet == IS_SUCCESS)
{
    // Set to parameter set 2
}
```

4.3.12 is_Convert

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
IDSEXP is_Convert(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

`is_Convert()` is a general function for conversions. In version 4.60 a raw Bayer image is converted to the desired format. You can set all parameters, which are important for software conversion:

- Pixel format
- Pixel converter (3×3, 5×5)
- Color correction
- Gamma
- Saturation
- Edge enhancement

The target buffer must be allocated with the `is_AllocImageMem()` function and must have the right size.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

The following functions are obsolete by the `is_Convert()` function:

- `is_ConvertImage()`
- `is_SetConvertParam()`

See also Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_CONVERT_CMD_APPLY_PARAMS_AND_CONVERT_BUFFER`	Converts a raw Bayer buffer with the passed conversion parameters
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

© 2015 Thorlabs Scientific Imaging

Contents of the BUFFER_CONVERSION_PARAMS structure

char*	pSourceBuffer	Pointer to the raw Bayer buffer which was created with the is_AllocImageMem() function.
char*	pDestBuffer	Pointer to the target buffer with the converted data which was created with the is_AllocImageMem() function.
INT	nDestPixelFormat	Color mode of the target image; see is_SetColorMode() for the possible modes
INT	nDestPixelConverter	Conversion mode of the target image; see is_SetColorConverter() for the possible modes
INT	nDestGamma	Sets the gamma correction, see is_SetGamma()
INT	nDestEdgeEnhancement	Sets the edge enhancement, see is_EdgeEnhancement()
INT	nDestColorCorrectionMode	Sets the color correction, see is_SetColorCorrection()
INT	nDestSaturationU	Sets the color saturation (saturation U), see is_SetSaturation()
INT	nDestSaturationV	Sets the color saturation (saturation V),see is_SetSaturation()

Return value

IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetColorMode()
- is_SetBayerConversion()

Example

```
BUFFER_CONVERSION_PARAMS conversionParams;

conversionParams.nDestPixelFormat                = IS_CM_BGRA8_PACKED;
conversionParams.nDestPixelConverter             = IS_CONV_MODE_SOFTWARE_3X3;
conversionParams.nDestColorCorrectionMode        = IS_CCOR_DISABLE;
conversionParams.nDestGamma                      = 100;
conversionParams.nDestSaturationU                = 100;
conversionParams.nDestSaturationV                = 100;
conversionParams.nDestEdgeEnhancement            = 0;

conversionParams.pSourceBuffer                   = pSourceBuffer;
conversionParams.pDestBuffer                     = pDestBuffer;

INT nRet = is_Convert(m_hCam,
                      IS_CONVERT_CMD_APPLY_PARAMS_AND_CONVERT_BUFFER,
                      (void*)&conversionParams,
                      sizeof(conversionParams)
                      );
```

4.3.13 is_CopyImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_CopyImageMem (HIDS hCam, char* pcSource, INT nID, char* pcDest)
```

Description

`is_CopyImageMem()` copies the contents of the image memory described by `pcSource` and `nID` to the memory area to whose starting address `pcDest` points.

Attention

The allocated memory must be large enough to accommodate the entire image in its current format (bits per pixel).

Input parameters

`hCam`	Camera handle
`pcSource`	Pointer to the image memory
`nID`	ID of this image memory
`pcDest`	Pointer to the destination memory to copy the image to

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`
- `is_SetAllocatedImageMem()`

© 2015 Thorlabs Scientific Imaging

4.3.14 is_CopyImageMemLines

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_CopyImageMemLines (HIDS hCam, char* pcSource,
                          INT nID, INT nLines, char* pcDest)
```

Description

`is_CopyImageMemLines()` copies the contents of the image memory described by `pcSource` and `nID` to the memory area to whose starting address `pcDest` points. The function only copies the number of lines indicated by `nLines`.

Attention

The allocated memory must be large enough to accommodate the in `nLines` given number of image lines considering the image width and format (Bits per Pixel).

Input parameters

`hCam`	Camera handle
`pcSource`	Pointer to the image memory
`nID`	ID of this image memory
`nLines`	Number of lines to be copied
`pcDest`	Pointer to the destination memory to copy the image to

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`
- `is_SetAllocatedImageMem()`

4.3.15 is_DeviceFeature

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Note

This command is supported by DCC1240x / DCC3240x cameras only!

Syntax

```
INT is_DeviceFeature (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

Using `is_DeviceFeature()` you can configure special camera functions provided by DCC1240x / DCC3240x Camera models:

- Set line scan mode, see Basics: Line scan mode (only monochrome models).
- Toggle between shutter modes, see Basics: Shutter methods.
- Control the Log mode, see uc480 Viewer: Shutter.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Input parameters

hCam	Camera handle
⊟ nCommand	
IS_DEVICE_FEATURE_CMD_GET_SUPPORTED_FEATURES	Returns the functions supported by the camera. See Status flags table (Example 1)
IS_DEVICE_FEATURE_CMD_SET_LINESCAN_MODE	Sets the line scan mode, see DEVICE_FEATURE_MODE_CAPS
IS_DEVICE_FEATURE_CMD_GET_LINESCAN_MODE	Returns the currently set line scan mode
IS_DEVICE_FEATURE_CMD_SET_LINESCAN_NUMBER	Sets the scan line used for the line scan mode
IS_DEVICE_FEATURE_CMD_GET_LINESCAN_NUMBER	Returns the scan line used for the line scan mode
IS_DEVICE_FEATURE_CMD_SET_SHUTTER_MODE	Sets the shutter mode, see DEVICE_FEATURE_MODE_CAPS
IS_DEVICE_FEATURE_CMD_GET_SHUTTER_MODE	Returns the shutter mode
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_DEFAULT	Returns the default settings for the Log mode (Example 2)
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE	Returns the currect Log mode (Example 3)
IS_DEVICE_FEATURE_CMD_SET_LOG_MODE	Sets the Log mode
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE_DEFAULT	Returns the default settings for the manual value of the Log mode (Example 4)
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE_RANGE	Returns the range of the manual value of the Log mode.

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE	Returns the current manual value of the Log mode (Example 5)
IS_DEVICE_FEATURE_CMD_SET_LOG_MODE_MANUAL_VALUE	Sets the manual value of the Log mode
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN_DEFAULT	Returns the default settings for the manual gain for the Log mode (Example 6)
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN_RANGE	Returns the range for the manual gain of the Log mode
IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN	Returns the current manual gain of the Log mode(Example 7)
IS_DEVICE_FEATURE_CMD_SET_LOG_MODE_MANUAL_GAIN	Sets the manual gain of the Log mode
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Status Flags from DEVICE_FEATURE_MODE_CAPS

IS_DEVICE_FEATURE_CAP_SHUTTER_MODE_ROLLING	Rolling shutter mode is supported/Set mode
IS_DEVICE_FEATURE_CAP_SHUTTER_MODE_ROLLING_GLOBAL_START	Rolling shutter mode with global start is supported/Set mode
IS_DEVICE_FEATURE_CAP_SHUTTER_MODE_GLOBAL	Global shutter mode is supported/Set mode
IS_DEVICE_FEATURE_CAP_SHUTTER_MODE_GLOBAL_ALTERNATIVE_TIMING	Global shutter mode with different timing parameters is supported/Set mode
IS_DEVICE_FEATURE_CAP_LINESCAN_MODE_FAST	Fast line scan mode is supported/Set mode
IS_DEVICE_FEATURE_CAP_LINESCAN_NUMBER	Line number at fast line scan mode is supported/Set number
IS_DEVICE_FEATURE_CAP_LOG_MODE	LinLog pixel mode is supported/Set mode

LOG_MODES

IS_LOG_MODE_FACTORY_DEFAULT	Default settings for the Log mode
IS_LOG_MODE_OFF	Log mode off
IS_LOG_MODE_MANUAL	Manual Log mode. In this case the LogMode value and the LogMode gain are effective.

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SUCCESS	Function executed successfully

Example 1

```
INT nSupportedFeatures;
INT nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_SUPPORTED_FEATURES,
                            (void*)&nSupportedFeatures, sizeof(nSupportedFeatures));
if (nRet == IS_SUCCESS)
{
  if (nSupportedFeatures & IS_DEVICE_FEATURE_CAP_LINESCAN_MODE_FAST)
  {
    // Enable line scan mode
    INT nMode = IS_DEVICE_FEATURE_CAP_LINESCAN_MODE_FAST;
    nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LINESCAN_MODE, (void*)&nMode,
                            sizeof(nMode));
    // Disable line scan mode
    nMode = 0;
    nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LINESCAN_MODE, (void*)&nMode,
                            sizeof(nMode));
    // Return line scan mode
    nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LINESCAN_MODE, (void*)&nMode,
                            sizeof(nMode));
  }
  if (nSupportedFeatures & IS_DEVICE_FEATURE_CAP_LINESCAN_NUMBER)
  {
    // Set line number
     INT nLineNumber = 512;
     nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LINESCAN_NUMBER,
                             (void*)&nLineNumber, sizeof(nLineNumber));
    nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LINESCAN_NUMBER,
                            (void*)&nLineNumber, sizeof(nLineNumber));
  }
}
```

Example 2

```
/* Read and set default Log mode */
UINT nDefaultLogMode = 0;
INT nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_DEFAULT,
```

 © 2015 Thorlabs Scientific Imaging

```
                              (void*)&nDefaultLogMode, sizeof(nDefaultLogMode));
if (nRet == IS_SUCCESS) {
  nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LOG_MODE,
                            (void*)&nDefaultLogMode, sizeof(nDefaultLogMode));
}
```

Example 3

```
/* Read current Log pixel mode */
UINT nLogMode = 0;
INT nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE,
                              (void*)&nLogMode, sizeof(nLogMode));
```

Example 4

```
/* Read default Log pixel mode manual value */
UINT nDefaultLogModeManualValue = 0;
INT nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE_DEFAULT,
                            (void*)&nDefaultLogModeManualValue,
                            sizeof(nDefaultLogModeManualValue));

/* Get the range of the manual value */
IS_RANGE_S32 nLogModeManualValueRange;
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE_RANGE,
                        (void*)&nLogModeManualValueRange,
                        sizeof(nLogModeManualValueRange);

if (nRet == IS_SUCCESS) {
  INT nMin = nLogModeManualValueRange.s32Min;
  INT nMax = nLogModeManualValueRange.s32Max;
  INT nInc = nLogModeManualValueRange.s32Inc;
}
```

Example 5

```
UINT nLogModeValue = 0;

/* Read current Log mode manual value */
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE,
                        (void*)&nLogModeValue, sizeof(nLogModeValue));

/* Set log pixel mode value */
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LOG_MODE_MANUAL_VALUE,
                        (void*)&nLogModeValue, sizeof(nLogModeValue));
```

Example 6

```
/* Read default Log mode manual gain */
UINT nDefaultLogModeManualGain = 0;
INT nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN_DEFAULT,
                            (void*)&nDefaultLogModeManualGain,
                            sizeof(nDefaultLogModeManualGain));

/* Get the range of the manual value */
IS_RANGE_S32 nLogModeManualGainRange;
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN_RANGE,
                        (void*)&nLogModeManualGainRange,
                        sizeof(nLogModeManualGainRange));

if (nRet == IS_SUCCESS) {
  INT nMin = nLogModeManualGainRange.s32Min;
  INT nMax = nLogModeManualGainRange.s32Max;
  INT nInc = nLogModeManualGainRange.s32Inc;
}
```

Example 7

```
UINT nLogModeGain = 0;

/* Read current Log mode gain */
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN,
                       (void*)&nLogModeGain, sizeof(nLogModeGain));

/* Set Log mode gain*/
nRet = is_DeviceFeature(hCam, IS_DEVICE_FEATURE_CMD_SET_LOG_MODE_MANUAL_GAIN,
                       (void*)&nLogModeGain, sizeof(nLogModeGain));
```


© 2015 Thorlabs Scientific Imaging

4.3.16 is_DeviceInfo

USB 3	USB 3

Note

This command is supported by DCC3240x USB3 cameras only!

Syntax

```
INT is_DeviceInfo (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

Using `is_DeviceInfo()`, you can query information about connected USB 3 Cameras. The resulting information is written to the `IS_DEVICE_INFO` structure. For this purpose, the cameras need not be opened.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_DEVICE_INFO_CMD_GET_DEVICE_INFO`	Returns a information structure about the specified device (Example 1)
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Contents of the IS_DEVICE_INFO structure

`IS_DEVICE_INFO_HEARTBEAT`	`infoDevHeartbeat`	Camera-related data retrieved from the camera (from the heartbeat telegram) See `IS_DEVICE_INFO_HEARTBEAT`
`IS_DEVICE_INFO_CONTROL`	`infoDevControl`	Camera-related driver data See `IS_DEVICE_INFO_CONTROL`
`BYTE`	`reserved[240]`	reserved

Contents of the IS_DEVICE_INFO::IS_DEVICE_INFO_HEARTBEAT structure

`BYTE`	`reserved_1[36]`	reserved
`DWORD`	`dwRuntimeFirmwareVersion`	Firmware version
`BYTE`	`reserved_2[8]`	reserved
`WORD`	`wTemperature`	Camera temperature in degree Celsius Bit 15: algebraic sign Bit 14...11: filled according to

		algebraic sign Bit 10...4: temperature (places before the decimal point) Bit 3...0: temperature (places after the decimal point) See the Ambient conditions chapter for permissible temperature range.
WORD	wLinkSpeed_Mb	Transfer rate: • IS_USB_HIGH_SPEED = 480 • IS_USB_SUPER_SPEED = 4000
BYTE	reserved[208]	reserved

Contents of the IS_DEVICE_INFO::IS_DEVICE_INFO_CONTROL structure

DWORD	dwDeviceId	Internal device ID of the camera
BYTE	reserved[148]	reserved

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_SUCCESS	Function executed successfully

Related functions

- is_GetCameraList()

Example 1

```
// The camera has the device ID 1
UINT nDeviceId = 1;

IS_DEVICE_INFO deviceInfo;
memset(&deviceInfo, 0, sizeof(IS_DEVICE_INFO));

INT nRet = is_DeviceInfo((HIDS)(nDeviceId | IS_USE_DEVICE_ID),
                         IS_DEVICE_INFO_CMD_GET_DEVICE_INFO,
                         (void*)&deviceInfo, sizeof(deviceInfo));
if (nRet == IS_SUCCESS)
{
  WORD wTemperature = deviceInfo.infoDevHeartbeat.wTemperature;
}
```


© 2015 Thorlabs Scientific Imaging

4.3.17 is_DirectRenderer

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_DirectRenderer (HIDS hCam, UINT nMode, void* pParam, UINT nSize)
```

Description

Note

The `is_DirectRenderer()` functions works under Linux only in OpenGL mode.

`is_DirectRenderer()` provides a set of advanced rendering functions and allows inserting overlay data into the camera's live image without flicker. The graphics card functions of the Direct3D library are supported under Windows.

The second input parameter `nMode` specifies the effect of the `is_DirectRenderer()` call.

The value of the third parameter `pParam` depends on the mode selected with `nMode`: For example, when setting the overlay size (`nMode = DR_SET_OVERLAY_SIZE`), a pointer to an array of two values (x and y) is passed (see code samples). When you load a bitmap image (`nMode = DR_LOAD_OVERLAY_FROM_FILE`), `pParam` passes the path to the file (see code samples). The required parameters are illustrated in the sample codes at the end of this section.

Notes

1. System requirements

- To use the Direct3D functionality, the appropriate version of the Microsoft DirectX Runtime has to be installed in your PC.
- When you are using high-resolution cameras, the maximum texture size supported by the graphics card should be at least 4096 x 4096 pixels. You can check the maximum texture size by reading out the `D3D_GET_MAX_OVERLAY_SIZE` parameter.
- The Direct3D mode automatically uses the Windows Desktop color depth setting for the display.

Please also read the notes on graphics cards which are provided in the System requirements chapter.

2. Displaying monochrome or raw data formats

To display monochrome or Bayer raw data in Direct3D, please set the appropriate constants using the `is_SetDisplayMode()` function.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

Input parameters

hCam	Camera handle
⊟ nMode	
DR_GET_SUPPORTED	Returns either if Direct3D or OpenGL is

hCam	Camera handle
	supported by the graphics card. More details • IS_SET_DM_DIRECT3D: Tests if Direct3D is suppported. • IS_SET_DM_OPENGL: Tests if OpenGL is supported. Example
DR_GET_OVERLAY_DC	Direct3D only: Returns the device context (DC) handle to the overlay area of the graphics card. More details In Direct3D mode, the DR_GET_OVERLAY_DC mode returns the device context (DC) handle of the overlay area. Using this handle, it is possible to access the overlay using the Windows GDI functionality. Thus, all Windows graphics commands (e.g. Line, Circle, Rectangle, TextOut) are available. To transfer the drawn elements to the overlay, release the DC handle by calling DR_RELEASE_OVERLAY_DC. Example
DR_RELEASE_OVERLAY_DC	Direct3D only: Releases the device context (DC) handle. More details Using DR_RELEASE_OVERLAY_DC, you can release the DC handle and update the overlay data. Example
DR_GET_MAX_OVERLAY_SIZE	Returns the width x and height y of the maximum overlay area supported by the graphics card. Example
DR_SET_OVERLAY_SIZE	Defines the size of the overlay area (default: current camera image size). Example
DR_GET_OVERLAY_SIZE	Returns the size of the overlay area. (Sample: see DR_SET_OVERLAY_SIZE)
DR_SET_OVERLAY_POSITION	Defines the position of the overlay area. Example
DR_GET_OVERLAY_KEY_COLOR	Returns the RGB values of the current key color (default: black). Example
DR_SET_OVERLAY_KEY_COLOR	Defines the RGB values of the key color. More details The key color specifies where the camera image will be visible in the overlay area. For example: if you fill the complete overlay with the key color, the whole camera image will be visible. If you fill part of the overlay with a different color, the camera image will be covered by the overlay in those places. The key color has no effect in semi-transparent mode!

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	Example
DR_SHOW_OVERLAY	Enables overlay display on top of the current camera image. Example
DR_HIDE_OVERLAY	Disables overlay display. Example
DR_ENABLE_SCALING	Enables real-time scaling of the image to the size of the display window. The overlay is scaled together with the camera image. Example
DR_ENABLE_IMAGE_SCALING	Direct3D only: Enables real-time scaling of the image to the size of the display window. The overlay is not scaled. (Sample: see DR_ENABLE_SCALING)
DR_DISABLE_SCALING	Disables real-time scaling. Example
DR_ENABLE_SEMI_TRANSPARENT_OVERLAY	Enables a semi-transparent display of the overlay area. More details In semi-transparent mode, the values of the camera image and the overlay data are added up for each pixel. Since black has the value 0, the complete camera image will be visible if the overlay is black; if the overlay is white, only the overlay will be visible. With all other colors, the camera image will be visible with the overlay superimposed. The key color has no effect in semi-transparent mode! Example
DR_DISABLE_SEMI_TRANSPARENT_OVERLAY	Disables the semi-transparent display of the overlay area. Example
DR_SET_VSYNC_AUTO	Enables synchronization of the image display with the monitor's image rendering. The image is displayed upon the monitor's next VSYNC signal. Example
DR_SET_VSYNC_OFF	Disables image display synchronization. The image is displayed immediately. Example
DR_SET_USER_SYNC	Direct3D only: Enables synchronization of the image display with a monitor pixel row specified by the user. More details When displaying very large camera images, the auto VSYNC function might not always optimally synchronize image rendering. In this case, you can eliminate flicker by manually setting a suitable position for synchronization. The position needs to be determined individually, based on the camera type and the graphics card. Example
DR_GET_USER_SYNC_POSITION_RANGE	Direct3D only: Returns the minimum and

hCam	Camera handle
	maximum row position for DR_SET_USER_SYNC. Example
DR_LOAD_OVERLAY_FROM_FILE	Direct3D only: Loads a bitmap image (*.BMP file) into the overlay area. If the bitmap image is larger than the overlay area, the bitmap image is clipped. Example
DR_CLEAR_OVERLAY	Deletes the data of the overlay area by filling it with black color. Example
DR_STEAL_NEXT_FRAME	Copies the next image to the active user memory (Steal function). More details Using the pParam parameter, you specify when the function should return: • IS_WAIT: The function waits until the image save is complete. • IS_DONT_WAIT: The function returns immediately. Example
DR_SET_STEAL_FORMAT	Defines the color format for the Steal function. More details For a list of all available color formats, see the function description for is_SetColorMode(). The default is IS_CM_BGRA8_PACKED (RGB 32). Example
DR_GET_STEAL_FORMAT	Returns the color format setting for the Steal function. Example
DR_SET_HWND	Sets a new window handle for image output in Direct3D. Example
DR_CHECK_COMPATIBILITY	Returns whether the graphics card supports the uc480 Direct3D functions. Example
DR_GET_OVERLAY_DATA	OpenGL only: Returns a pointer to the overlay.
DR_UPDATE_OVERLAY_DATA	OpenGL only: Updates the overlay.
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSize	Size (in bytes) of the data object or array.

© 2015 Thorlabs Scientific Imaging

Return values

When used with DR_CHECK_COMPATIBILITY	IS_DR_DEVICE_CAPS_INSUFFICIENT The graphics hardware does not fully support the uc480 Direct3D functions.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_CANNOT_GET_OVERLAY_DC	Could not get the device context handle for the overlay.
IS_DR_CANNOT_LOCK_OVERLAY_SURFACE	The overlay surface could not be locked.
IS_DR_CANNOT_RELEASE_OVERLAY_DC	Could not release the device context handle for the overlay.
IS_DR_CANNOT_UNLOCK_OVERLAY_SURFACE	The overlay surface could not be unlocked.
IS_DR_DEVICE_CAPS_INSUFFICIENT	Function is not supported by the graphics hardware.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_NOT_ALLOWED_WHILE_DC_IS_ACTIVE	A device context handle is still open in the application.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_SetDisplayMode()
- is_SetColorMode()
- is_SetImageMem()
- is_RenderBitmap()

Example Supported function

```
UINT nType = IS_SET_DM_DIRECT3D;
if (is_DirectRenderer(m_pMainView->GetCameraHandle(), DR_GET_SUPPORTED,
                      (void*)&nType, sizeof(nType)) == IS_SUCCESS)
{
  // Direct3D is supported
}
nType = IS_SET_DM_OPENGL;
if (is_DirectRenderer(m_pMainView->GetCameraHandle(), DR_GET_SUPPORTED,
                      (void*)&nType, sizeof(nType)) == IS_SUCCESS)
{
```

```
// OpenGL is supported
}
```

Example DC handle

```
//-----------------------------------
//          DC-Handle
//-----------------------------------

// Get DC handle for Overlay
HDC hDC;
is_DirectRenderer (hCam, DR_GET_OVERLAY_DC, (void*)&hDC, sizeof (hDC));

// Release DC handle
is_DirectRenderer (hCam, DR_RELEASE_OVERLAY_DC, NULL, NULL);
```


© 2015 Thorlabs Scientific Imaging

Example overlay size and position

```
//-------------------------------------
//          Size of overlay
//-------------------------------------

// Query maximum size of overlay area
UINT OverlaySize[2];
is_DirectRenderer (hCam, DR_GET_MAX_OVERLAY_SIZE,
                   (void*)OverlaySize, sizeof(OverlaySize));
INT nWidth = OverlaySize[0];
INT nHeight = OverlaySize[1];

// Set size of overlay area
UINT Size[2];
Size[0] = 100;
Size[1] = 120;
is_DirectRenderer (hCam, DR_SET_OVERLAY_SIZE,
                   (void*)Size, sizeof (Size));

// Set position of overlay area
UINT Position[2];
Position[0] = 20;
Position[1] = 0;
is_DirectRenderer (hCam, DR_SET_OVERLAY_POSITION,
                   void*)Position, sizeof (Position));
```

Example key color

```
//-------------------------------------
//          Key color
//-------------------------------------

// Get current key color
UINT OverlayKeyColor[3];
is_DirectRenderer (hCam, DR_GET_OVERLAY_KEY_COLOR,
                   (void*)OverlayKeyColor, sizeof(OverlayKeyColor));

INT nRed = OverlayKeyColor[0];
INT nGreen = OverlayKeyColor[1];
INT nBlue = OverlayKeyColor[2];

// Set new key color
OverlayKeyColor[0] = GetRValue(m_rgbKeyColor);
OverlayKeyColor[1] = GetGValue(m_rgbKeyColor);
OverlayKeyColor[2] = GetBValue(m_rgbKeyColor);
is_DirectRenderer (hCam, DR_SET_OVERLAY_KEY_COLOR,
                   (void*)OverlayKeyColor, sizeof(OverlayKeyColor));
```

Example display

```
//-------------------------------------
//          Display
//-------------------------------------

// Show overlay
is_DirectRenderer (hCam, DR_SHOW_OVERLAY, NULL, NULL);

// Hide overlay
is_DirectRenderer (hCam, DR_HIDE_OVERLAY, NULL, NULL);
```

Example scaling

```
//-------------------------------------
//          Scaling
//-------------------------------------

// Enable scaling
is_DirectRenderer (hCam, DR_ENABLE_SCALING, NULL, NULL);

// Disable scaling
is_DirectRenderer (hCam, DR_DISABLE_SCALING, NULL, NULL);
```

Example transparency

```
//-----------------------------------
//          Transparency
//-----------------------------------

// Enable semi-transparent overlay
is_DirectRenderer (hCam, DR_ENABLE_SEMI_TRANSPARENT_OVERLAY, NULL, NULL);

// Disable semi-transparent overlay
is_DirectRenderer (hCam, DR_DISABLE_SEMI_TRANSPARENT_OVERLAY, NULL, NULL);
```

Example synchronization

```
//-----------------------------------
//          Synchronization
//-----------------------------------

// Enable auto-synchronization
is_DirectRenderer (hCam, DR_SET_VSYNC_AUTO, NULL, NULL);

// User defined synchronization: Query range and set position
UINT UserSync[2];
is_DirectRenderer (hCam, DR_GET_USER_SYNC_POSITION_RANGE,
                   (void*)UserSync, sizeof (UserSync));
INT Min = UserSync[0];
INT Max = UserSync[1];
INT SyncPosition = 400;
is_DirectRenderer (hCam, DR_SET_USER_SYNC,
                   void*)&SyncPosition, sizeof (SyncPosition));

// Disable synchronization
is_DirectRenderer (hCam, DR_SET_VSYNC_OFF, NULL, NULL);
```

Example overlay with BMP

```
//-----------------------------------
//          BMP file
//-----------------------------------

// Load overlay from BMP file
is_DirectRenderer (hCam, DR_LOAD_OVERLAY_FROM_FILE,
                   (void*)”c:\test.bmp”, NULL);

//-----------------------------------
//          Delete overlay
//-----------------------------------

// Delete overlay area
is_DirectRenderer (hCam, DR_CLEAR_OVERLAY, NULL, NULL);
```

Example steal mode

```
//-----------------------------------
//          Steal mode
//-----------------------------------

// Get and set color mode for image to be copied
INT nColorMode;
is_DirectRenderer (hCam, DR_GET_STEAL_FORMAT,
                   (void*)&nColorMode, sizeof (nColorMode));

nColorMode = IS_CM_MONO8;
is_DirectRenderer (hCam, DR_SET_STEAL_FORMAT,
                   void*)&nColorMode, sizeof (nColorMode));

// Copy image with function returning immediately
INT nwait = IS_DONT_WAIT;
is_DirectRenderer(hCam, DR_STEAL_NEXT_FRAME,
                  (void*)&wait, sizeof (wait));
```

 © 2015 Thorlabs Scientific Imaging

Example window handle

```
//-----------------------------------
//          Handle to window
//-----------------------------------

// Set new window handle for image display
is_DirectRenderer (hCam, DR_SET_HWND,
                   (void*)&hWnd, sizeof (hWnd));
```

Example compatibility

```
//-----------------------------------
//          Compatibility
//-----------------------------------

// Check graphics card compatibility
INT nRet = is_DirectRenderer (hCam, DR_CHECK_COMPATIBILITY, NULL, NULL);

if (nRet == IS_DR_DEVICE_CAPS_INSUFFICIENT )
// Graphics card does not support Direct3D
```

Example OpenGL under Linux

```
//OpenGL initialize
OPENGL_DISPLAY display;
display.pDisplay = NULL;
display.nWindowID = 0 /* window id */

is_InitCamera(&hCam, (void*)&display);
```

Example under Linux (with usage of the Cairo library)

```
UINT Size[2] = { 480, 480 };
is_DirectRenderer (hCam, DR_SET_OVERLAY_SIZE, (void*)Size, sizeof (Size));

char *pOverlayBuffer;
is_DirectRenderer(hCam, DR_GET_OVERLAY_DATA, (void*)&pOverlayBuffer, sizeof(pOverlayBuffer));

cairo_surface_t *surface = 0;
cairo_t *cr = 0;
int w, h;
w = Size[0];
h = Size[1];
surface = cairo_image_surface_create_for_data(buffer, CAIRO_FORMAT_ARGB32, w, h, w * 4);
cr = cairo_create(surface);
cairo_set_line_width (cr, 6);
cairo_rectangle (cr, 12, 12, 232, 70);
cairo_new_sub_path (cr); cairo_arc (cr, 64, 64, 40, 0, 2* 3.14);
cairo_new_sub_path (cr); cairo_arc_negative (cr, 192, 64, 40, 0, -2*3.14);
cairo_set_fill_rule (cr, CAIRO_FILL_RULE_EVEN_ODD);
cairo_set_source_rgb (cr, 0, 0.7, 0); cairo_fill_preserve (cr);
cairo_set_source_rgb (cr, 0, 0, 0); cairo_stroke (cr);
cairo_translate (cr, 0, 128);
cairo_rectangle (cr, 12, 12, 232, 70);
cairo_new_sub_path (cr); cairo_arc (cr, 64, 64, 40, 0, 2*3.14);
cairo_new_sub_path (cr); cairo_arc_negative (cr, 192, 64, 40, 0, -2*3.14);
cairo_set_fill_rule (cr, CAIRO_FILL_RULE_WINDING);
cairo_set_source_rgb (cr, 0, 0, 0.9); cairo_fill_preserve (cr);
cairo_set_source_rgb (cr, 0, 0, 0); cairo_stroke (cr);
cairo_select_font_face (cr, "Sans", CAIRO_FONT_SLANT_NORMAL,
                        CAIRO_FONT_WEIGHT_BOLD );
cairo_set_font_size (cr, 90.0);
cairo_move_to (cr, 10.0, 135.0);
cairo_show_text (cr, "Hello");
cairo_move_to (cr, 70.0, 165.0);
cairo_set_font_size (cr, 150.0);
cairo_text_path (cr, "uc480");
cairo_set_source_rgb (cr, 0.5, 0.5, 1);
cairo_fill_preserve (cr);
cairo_set_source_rgb (cr, 0, 0, 0);
cairo_set_line_width (cr, 2.56);
cairo_stroke (cr);
/* draw helping lines */
cairo_set_source_rgba (cr, 1, 0.2, 0.2, 0.6);
cairo_arc (cr, 10.0, 135.0, 5.12, 0, 2*3.14);
cairo_close_path (cr);
cairo_arc (cr, 70.0, 165.0, 5.12, 0, 2*3.14);
cairo_fill (cr);
cairo_destroy (cr);
cairo_surface_destroy (surface);

// update overlay
is_DirectRenderer(hCam, DR_UPDATE_OVERLAY_DATA, NULL, 0);
```

Sample programs

- uc480DirectRenderer
- uc480Steal

 © 2015 Thorlabs Scientific Imaging

4.3.18 is_DisableEvent

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_DisableEvent (HIDS hCam, INT which)
```

Description

Using `is_DisableEvent()`, you disable the event indicated here. The event (e.g. image capture completed) will usually still occur, but will no longer trigger an event signal. Disabled events are no longer signaled to the application. You can re-enable the desired event using `is_EnableEvent()`. See also `is_InitEvent()`.

Input parameters

`hCam`	Camera handle
`which`	ID of the event to be disabled. See also `is_EnableEvent()`.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_EnableEvent()`
- Windows only: `is_InitEvent()`
- Windows only: `is_ExitEvent()`
- Linux only: `is_WaitEvent()`

4.3.19 is_EdgeEnhancement

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_EdgeEnhancement(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

This function enables/disables a software edge filter.

Due to Bayer format color conversion, the original edges of a color image may easily become blurred. By enabling the digital edge filter, you can optimize edge representation. This function causes a higher CPU load.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

The following function is obsolete by the `is_EdgeEnhancement()` function:

- `is_SetEdgeEnhancement()`

See also Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_EDGE_ENHANCEMENT_CMD_GET_RANGE`	Returns the range of the edge enhancement (Example 1)
`IS_EDGE_ENHANCEMENT_CMD_GET_DEFAULT`	Returns the standard value of the edge enhancement (Example 2)
`IS_EDGE_ENHANCEMENT_CMD_GET`	Returns the current set edge enhancement (Example 3)
`IS_EDGE_ENHANCEMENT_CMD_SET`	Sets the edge enhancement (Example 4) 0: no edge enhancement
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Return values

`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

 © 2015 Thorlabs Scientific Imaging

Related functions

- `is_SetColorMode()`
- `is_SetColorConverter()`

Example 1

```
UINT nRange[3];
ZeroMemory(nRange, sizeof(nRange));

INT nRet = is_EdgeEnhancement(m_hCam,
                              IS_EDGE_ENHANCEMENT_CMD_GET_RANGE,
                              (void*)nRange,
                              sizeof(nRange)
                             );

if (nRet == IS_SUCCESS)
{
    UINT nEdgeEnhancementMin = nRange[0];
    UINT nEdgeEnhancementMax = nRange[1];
    UINT nEdgeEnhancementInc = nRange[2];
}
```

Example 2

```
UINT nDefault;
INT nRet = is_EdgeEnhancement(m_hCam,
                              IS_EDGE_ENHANCEMENT_CMD_GET_DEFAULT,
                              (void*)&nDefault,
                              sizeof(nDefault)
                             );
```

Example 3

```
UINT nEdgeEnhancement;
INT nRet = is_EdgeEnhancement(m_hCam,
                              IS_EDGE_ENHANCEMENT_CMD_GET,
                              (void*)&nEdgeEnhancement,
                              sizeof(nEdgeEnhancement)
                             );
```

Example 4

```
UINT nEdgeEnhancement = 4;
INT nRet = is_EdgeEnhancement(m_hCam,
                              IS_EDGE_ENHANCEMENT_CMD_SET,
                              (void*)&nEdgeEnhancement,
                              sizeof(nEdgeEnhancement)
                             );
```


© 2015 Thorlabs Scientific Imaging

4.3.20 is_EnableAutoExit

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_EnableAutoExit (HIDS hCam, INT nMode)
```

Description

`is_EnableAutoExit()` enables automatic closing of the camera handle after a camera has been removed on-the-fly. Upon closing of the handle, the entire memory allocated by the driver will be released.

Input parameters

`hCam`	Camera handle
⊟ `nMode`	
`IS_ENABLE_AUTO_EXIT`	Enables automatic closing
`IS_DISABLE_AUTO_EXIT`	Disables automatic closing
`IS_GET_AUTO_EXIT_ENABLED`	Returns the current setting

Return values

Current setting when used together with`IS_GET_AUTO_EXIT_ENABLED`	
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_ExitCamera()`

4.3.21 is_EnableEvent

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_EnableEvent (HIDS hCam, INT which)
```

Description

Using `is_EnableEvent()`, you release an event object. Following the release, the event messages for the created event object are enabled. Depending on the operating system different functions are to call.

Windows	• Event has to be provided by the application program • Event has to be declared by `is_InitEvent()` • Event has to be activated by `is_EnableEvent()` • You have to wait for the event in the application program by `WaitForSingleObject` or `WaitForMultipleObject` • Event has to be deactivated by `is_DisableEvent()` • Event has to be logged off by `is_ExitEvent()`
Linux	• Event has to be provided by the uc480 API • Event has to be activated by `is_EnableEvent()` • You have to wait for the event by `is_WaitEvent()` • Event has to be deactivated by `is_DisableEvent()`

Input parameters

`hCam`	Camera handle
⊟ `which:` ID of the event to be released.	
`IS_SET_EVENT_AUTOBRIGHTNESS_FINISHED`	The automatic brightness control in the run-once mode is completed.
`IS_SET_EVENT_CAMERA_MEMORY`	In the camera memory mode an image acquisition iteration is finished.
`IS_SET_EVENT_CAPTURE_STATUS`	There is an information about image capturing available. This information can be requested by `is_CaptureStatus()`. Note that this event replaces the former `IS_SET_EVENT_TRANSFER_FAILED` from previous versions.
`IS_SET_EVENT_CONNECTIONSPEED_CHANGED`	The connection speed of a USB 3 DCx camera changed from USB 2.0 to USB 3.0 or from USB 3.0 to USB 2.0.
`IS_SET_EVENT_DEVICE_RECONNECTED`	A camera initialized with `is_InitCamera()` and disconnected afterwards was reconnected.
`IS_SET_EVENT_EXTTRIG`	An image which was captured following the arrival of a trigger has been transferred completely.

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	This is the earliest possible moment for a new capturing process. The image must then be post-processed by the driver and will be available after the IS_FRAME processing event.
IS_SET_EVENT_FRAME	A new image is available.
IS_SET_EVENT_NEW_DEVICE	A new camera was connected. This is independent of the device handle (hCam is ignored).
IS_SET_EVENT_OVERLAY_DATA_LOST	Direct3D/OpenGL mode: Because of a re-programming the parameters of the overlay are invalid. The overlay must be draw new.
IS_SET_EVENT_REMOVAL	A camera was removed. This is independent of the device handle (hCam is ignored).
IS_SET_EVENT_REMOVE	A camera initialized with is_InitCamera() was disconnected.
IS_SET_EVENT_SEQ	The sequence is completed.
IS_SET_EVENT_STATUS_CHANGED	**Linux only:** The availability of a camera has changed, e.g. an available camera was opened.
IS_SET_EVENT_STEAL	An image extracted from the overlay is available.
IS_SET_EVENT_WB_FINISHED	The automatic white balance control is completed.

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_DisableEvent()
- Windows only: is_InitEvent()
- Windows only: is_ExitEvent()
- Linux only: is_WaitEvent()

Example Windows

```
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
is_InitEvent(hCam, hEvent, IS_SET_EVENT_FRAME);
is_EnableEvent(hCam, IS_SET_EVENT_FRAME);
is_FreezeVideo(hCam, IS_DONT_WAIT);
DWORD dwRet = WaitForSingleObject(hEvent, 1000);
if (dwRet == WAIT_TIMEOUT)
{
    /* wait timed out */
```

```
}
else if (dwRet == WAIT_OBJECT_0)
{
    /* event signalled */
}
is_DisableEvent(hCam, IS_SET_EVENT_FRAME);
is_ExitEvent(hCam, IS_SET_EVENT_FRAME);
CloseHandle(hEvent);
```

Example Linux

```
is_EnableEvent(hCam, IS_SET_EVENT_FRAME);
is_FreezeVideo(hCam, IS_DONT_WAIT);
INT nRet = is_WaitEvent(hCam, IS_SET_EVENT_FRAME, 1000);
if (nRet == IS_TIMED_OUT)
{
    /* wait timed out */
}
else if (nRet == IS_SUCCESS)
{
    /* event signalled */
}
is_DisableEvent(hCam, IS_SET_EVENT_FRAME);
```

Sample programs

- SimpleLive (C++)
- uc480Event (C++)

© 2015 Thorlabs Scientific Imaging

4.3.22 is_EnableMessage

USB 2.0 USB 3.0	-

Syntax

```
INT is_EnableMessage (HIDS hCam, INT which, HWND hWnd)
```

Description

Using `is_EnableMessage()`, you can enable Windows messages. If a particular event occurs, the messages are sent to the application.

Each message is structured as follows:

- Message: `IS_uc480_MESSAGE`
- `wParam`: Event (see table)
- `lParam`: DCx camera handle associated with the message

Attention

You have to deactivate Windows messages with `hWnd == NULL` before you free the uc480 API library. Otherwise the application may not close properly.

Input parameters

`hCam`	Camera handle
⊟ `which`: ID of the message to be enabled/disabled	
`IS_FRAME`	A new image is available.
`IS_SEQUENCE`	The sequence is completed.
`IS_CAPTURE_STATUS`	An error occurred during the data transfer, see `is_CaptureStatus()`. The parameter `IS_CAPTURE_STATUS` replaces the previous parameter `IS_TRANSFER_FAILED`. The parameter `IS_TRANSFER_FAILED` was moved into the new header file `uc480_deprecated.h`, which contains all obsolete function definitions and constants. If necessary the header file `uc480_deprecated.h` can be included in addition to the header file `uc480.h`.
`IS_TRIGGER`	An image which was captured following the arrival of a trigger has been transferred completely. This is the earliest possible moment for a new capturing process. The image must then be post-processed by the driver and is available after the `IS_FRAME` message has occurred.
`IS_DEVICE_REMOVED`	A camera initialized with `is_InitCamera()` was disconnected.
`IS_DEVICE_RECONNECTED`	A camera initialized with `is_InitCamera()` and disconnected afterwards was reconnected.

hCam	Camera handle
IS_NEW_DEVICE	A new camera was connected.
IS_DEVICE_REMOVAL	A camera was removed.
IS_WB_FINISHED	Automatic white balance control is completed (only if this control was started using the IS_SET_AUTO_WB_ONCE function).
IS_AUTOBRIGHTNESS_FINISHED	Automatic brightness control is completed (only if this control was started using the IS_SET_AUTO_BRIGHTNESS_ONCE function).
IS_CAMERA_MEMORY	In the camera memory mode an image acquisition iteration is finished.
IS_CONNECTIONSPEED_CHANGED	The connection speed of a USB 3 DCx camera changed from USB 2.0 to USB 3.0 or from USB 3.0 to USB 2.0.
hWnd	Application window for receiving the message. NULL disables the message designated by the which parameter.

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_InitEvent()

© 2015 Thorlabs Scientific Imaging

4.3.23 is_ExitCamera

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ExitCamera (HIDS hCam)
```

Description

`is_ExitCamera()` disables the `hCam` camera handle and releases the data structures and memory areas taken up by the DCx camera. Image memory allocated using the `is_AllocImageMem()` function which has not been released yet is automatically released.

Note

We recommend that you call the following functions only from a single thread in order to avoid unpredictable behaviour of the application.

- `is_InitCamera()`
- `is_SetDisplayMode()`
- `is_ExitCamera()`

See also Programming: Thread programming

Input parameters

`hCam`	Camera handle

Return values

`IS_CANT_OPEN_REGISTRY`	Error opening a Windows registry key
`IS_CANT_READ_REGISTRY`	Error reading settings from the Windows registry
`IS_ERROR_CPU_IDLE_STATES_CONFIGURATION`	The configuration of the CPU idle has failed.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_IMAGE_MEM_ALLOCATED`	The driver could not allocate memory.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_InitCamera()`
- `is_EnableAutoExit()`

4.3.24 is_ExitEvent

USB 2.0 USB 3.0	-

Syntax

```
INT is_ExitEvent (HIDS hCam, INT which)
```

Description

`is_ExitEvent()` deletes an existing event object. After an event has been deleted, you can no longer enable it by calling the `is_EnableEvent()` function.

Input parameters

`hCam`	Camera handle
`which`	ID of the event to be deleted. See also `is_EnbaleEvent()`.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_EnableEvent()`
- `is_InitEvent()`

Example

See also `is_ForceTrigger()`

© 2015 Thorlabs Scientific Imaging

4.3.25 is_ExitImageQueue

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ExitImageQueue (HIDS hCam)
```

Description

`is_ExitImageQueue()` deletes a queue which has been initialized with `is_InitImageQueue()` and discards all information about the order of queued images. The image memories will be unlocked. The memory sequence itself persists and can be deleted with `is_ClearSequence()`.

Input parameters

hCam	Camera handle

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- `is_InitImageQueue()`
- `is_WaitForNextImage()`

4.3.26 is_Exposure

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_Exposure (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

Using `is_Exposure()` you can query the exposure time ranges available in your camera, and set new exposure times.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Notes

1. Dependencies on other settings

The use of the following functions will affect the exposure time:

- `is_PixelClock()`
- `is_SetOptimalCameraTiming()`
- `is_SetFrameRate()`
- `is_AOI()` (if the image size is changed)
- `is_SetSubSampling()`
- `is_SetBinning()`

Changes made to the image size, the frame rate or the pixel clock frequency also affect the exposure time. For this reason, you need to call `is_Exposure()` again after such changes.

2. New driver versions

Newer driver versions sometimes allow an extended value range for the exposure time setting. We recommend querying the value range every time and set the exposure time explicitly.

Applying new settings

In freerun mode (`is_CaptureVideo()`), any modification of the exposure time will only become effective when the next image but one is captured. In trigger mode (`is_SetExternalTrigger()`), the modification will be applied to the next image. See also the Applying new parameters chapter.

Accuracy of the exposure time setting

The increments for setting the exposure time (`IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_INC`) depend on the sensor's current timing settings (pixel clock, frame rate). The smallest increment usually corresponds to the duration of one pixel row, which is the time it takes the sensor to read out one pixel row.

You can query the actual exposure time setting with the `IS_EXPOSURE_CMD_GET_EXPOSURE` parameter.

Some sensors allow setting the exposure time in smaller increments. Using the `IS_EXPOSURE_CMD_GET_CAPS` parameter, you can check whether your sensor supports this function.

For minimum and maximum exposure times as well as other sensor-based dependencies, please refer to the Camera and sensor data chapter.

 © 2015 Thorlabs Scientific Imaging

Rounding errors from increments

When calculating a new exposure time based on the returned increment, note that calculations with floating point values in the PC will always be subject to rounding errors. Therefore, an addition or subtraction of the `n*INCREMENT` value might not always produce the exact desired result. In this case, the uc480 API rounds down the floating point value and sets the exposure time to the next lower value.

You can avoid this behavior by additionally adding the value `INCREMENT/2.f` (half increment) when calculating with `n*INCREMENT`. This ensures that the desired value will be set even after rounding.

Note

Older uc480 exposure time functions

The following uc480 API commands are obsolete by the `is_Exposure()` function:

- `is_GetExposureRange()`
- `is_SetExposureTime()`

See also Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_EXPOSURE_CMD_GET_CAPS`	Returns the supported function modes. ⊟ More details • `pParam`: Pointer to bit mask of type `UINT` In the bit mask, the status flags from `EXPOSURE_CAPS` are returned. • `nSizeOfParam`: 4 Example 1
`IS_EXPOSURE_CMD_GET_EXPOSURE_DEFAULT`	Returns the default setting for the exposure time. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the default value. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_EXPOSURE`	Returns the currently set exposure time (in ms). ⊟ More details • `pParam`: Pointer to variable of type `double` returning the current value. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MIN`	Returns the minimum exposure time. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the minimum value. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MAX`	Returns the maximum exposure time. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the maximum value. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_INC`	Returns the exposure time increment. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the increment. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE`	Returns the exposure time range. ⊟ More details • `pParam`: Pointer to array of type `double`

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	returning the minimum and maximum values and the increment (in exactly this order). • `nSizeOfParam`: 24
`IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MIN`	Returns the minimum exposure time in fine increments for some sensors. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the minimum value. • `nSizeOfParam`: 8 Example 2
`IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MAX`	Returns the maximum exposure time in fine increments for some sensors. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the maximum value. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_INC`	Returns the exposure time increment in fine increments for some sensors. ⊟ More details • `pParam`: Pointer to variable of type `double` returning the increment. • `nSizeOfParam`: 8
`IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE`	Returns the exposure time range in fine increments for some sensors. ⊟ More details • `pParam`: Pointer to array of type `double` returning the minimum and maximum values and the increment (in exactly this order). • `nSizeOfParam`: 24 Example 3
`IS_EXPOSURE_CMD_SET_EXPOSURE`	Sets the exposure time (in ms). ⊟ More details • `pParam`: Pointer to variable of type `double` that passes the value to be set. After setting the exposure time this value contains the actually set exposure time. Depending on the sensor the set exposure time may vary slightly from the desired exposure time. • `nSizeOfParam`: 8 If 0 is passed, the exposure time is set to the maximum value of 1/frame rate.
`IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE_MIN`	Returns the minimum long exposure time. ⊟ More information • `pParam`: Pointer to variable of type `double`

hCam	Camera handle
	returning the minimum value. • nSizeOfParam: 8
IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE_MAX	Returns the maximum long exposure time. More information • pParam: Pointer to variable of type double returning the maximum value. • nSizeOfParam: 8
IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE_INC	Returns the increments for long exposure. More information • pParam: Pointer to variable of the type double returning the increment. • nSizeOfParam: 8
IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE	Returns the value range for long exposure. More information • pParam: Pointer to an array of the type double returning the minmum and maximum values and the increment. • nSizeOfParam: 24
IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_ENABLE	Returns the current settings for long exposure. More information pval1: returns the current setting
IS_EXPOSURE_CMD_SET_LONG_EXPOSURE_ENABLE	Enables/Disables long exposure. More information pval1 = 1 enables control, 0 disables control
pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the EXPOSURE_CAPS structure

INT	IS_EXPOSURE_CAP_EXPOSURE	The exposure time setting is supported
INT	IS_EXPOSURE_CAP_FINE_INCREMENT	Fine exposure time increments are supported
INT	IS_EXPOSURE_CAP_LONG_EXPOSURE	Long time exposure is supported

Return values

IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle

© 2015 Thorlabs Scientific Imaging

IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_SetFrameRate()
- is_PixelClock()
- is_SetOptimalCameraTiming()
- is_SetAutoParameter()
- is_AutoParameter()
- is_SetHardwareGain()

Example 1

```
UINT nCaps = 0;
INT nRet = is_Exposure(m_hCam, IS_EXPOSURE_CMD_GET_CAPS, (void*)&nCaps, sizeof(nCaps));

if (nRet == IS_SUCCESS)
{
    if (nCaps & IS_EXPOSURE_CAP_FINE_INCREMENT)
    {
        // Fine increment supported
    }
}
```

Example 2

```
double dblMin, dblMax, dblInc;

INT nRet = is_Exposure(m_hCam,
               IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MIN,
              (void*)&dblMin,
              sizeof(dblMin)
             );

INT nRet = is_Exposure(m_hCam,
               IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MAX,
              (void*)&dblMax,
              sizeof(dblMax)
             );


INT nRet = is_Exposure(m_hCam,
               IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_INC,
              (void*)&dblInc,
              sizeof(dblInc)
             );
```

Example 3

```
double dblRange[3];
double dblMin, dblMax, dblInc;

INT nRet = is_Exposure(m_hCam,
              IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE,
             (void*)dblRange,
             sizeof(dblRange)
            );

if (nRet == IS_SUCCESS)
{
    dblMin = dblRange[0];
    dblMax = dblRange[1];
    dblInc = dblRange[2];
}
```

 © 2015 Thorlabs Scientific Imaging

4.3.27 is_ForceTrigger

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ForceTrigger (HIDS hCam)
```

Description

You can use is_ForceTrigger() to force a software-controlled capture of an image while a capturing process triggered by hardware is in progress. This function can only be used if the triggered capturing process was started using the IS_DONT_WAIT parameter.

Input parameters

hCam	Camera handle

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_FreezeVideo()
- is_CaptureVideo()
- is_SetExternalTrigger()

Example

Enable trigger and wait 1 second for the external trigger. If no trigger signal has arrived, force an exception using `is_ForceTrigger()`.

```
HANDLE hEvent = CreateEvent(NULL, TRUE, FALSE, "");
if ( hEvent != NULL )
{
    is_InitEvent(hCam, m_hEvent, IS_SET_EVENT_FRAME);
    is_EnableEvent(hCam, IS_SET_EVENT_FRAME);

    is_SetExternalTrigger(hCam, IS_SET_TRIGGER_HI_LO);
    is_FreezeVideo(hCam, IS_DONT_WAIT);

    if (WaitForSingleObject(m_hEvent, 1000) != WAIT_OBJECT_0)
```

```
    {
        // No trigger has been received, so force image capture
        is_ForceTrigger(hCam);
    }

    is_DisableEvent(hCam, IS_SET_EVENT_FRAME);
    is_ExitEvent(hCam, IS_SET_EVENT_FRAME);
}
```


© 2015 Thorlabs Scientific Imaging

4.3.28 is_FreeImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_FreeImageMem (HIDS hCam, char* pcImgMem, INT id)
```

Description

`is_FreeImageMem()` releases an image memory that was allocated using `is_AllocImageMem()` and removes it from the driver management.

Note

If the memory was not allocated using an SDK function, you need to call `is_FreeImageMem()` as well. Otherwise, there may be errors when the driver keeps trying to access this memory.

This does however not release the memory. So you need to make sure that the memory will be released again.

Input parameters

`hCam`	Camera handle
`pcImgMem`	Points to the starting address of the memory (e.g. set in the `is_AllocImageMem()` function)
`id`	ID of this memory

Return values

`IS_CANT_ADD_TO_SEQUENCE`	The image memory is already included in the sequence and cannot be added again.
`IS_CANT_CLEANUP_MEMORY`	The driver could not release the allocated memory.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`

© 2015 Thorlabs Scientific Imaging

4.3.29 is_FreezeVideo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_FreezeVideo (HIDS hCam, INT Wait)
```

Description

`is_FreezeVideo()` acquires a single image from the camera. In DIB mode, the image is stored in the active image memory. If ring buffering is used in DIB mode, the captured image is transferred to the next available image m emory of the sequence. Once the last available sequence memory has been filled, the sequence event or message will be triggered.

In Direct3D or OpenGL mode, the is directly copied to the graphics card buffer and then displayed.

Image capture will be started by a trigger if you previously enabled the trigger mode using `is_SetExternalTrigger()`. A hardware triggered image acquisition can be cancelled using `is_StopLiveVideo()` if exposure has not started yet.

For further information on the image capture modes of the DCx camera, see the How to proceed: Image capture section.

Input parameters

`hCam`	Camera handle
⊟ `Wait`	
`IS_DONT_WAIT`	Timeout value for image capture (see also the How to proceed: Timeout values for image capture section)
`IS_WAIT`	
Time `t`	

Return values

`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_CAPTURE_RUNNING`	A capturing operation is in progress and must be terminated before you can start another one.
`IS_INVALID_BUFFER_SIZE`	The image memory has an inappropriate size to store the image in the desired format.
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_EXPOSURE_TIME`	This setting is not available for the currently set exposure time.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available

	in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the is_SetImageMem() function or create a sequence using the is_AddToSequence() function.
IS_NO_USB20	The camera is connected to a port which does not support the USB 2.0 high-speed standard.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_SUCCESS	Function executed successfully
IS_TRANSFER_ERROR	Transfer error. Frequent transfer errors can mostly be avoided by reducing the pixel rate.

Related functions

- is_HasVideoStarted()
- is_IsVideoFinish()
- is_SetExternalTrigger()
- is_ForceTrigger()
- is_CaptureVideo()
- is_SetTimeout()
- is_CaptureStatus()

Example

Enable trigger mode, set high-active flash mode and capture an image:

```
is_SetExternalTrigger(hCam, IS_SET_TRIGGER_SOFTWARE);

// Set the flash to a high active pulse for each image in the trigger mode
UINT nMode = IO_FLASH_MODE_TRIGGER_HI_ACTIVE;
is_IO(m_hCam, IS_IO_CMD_FLASH_SET_MODE, (void*)&nMode, sizeof(nMode));

is_FreezeVideo(hCam, IS_WAIT);
```

Sample programs

- SimpleAcquire (C++)
- uc480C# Demo (C#)

 © 2015 Thorlabs Scientific Imaging

4.3.30 is_Gamma

USB 2.0 USB 3.0 GigE	USB 2.0 USB 3.0 GigE

Syntax

```
INT is_Gamma(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

`is_Gamma()` enables digital gamma correction which applies a gamma characteristic to the image. When hardware color conversion is used on GigE uEye HE cameras the gamma correction is performed in the camera hardware as well. When the color conversion is performed in the PC (software conversion) the gamma correction is performed in software.

When the color format is set to Raw Bayer the gamma correction can not be used.

Typical values for gamma range between 1.6 and 2.2.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note: The following function is obsolete by the `is_Gamma()` function:

- `is_SetGamma()`

See also: Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_GAMMA_CMD_SET`	Gamma value to be set, multiplied by 100 (Range: 1…1000) Example 1
`IS_GAMMA_CMD_GET_DEFAULT`	Returns the standard gamma value (Default = 100, corresponds to a gamma value of 1.0) Example 2
`IS_GAMMA_CMD_GET`	Returns the current set gamma value Example 3
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetHardwareGamma()

Example 1

```
/* set gamma value to 1.6 */
INT nGamma = 160;
INT nRet = is_Gamma(hCam, IS_GAMMA_CMD_SET, (void*) &nGamma, sizeof(nGamma));
```

Example 2

```
/* set gamma to default value */
INT nGamma;

INT nRet = is_Gamma(hCam, IS_GAMMA_CMD_GET_DEFAULT, (void*)&nGamma, sizeof(nGamma));
if (nRet == IS_SUCCESS)
{
  is_Gamma(hCam, IS_GAMMA_CMD_SET, (void*)&nGamma, sizeof(nGamma));
}
```

Beispiel 3

```
/* read gamma value to nGamma */
INT nGamma;
INT nRet = is_Gamma(hCam, IS_GAMMA_CMD_GET, (void*) &nGamma, sizeof(nGamma));
```

See also:
- Basics: Characteristics and LUT
- Basics: Color filter (Bayer filter)
- Programming: is_SetColorConverter()

4.3.31 is_GetActiveImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetActiveImageMem (HIDS hCam, char** ppcMem, INT* pnID)
```

Description

is_GetActiveImageMem() returns the pointer to the starting address and the ID number of the active image memory.

If a Direct3D mode is active and image memory was nevertheless allocated, the pointer to the image memory and its ID will be returned. However, in Direct3D mode, the image will not be copied automatically to this image memory.

 © 2015 Thorlabs Scientific Imaging

Input parameters

hCam	Camera handle
ppcMem	Returns the pointer to the starting address of the active image memory.
pnID	Returns the ID of the active image memory.

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_AllocImageMem()
- is_GetImageMem()
- is_SetImageMem()
- is_SetAllocatedImageMem()

4.3.32 is_GetActSeqBuf

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetActSeqBuf (HIDS hCam, INT* pnNum,
                     char** ppcMem, char** ppcMemLast);
```

Description

Using `is_GetActSeqBuf()`, you can determine the image memory which is currently used for capturing an image (`ppcMem`) or the image memory that was last used for capturing an image (`ppcMemLast`). This function is only available if you have enabled ring buffering.

Attention

All input parameters of a function have to be initialized with valid values before the function is called; this also applies to parameters that are not used. Variables can be preset with '0', for example. For unused parameters, the `NULL` pointer has to be passed.

Note

This number is not the ID of the image memory that was allocated using the `is_AllocImageMem()` function, but the running number from the order in which memory was allocated by the `is_AddToSequence()` function.

Input parameters

`hCam`	Camera handle
`pnNum`	Contains the number of the image memory currently used for image capturing. If image capturing is already in progress when `is_GetActSeqBuf()` is called, `pnNum` will return the value 0 until the sequence arrives at the first image memory again.
`ppcMem`	Contains the starting address of the image memory currently used for image capturing.
`ppcMemLast`	Contains the starting address of the image memory last used for image capturing.

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed.

© 2015 Thorlabs Scientific Imaging

	Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SEQUENCE_LIST_EMPTY`	The sequence list is empty and cannot be deleted.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AddToSequence()`
- `is_GetImageMem()`

4.3.33 is_GetAutoInfo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetAutoInfo (HIDS hCam, UC480_AUTO_INFO* pInfo)
```

Description

Using the `is_GetAutoInfo()` function, you can query status information on the automatic image control features. This information is written to the `UC480_AUTO_INFO` structure.

For further information on automatic control, please refer to the Automatic image control chapter.

Attention

The status information returned in the `UC480_AUTO_INFO` structure is only valid if at least one of the auto control feature has been enabled using `is_SetAutoParameter()`.

Input parameters

`hCam`	Camera handle
`pinfo`	`UC480_AUTO_INFO` structure (see below)

Contents of the UC480_AUTO_INFO Structure

`INT`	`AutoAbility`	Supported auto control features
	`AC_SHUTTER`	Auto exposure shutter is supported
	`AC_SENSOR_SHUTTER`	The sensor's internal auto exposure shutter is supported
	`AC_FRAMERATE`	Auto frame rate is supported
	`AC_SENSOR_FRAMERATE`	The sensor's internal auto frame rate is supported
	`AC_GAIN`	Auto gain control is supported
	`AC_SENSOR_GAIN`	The sensor's internal auto gain control is supported
	`AC_SENSOR_AUTO_CONTRAST_CORRECTION`	Auto contrast correction for automatic brightness control is supported
	`AC_SENSOR_AUTO_CONTRAST_FDT_AOI`	Use of face detection as field of view for automatic brightness control is supported
	`AC_SENSOR_AUTO_BACKLIGHT_COMP`	Backlight compensation for automatic brightness control is supported
	`AC_WHITEBAL`	Auto white balance is supported
	`AC_SENSOR_WB`	The sensor's internal auto white balance is supported
`AUTO_BRIGHT_STATUS`	`sBrightCtrlStatus`	Status of automatic brightness control, see below
`AUTO_WB_STATUS`	`sWBCtrlStatus`	Status of auto white balance, see below
`DWORD`	`AShutterPhotomCaps`	Returns a bit mask containing all supported

© 2015 Thorlabs Scientific Imaging

		photometry settings (fields of view) for auto exposure shutter, see below.
DWORD	AGainPhotomCaps	Returns a bit mask containing all supported photometry settings (fields of view) for auto gain control, see below.
DWORD	AAntiFlickerCaps	Returns a bit mask containing all supported anti flicker settings for automatic control, see below.
DWORD	SensorWBModeCaps	Returns a bit mask containing all supported settings for the sensor's auto white balance, see below.
DWORD	reserved[8]	Reserved space for extensions

Contents of the UC480_AUTO_INFO::AUTO_BRIGHT_STATUS Structure

INT	curValue	Current average brightness of the image (actual value); the following rule applies independently of the image bit depth: 0 = black 255 = white
INT	curError	Current control deviation (error)
INT	curController	Current parameter value
	AC_SHUTTER	Exposure time (shutter)
	AC_GAIN	Gain
INT	curCtrlStatus	Current control status
	ACS_ADJUSTING	Control is active.
	ACS_FINISHED	Control is completed.
	ACS_DISABLED	Control is disabled.

Contents of the UC480_AUTO_INFO::AUTO_WB_STATUS Structure

INT	curController	Current white balance control
	AC_WB_RED_CHANNEL	Value of the red channel
	AC_WB_GREEN_CHANNEL	Value of the green channel
	AC_WB_BLUE_CHANNEL	Value of the blue channel
AUTO_WB_CHANNEL_STATUS	RedChannel	See AUTO_WB_CHANNEL_STATUS
AUTO_WB_CHANNEL_STATUS	GreenChannel	See AUTO_WB_CHANNEL_STATUS
AUTO_WB_CHANNEL_STATUS	BlueChannel	See AUTO_WB_CHANNEL_STATUS

Contents of the UC480_AUTO_INFO::AUTO_WB_STATUS::AUTO_WB_CHANNEL_STATUS Structure

INT	curValue	Current average grayscale value (actual value)
INT	curError	Current control deviation (error)
INT	curCtrlStatus	Current control status
	ACS_ADJUSTING	Control is active.
	ACS_FINISHED	Control is completed.

	ACS_DISABLED	Control is disabled.

Status Flags in UC480_AUTO_INFO::AShutterPhotomCaps

AS_PM_NONE	The entire field of view is used for metering.
AS_PM_SENS_CENTER_WEIGHTED	Metering is based on the entire field of view, but gives greater emphasis to the center area of the image.
AS_PM_SENS_CENTER_SPOT	Only a small area in the image center is used for metering.
AS_PM_SENS_PORTRAIT	Metering is based on that part of the field of view that corresponds to the portrait format.
AS_PM_SENS_LANDSCAPE	Metering is based on that part of the field of view that corresponds to the landscape format.

Status Flags in UC480_AUTO_INFO::AGainPhotomCaps

AG_PM_NONE	The entire field of view is used for metering.
AG_PM_SENS_CENTER_WEIGHTED	Metering is based on the entire field of view, but gives greater emphasis to the center area of the image.
AG_PM_SENS_CENTER_SPOT	Only a small area in the image center is used for metering.
AG_PM_SENS_PORTRAIT	Metering is based on that part of the field of view that corresponds to the portrait format.
AG_PM_SENS_LANDSCAPE	Metering is based on that part of the field of view that corresponds to the landscape format.

Status Flags in UC480_AUTO_INFO::AAntiFlickerCaps

ANTIFLCK_MODE_OFF	Anti flicker function disabled.
ANTIFLCK_MODE_SENS_AUTO	The anti flicker mode is selected automatically (50 or 60 Hz).
ANTIFLCK_MODE_SENS_50_FIXED	The anti flicker mode is set to a fixed value of 50 Hz.
ANTIFLCK_MODE_SENS_60_FIXED	The anti flicker mode is set to a fixed value of 60 Hz.

Status Flags in UC480_AUTO_INFO::SensorWBModeCaps

WB_MODE_DISABLE	Disables the sensor's auto white balance
WB_MODE_AUTO	Sensor automatically determines auto white balance
WB_MODE_ALL_PULLIN	Sensor automatically determines auto white balance using the Gray World algorithm. This algorithm assumes that the average color value in the scene is gray.
WB_MODE_INCANDESCENT_LAMP	Sensor sets auto white balance to incandescent light
WB_MODE_FLUORESCENT_DL	Sensor sets auto white balance to fluorescent

© 2015 Thorlabs Scientific Imaging

	light (daylight type)
WB_MODE_OUTDOOR_CLEAR_SKY	Sensor sets auto white balance to direct daylight
WB_MODE_OUTDOOR_CLOUDY	Sensor sets auto white balance to cloudy sky

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- `is_SetAutoParameter()`

4.3.34 is_GetBusSpeed

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetBusSpeed (HIDS hCam)
```

Description

Using `is_GetBusSpeed()`, you can query whether a camera is connected to a USB 2.0 or USB 3.0 host controller. You can see in the uc480 Camera Manager below "General Information" which kind of USB host controller are available on your PC.

Input parameters

`hCam`	Camera handle

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully
`IS_USB_10`	The controller to which the camera is connected does not support USB 2.0.
`IS_USB_20`	The camera is connected to a USB 2.0 controller.
`IS_USB_30`	The camera is connected to a USB 3.0 controller.

© 2015 Thorlabs Scientific Imaging

4.3.35 is_GetCameraInfo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetCameraInfo (HIDS hCam, CAMINFO* pInfo)
```

Description

`is_GetCameraInfo()` reads out the data hard-coded in the EEPROM and writes it to the data structure that `pInfo` points to.

Note

The serial number or model name should not be used to find a specific camera (e.g. in order to control this specific camera). If you use the serial number, the software may not find the serial number after exchanging the camera. The model name can be changed when updating the camera driver.

Instead, we recommend identifying a camera by a fixed camera ID, the camera type or by the sensor ID (see `is_GetCameraList()`). The advantage of the camera ID is that you can set it manually. That means if you exchange a camera, you can set the same camera ID for the new camera.

Attention

For technical reasons, the following values for `CAMINFO::Type` are internally redirected to the same value:

`IS_CAMERA_TYPE_UC480_USB_SE` and `IS_CAMERA_TYPE_UC480_USB_RE`

You can use the parameter strSensorName of the `is_GetSensorInfo()` function to discern the camera models DCU223x, DCU224x and DCC1240x.

Input parameters

`hCam`	Camera handle
`pInfo`	Pointer to a `CAMINFO` data structure

Contents of the CAMINFO Structure

`char`	`SerNo[12]`	Serial number of the camera
`char`	`ID[20]`	Manufacturer of the camera
`char`	`Version[10]`	For USB cameras, this value indicates the USB board hardware version (e.g. `V2.10`)
`char`	`Date[12]`	System date of the final quality check (e.g. `01.08.2011` (DD.MM.YYYY))
`unsigned char`	`Select`	Camera ID
`unsigned char`	`Type`	Camera type: • `IS_CAMERA_TYPE_UEYE_USB_SE`: DCU223x, DCU224x and DCC1240x • `IS_CAMERA_TYPE_UEYE_USB_LE`: DCC1545M / DCC1645C • `IS_CAMERA_TYPE_UEYE_USB3_CP`: DCC3240x

char	Reserved[8]	Reserved

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_CameraStatus()
- is_GetSensorInfo()

© 2015 Thorlabs Scientific Imaging

4.3.36 is_GetCameraList

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetCameraList (UC480_CAMERA_LIST* pucl)
```

Description

Using `is_GetCameraList()`, you can query information about the connected cameras. To get all information that is available, you need to adjust the field size to the number of connected cameras. The following tables explain the structures used for that purpose.

Input parameters

`pucl`	Handle to the `UC480_CAMERA_LIST` structure

Contents of the UC480_CAMERA_LIST Structure

`ULONG`	`dwCount`	Has to be initialized with the number of cameras connected to the system. This value can be read out with `is_GetNumberOfCameras()`.
`UC480_CAMERA_INFO`	`uci[1]`	Placeholder for 1 … n `UC480_CAMERA_INFO` structures

Contents of the UC480_CAMERA_LIST::UC480_CAMERA_INFO Structure

`DWORD`	`dwCameraID`	Customizable camera ID. This ID is stored in the camera and is persistent.
`DWORD`	`dwDeviceID`	Internal device ID. This ID is generated by the driver depending on order of connection and camera type. The device ID is not persistent.
`DWORD`	`dwSensorID`	Sensor ID
`DWORD`	`dwInUse`	1 = camera is being used. 0 = camera is not being used.
`Char`	`SerNo[16]`	Serial number of the camera
`Char`	`Model[16]`	Camera model
`DWORD`	`dwStatus`	Information for the status of the camera
`DWORD`	`dwReserved[15]`	Reserved for later use

Note

The serial number or model name should not be used to find a specific camera (e.g. in order to control this specific camera). If you use the serial number, the software may not find the serial number after exchanging the camera. The model name can be changed when updating the camera driver.

Instead, we recommend identifying a camera by a fixed camera ID, the camera type or by the sensor ID. The advantage of the camera ID is that you can set it manually. That means if you exchange a camera, you can set the same camera ID for the new camera.

Return values

IS_ACCESS_VIOLATION	Not enough memory allocated for the UC480_CAMERA_LIST structure
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_GetNumberOfCameras()

Example

```
// At least one camera must be available
INT nNumCam;
if( is_GetNumberOfCameras( &nNumCam ) == IS_SUCCESS) {
  if( nNumCam >= 1 ) {
    // Create new list with suitable size
    UC480_CAMERA_LIST* pucl;
    pucl = (UC480_CAMERA_LIST*) new BYTE [sizeof (DWORD) + nNumCam * sizeof (UC480_CAMERA_INFO)];
    pucl->dwCount = nNumCam;

    //Retrieve camera info
    if (is_GetCameraList(pucl) == IS_SUCCESS) {
      int iCamera;
      for (iCamera = 0; iCamera < (int)pucl->dwCount; iCamera++) {
        //Test output of camera info on the screen
        printf("Camera %i Id: %d", iCamera,
        pucl->uci[iCamera].dwCameraID);
      }
    }
  }
  delete [] pucl;
}
```

 © 2015 Thorlabs Scientific Imaging

4.3.37 is_GetCameraLUT

USB 3.0	USB 3.0

Syntax

```
INT is_GetCameraLUT (HIDS hCam,
                     UINT Mode, UINT NumberOfEntries,
                     double* pRed_Grey,
                     double* pGreen,
                     double* pBlue)
```

Description

`is_GetCameraLUT()` returns the current LUT values. Using the `is_SetCameraLUT()` function, you can select a different LUT for the camera.

Note

The `is_SetCameraLUT()` function is only supported by DCC3240 cameras.

Input parameters

hCam	Camera handle
⊟ Mode	
IS_GET_CAMERA_LUT_USER	Returns the LUT values set by the user without modifications.
IS_GET_CAMERA_LUT_COMPLETE	Returns the LUT values set by the user after the gamma, contrast and brightness values have been taken into account.
NumberOfEntries	Number of the LUT values
IS_CAMERA_LUT_64	LUT with 64 values
pRed_Grey	Pointer to the array to which the red channel values or the gray scale value of the LUT are written.
pGreen	Pointer to the array to which the green channel values of the LUT are written.
pBlue	Pointer to the array to which the blue channel values of the LUT are written.

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

4.3.38 is_GetColorConverter

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetColorConverter (HIDS hCam,
                          INT ColorMode,
                          INT* pCurrentConvertMode,
                          INT* pDefaultConvertMode,
                          INT* pSupportedConvertModes)
```

Description

For color cameras, `is_GetColorConverter()` returns the set mode or all available Bayer conversion modes for the specified color mode. The return value depends on the selected color mode. For further information, please refer to the Appendix: color and memory formats section.

Input parameters

`hCam`	Camera handle
`ColorMode`	Color mode for which the converter is to be returned For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats section.
`pCurrentConvertMode`	Currently selected converter for this color mode
`pDefaultConvertMode`	Default converter for this color mode

`pSupportedConvertModes`	All converters supported for this color mode. Possible converters are:
	`IS_CONV_MODE_NONE`
	`IS_CONV_MODE_SOFTWARE_3X3`
	`IS_CONV_MODE_SOFTWARE_5X5`
	`IS_CONV_MODE_HARDWARE_3X3`

Return values

`IS_INVALID_COLOR_FORMAT`	Invalid color format
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetColorConverter()`
- `is_SetColorMode()`

© 2015 Thorlabs Scientific Imaging

4.3.39 is_GetColorDepth

USB 2.0 USB 3.0	-

Syntax

```
INT is_GetColorDepth(HIDS hCam, INT* pnCol, INT* pnColMode)
```

Description

`is_GetColorDepth()` retrieves the current Windows Desktop color setting and returns the bit depth per pixel and the matching uc480 color mode. The color mode can be passed directly to the `is_SetColorMode()` function. You need to pass the bit depth when allocating an image memory.

Input parameters

`hCam`	Camera handle
`pnCol`	Returns the bit depth of the color setting.
`pnColMode`	Returns the uc480 color mode that corresponds to `pnCol`. For a list of all available color formats and the associated input parameters, see the Appendix: color and Memory Formats section.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetColorMode()`
- `is_AllocImageMem()`

4.3.40 is_GetDLLVersion

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetDLLVersion()
```

Description

Returns the version of the `uc480.dll`.

Input parameters

<none>

Return values

The return value contains the version number which is coded as follows:

Bits 31-24:	Major version
Bits 23-16:	Minor version
Bits 15-0:	Build version

Related functions

- `is_GetOsVersion()`

Example

```
int version = is_GetDLLVersion();
int build = version & 0xFFFF;
version = version >> 16;
int minor = version & 0xFF;
version = version >> 8;
int major = version & 0xFF;
printf("API version %d.%d.%d \n\n", major, minor, build);
```


© 2015 Thorlabs Scientific Imaging

4.3.41 is_GetError

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetError (HIDS hCam, INT* pErr, IS_CHAR**  ppcErr)
```

Description

`is_GetError()` queries the last error that occurred and returns the associated error code and message. We recommend to use this function after an error has occurred that returned `IS_NO_SUCCESS`. Each error message will be overwritten when a new error occurs.

Input parameters

`hCam`	Camera handle
`PErr`	Pointer to the variable containing the error code
`PpcErr`	Pointer to the string containing the error text

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_CaptureStatus()`
- `is_SetErrorReport()`
- `is_CameraStatus()`

4.3.42 is_GetFramesPerSecond

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetFramesPerSecond (HIDS hCam, double* dblFPS)
```

Description

In live capture mode started by `is_CaptureVideo()`, the `is_GetFramesPerSecond()` function returns the number of frames actually captured per second.

Input parameters

`hCam`	Camera handle
`dblFPS`	Returns the current frame rate.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetFrameTimeRange()`
- `is_SetFrameRate()`
- `is_PixelClock()`
- `is_Exposure()`

© 2015 Thorlabs Scientific Imaging

4.3.43 is_GetFrameTimeRange

USB 2.0	USB 2.0

Syntax

```
INT is_GetFrameTimeRange (HIDS hCam,
                          double* min, double* max, double* intervall)
```

Description

Using `is_GetFrameTimeRange()`, you can read out the frame rate settings which are available for the current pixel clock setting. The returned values indicate the minimum and maximum frame duration in seconds. You can set the frame duration between `min` and `max` in increments defined by the `intervall` parameter.

The following applies:

$$fps_{min} = 1/_{max}$$

$$fps_{max} = 1/_{min}$$

$$fps_{n} = 1/_{(min + n*intervall)}$$

The call of this function makes only sense in the freerun mode.

Note

The use of the following functions will affect the frame duration:

- `is_PixelClock()`
- `is_SetOptimalCameraTiming()`
- `is_AOI()` (if the image size is changed)
- `is_SetSubSampling()`
- `is_SetBinning()`

Changes made to the window size, the frame rate or the read-out timing (pixel clock frequency) also affect the defined frame duration. For this reason, you need to call `is_GetFrameTimeRange()` again after such changes.

Input parameters

`hCam`	Camera handle
`min`	Returns the minimum available frame duration.
`max`	Returns the maximum available frame duration.
`intervall`	Returns the increment you can use to change the frame duration.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetFramesPerSecond()`
- `is_SetFrameRate()`
- `is_PixelClock()`
- `is_Exposure()`

© 2015 Thorlabs Scientific Imaging

4.3.44 is_GetImageHistogram

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetImageHistogram (HIDS hCam,
                          int nID, INT ColorMode, DWORD* pHistoMem)
```

Description

`is_GetImageHistogram()` computes the histogram of the submitted image. The histogram always contains 256 values per channel. For color modes with a bit depth of more than 8 bits, the system evaluates the 8 most significant bits (MSBs).

Input parameters

`hCam`	Camera handle
`nID`	Memory ID
`ColorMode`	Color mode of the image with the `nID` memory ID For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats section.
`pHistoMem`	Pointer to a `DWORD` array The array must be allocated in such a way that it can accommodate 3*256 values for color formats and in raw Bayer mode. In monochrome mode, the array must be able to accommodate 1*256 values.

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_COLOR_FORMAT`	Invalid color format
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NULL_POINTER`	Invalid array
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetColorMode()`

© 2015 Thorlabs Scientific Imaging

Example

```
char * pcSource;
INT nIDSource;
is_AllocImageMem (hCam, 256, 256, 24, &pcSource, &nIDSource);

int nX, nY, nBits, nPitch;
is_InquireImageMem (hCam, pcSource, nIDSource, &nX ,&nY, &nBits, &nPitch);

//Create RGB test image
for (int j = 0; j < nY; j++)
{
    for (int i = 0; i < nX*3; i += 3)
    {
        pcSource[i + j*nPitch] = 0; // Blue pixels
        pcSource[i + j*nPitch + 1] = i/3; // Green pixels
        pcSource[i + j*nPitch + 2] = 255; // Red pixels
    }
}

// Create memory for RGB histogram
DWORD bgrBuffer [256*3];

//Create pointer for each histogram color
DWORD * pBlueHisto  = bgrBuffer;
DWORD *pGreenHisto = bgrBuffer + 256;
DWORD * pRedHisto = bgrBuffer + 512;

//Retrieve histogram and release memory
is_GetImageHistogram (hCam, nIDSource, IS_CM_RGB8_PACKED, bgrBuffer);
is_FreeImageMem (hCam, pcSource, nIDSource);
```

4.3.45 is_GetImageInfo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetImageInfo (HIDS hCam, INT nImageBufferID, UEYEIMAGEINFO* pImageInfo, INT nImageInfoSize)
```


- Description
- Input parameters
- Contents of the UEYEIMAGEINFO structure
- Status flags in UEYEIMAGEINFO::dwIoStatus
- Contents of the UEYETIME structure
- Return values
- Related functions
- Code sample

Description

`is_GetImageInfo()` provides additional information on the images you take. The function returns a timestamp indicating the time of image capture, and the states of the camera I/Os at that point in time. To get information on the last image that was taken, call `is_GetImageInfo()` directly after receiving the `IS_FRAME` event.

Using the function with USB DCx Cameras

The `u64TimestampDevice` timestamp returns the time when image data transfer to the PC was completed.

The `UEYETIME` structure returns the timestamp (with a resolution of 1 ms) synchronized with the PC system time.

Attention

Image buffers that are part of a sequence need to be locked using `is_LockSeqBuf()`. This is important to ensure correct assignment between image data and image information. Otherwise, it may happen that an image buffer is filled with new image data. In this case, the image information will not match the image data any more.

Input parameters

`hCam`	Camera handle
`nImageBufferID`	ID of the image buffer for which information is requested
`pImageInfo`	Pointer to a `UC480IMAGEINFO` type structure to which the information will be written
`nImageInfoSize`	Size of the structure

Contents of the UC480IMAGEINFO structure

`DWORD`	`dwFlags`	Internal status flags (currently not used)

© 2015 Thorlabs Scientific Imaging

BYTE	byReserved1[4]	Reserved
unsigned long long	u64TimestampDevice	Internal timestamp of image capture (tick count of the camera in 0.1 µs steps)
UEYETIME	TimestampSystem	Structure with timestamp information in PC system time format, see UEYETIME below
DWORD	dwIoStatus	With DCC3240x Cameras: Returns the states of the digital I/Os at the time of image capture: • Digital input (trigger): Pending signal • GPIO as input: Pending signal • GPIO as output: Set level With all other cameras, dwIoStatus is empty. See dwIOStatus below.
WORD	wAOIIndex	AOI index (only AOI sequence mode of DCC1240x)
WORD	wAOICycle	Readout cycles (only AOI sequence mode of DCC1240x)
unsigned long long	u64FrameNumber	Internal image number
DWORD	dwImageBuffers	Number of image buffers existing in the camera
DWORD	dwImageBuffersInUse	Number of image buffers in use in the camera
DWORD	dwReserved3	Reserved
DWORD	dwImageHeight	Image height
DWORD	dwImageWidth	Image width

Status flags in UC480IMAGEINFO::dwIoStatus

Bit combination	State of digital input	State of GPIO 1	State of GPIO 2
000	0	0	0
001	0	0	1
010	0	1	0
011	0	1	1
100	1	0	0
101	1	0	1
110	1	1	0
111	1	1	1

Contents of the UC480IMAGEINFO::UEYETIME structure

WORD	wYear	Timestamp year
WORD	wMonth	Timestamp month
WORD	wDay	Timestamp day
WORD	wHour	Timestamp hour
WORD	wMinute	Timestamp minute
WORD	wSecond	Timestamp second
WORD	wMilliseconds	Timestamp millisecond
WORD	wReserved[2]	Reserved

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode. This may happen when e.g.: • more memory is allocated than the UC480IMAGEINFO structure needs • nImageBufferID <= 0 • pImageInfo == NULL • nImageInfoSize <= 0
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

© 2015 Thorlabs Scientific Imaging

Related functions

- is_CaptureStatus()
- is_LockSeqBuf()
- is_UnlockSeqBuf()
- is_SetImageMem()
- is_IO()

Example

```
UEYEIMAGEINFO ImageInfo;
// Read out camera timestamp
INT nRet = is_GetImageInfo( m_hCam,
           m_lMemoryId, &ImageInfo, sizeof(ImageInfo));
if (nRet == IS_SUCCESS)
{
unsigned long long u64TimestampDevice;
   u64TimestampDevice = ImageInfo.u64TimestampDevice;

CString Str; // Read out timestamp in system time
Str.Format("%02d.%02d.%04d, %02d:%02d:%02d:%03d",
          ImageInfo.TimestampSystem.wDay,
          ImageInfo.TimestampSystem.wMonth,
          ImageInfo.TimestampSystem.wYear,
          ImageInfo.TimestampSystem.wHour,
          ImageInfo.TimestampSystem.wMinute,
          ImageInfo.TimestampSystem.wSecond,
          ImageInfo.TimestampSystem.wMilliseconds);

DWORD dwTotalBuffers = ImageInfo.dwImageBuffers;
DWORD dwUsedBuffers = ImageInfo.dwImageBuffersInUse;
}
```

Sample Program

- uc480Timestamp (C++)

4.3.46 is_GetImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetImageMem (HIDS hCam, VOID** pMem)
```

Description

`is_GetImageMem()` returns the pointer to the starting address of the active image memory. If you use ring buffering, `is_GetImageMem()` returns the starting address of the image memory last used for image capturing.

Input parameters

`hCam`	Camera handle
`pMem`	Pointer to the starting address of the image memory

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetImageMemPitch()`
- `is_AllocImageMem()`
- `is_AddToSequence()`
- `is_SetImageMem()`
- `is_SetAllocatedImageMem()`

Sample programs

- uc480PixelPeek (C++)

© 2015 Thorlabs Scientific Imaging

4.3.47 is_GetImageMemPitch

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetImageMemPitch (HIDS hCam, INT* pPitch)
```

Description

`is_GetImageMemPitch()` returns the line increment (in bytes). The line increment is defined as the number of bytes from the beginning of a line to the beginning of the next line. It may be greater than suggested by the parameters passed when calling `is_AllocImageMem()`. The line increment is always a number that can be divided by 4.

The line increment is calculated as:

`line = width * int[(bitspixel + 7) / 8]`

`lineinc = line + adjust`

`adjust = 0` – if `line` can be divided by 4 without remainder

`adjust = 4 - rest(line / 4)` if line cannot be divided by 4 without remainder

Input parameters

`hCam`	Camera handle
`pPitch`	Pointer to the variable containing the line increment

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetImageMem()`
- `is_AllocImageMem()`
- `is_AddToSequence()`

- `is_SetImageMem()`
- `is_SetAllocatedImageMem()`

© 2015 Thorlabs Scientific Imaging

4.3.48 is_GetNumberOfCameras

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetNumberOfCameras (INT* pnNumCams)
```

Description

`is_GetNumberOfCameras()` returns the number of DCx Cameras connected to the PC.

Input parameters

pNumCams	Returns the number of connected cameras.

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_SUCCESS	Function executed successfully

Related functions

- is_GetCameraList()
- is_DeviceInfo()

4.3.49 is_GetOsVersion

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetOsVersion ()
```

Description

`is_GetOsVersion()` returns the operating system type used at runtime.

Input parameters

<none>

Return values

IS_OS_WIN2000	Windows 2000 operating system
IS_OS_WINXP	Windows XP operating system
IS_OS_WINSERVER2003	Windows Server 2003 operating system
IS_OS_WINVISTA	Windows Vista operating system
IS_OS_WIN7	Windows 7 operating system
IS_OS_LINUX26	Linux 2.6 operating system
IS_OS_UNDETERMINED	Unknown operating system

Related functions

- is_GetDLLVersion()

© 2015 Thorlabs Scientific Imaging

4.3.50 is_GetSensorInfo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetSensorInfo (HIDS hCam, SENSORINFO* pInfo)
```

Description

Using `is_GetSensorInfo()`, you can query information about the sensor type used in the camera. The information contained in the `SENSORINFO` structure is listed in the table below.

The `ueye.h` file provides a complete up-to-date list of all supported sensor types. To quickly locate the list, search the file for the keyword "Sensor types".

Input parameters

`hCam`	Camera handle
`pInfo`	Pointer to the `SENSORINFO` Structure

Contents of the SENSORINFO structure

`WORD`	`SensorID`	Returns the sensor type (e.g.: `IS_SENSOR_xxxxx_x`).
`Char`	`strSensorName[32]`	Returns the camera model (e.g.: `xxxxxxx-x`).

`Char`	`nColorMode`	Returns the sensor color mode.
		`IS_COLORMODE_BAYER`
		`IS_COLORMODE_MONOCHROME`

`DWORD`	`nMaxWidth`	Returns the maximum image width
`DWORD`	`nMaxHeight`	Returns the maximum image height
`BOOL`	`bMasterGain`	Indicates whether the sensor provides analog master gain
`BOOL`	`bRGain`	Indicates whether the sensor provides analog red channel gain
`BOOL`	`bGGain`	Indicates whether the sensor provides analog green channel gain
`BOOL`	`bBGain`	Indicates whether the sensor provides analog blue channel gain
`BOOL`	`bGlobShutter`	Indicates whether the sensor has a global shutter. `TRUE` = global shutter `FALSE` = rolling shutter
`WORD`	`wPixelSize`	Returns the pixel size in µm (e.g. 465 is equivalent to 4.65 µm)
`Char`	`Reserved[14]`	Reserved

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or

	is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_GetCameraInfo()
- is_CameraStatus()

© 2015 Thorlabs Scientific Imaging

4.3.51 is_GetSensorScalerInfo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetSensorScalerInfo (HIDS hCam,
                            SENSORSCALERINFO* pSensorScalerInfo,
                            INT nSensorScalerInfoSize)
```

Description

Using `is_GetSensorScalerInfo()` you can query information on the image scaling which is supported by some sensors.

Note

Internal image scaling is only supported by DCC1240x and DCC3240x series sensors.

Input parameters

`hCam`	Camera handle
`pSensorScalerInfo`	Pointer to a `SENSORSCALERINFO` type structure to which the information will be written
`nSensorScalerInfoSize`	Size of the structure

Contents of the SENSORSCALERINFO structure

`INT`	`nCurrMode`	Returns the current mode
`INT`	`nNumberOfSteps`	Returns the number of steps for the scaling factor
`double`	`dblFactorIncrement`	Returns the increment for the scaling factor
`double`	`dblMinFactor`	Returns the minimum scaling factor
`double`	`dblMaxFactor`	Returns the maximum scaling factor
`double`	`dblCurrFactor`	Returns the current scaling factor
`INT`	`nSupportedModes`	Returns the supported function modes, see `is_SetSensorScaler()`
`BYTE`	`bReserved[84]`	Reserved

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetSensorScaler()`

4.3.52 is_GetSupportedTestImages

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetSupportedTestImages (HIDS hCam, INT* SupportedTestImages)
```

Description

`is_GetSupportedTestImages()` returns all test images supported by the camera. You can enable the sensor test image feature using `is_SetSensorTestImage()`.

Input parameters

`hCam`	Camera handle
`SupportedTestImages`	Returns a bit mask of all test images supported by the camera.
`IS_TEST_IMAGE_NONE`	No test image
`IS_TEST_IMAGE_WHITE`	White image
`IS_TEST_IMAGE_BLACK`	Black image
`IS_TEST_IMAGE_HORIZONTAL_GREYSCALE`	Horizontal grayscale
`IS_TEST_IMAGE_VERTICAL_GREYSCALE`	Vertical grayscale
`IS_TEST_IMAGE_DIAGONAL_GREYSCALE`	Diagonal grayscale
`IS_TEST_IMAGE_WEDGE_GRAY_SENSOR`	Gray wedges, generated by the sensor
`IS_TEST_IMAGE_WEDGE_COLOR`	Color wedges
`IS_TEST_IMAGE_ANIMATED_WEDGE_GRAY_SENSOR`	Gray wedges, animated, generated by the sensor
`IS_TEST_IMAGE_ANIMATED_WEDGE_COLOR`	Color wedges, animated
`IS_TEST_IMAGE_COLOR_BARS1`	Color bars
`IS_TEST_IMAGE_GREY_AND_COLOR_BARS`	Gray and color bars
`IS_TEST_IMAGE_MOVING_GREY_AND_COLOR_BARS`	Gray and color bars, animated
`IS_TEST_IMAGE_ANIMATED_LINE`	Line, animated
`IS_TEST_IMAGE_ALTERNATE_PATTERN`	Alternating pattern (raw Bayer mode only)
`IS_TEST_IMAGE_RAMPING_PATTERN`	Diagonal color pattern
`IS_TEST_IMAGE_MONOCHROME_HORIZONTAL_BARS`	Monochrome bars, horizontal
`IS_TEST_IMAGE_MONOCHROME_VERTICAL_BARS`	Monochrome bars, vertical
`IS_TEST_IMAGE_COLDPIXEL_GRID`	Camera image overlaid with a grid of blue dots
`IS_TEST_IMAGE_HOTPIXEL_GRID`	Camera image overlaid with a grid of red dots
`IS_TEST_IMAGE_VARIABLE_GREY`	Adjustable grayscale image
`IS_TEST_IMAGE_VARIABLE_RED_PART`	Image with adjustable red content
`IS_TEST_IMAGE_VARIABLE_GREEN_PART`	Image with adjustable green content
`IS_TEST_IMAGE_VARIABLE_BLUE_PART`	Image with adjustable blue content

© 2015 Thorlabs Scientific Imaging

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetSensorTestImage()
- is_GetTestImageValueRange()

4.3.53 is_GetTestImageValueRange

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetTestImageValueRange (HIDS hCam,
                               INT TestImage,
                               INT* TestImageValueMin,
                               INT* TestImageValueMax)
```

Description

Using `is_GetTestImageValueRange()`, you can query the value range of the additional parameter required for some camera test images. You can enable the sensor test image feature using `is_SetSensorTestImage()`.

Input parameters

`hCam`	Camera handle
`TestImage`	Test image for which the value range is queried
`TestImageValueMin`	Minimum value
`TestImageValueMax`	Maximum value

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting. In this case, the `TestImageValueMin` and `TestImageValueMax` parameters are equal to 0.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetSupportedTestImages()`
- `is_SetSensorTestImage()`

© 2015 Thorlabs Scientific Imaging

4.3.54 is_GetTimeout

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetTimeout (HIDS hCam, UINT nMode, UINT* pTimeout)
```

Description

Using `is_GetTimeout()`, you can read out user-defined timeout values from the uc480 API.

For further information, please refer to the How to proceed: Timeout values for image capture section.

Input parameters

`hCam`	Camera handle
`nMode`	Selects the timeout value to be returned
`IS_TRIGGER_TIMEOUT`	Returns the timeout value in steps of 10 ms for triggered image capture
`pTimeout`	Pointer to the variable that holds the timeout value. Returns 0 if the default value of the uc480 API is used.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetTimeout()`
- `is_CaptureVideo()`
- `is_FreezeVideo()`
- `is_SetExternalTrigger()`

Example

```
// Return user-defined timeout
UINT nTimeout;
INT ret = is_GetTimeout(hCam, IS_TRIGGER_TIMEOUT, &nTimeout);
```

4.3.55 is_GetUsedBandwidth

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetUsedBandwidth (HIDS hCam)
```

Description

`is_GetUsedBandwidth()` returns the bus bandwidth (in MByte/s) currently used by all initialized or selected cameras. This is an approximate value which is calculated based on the pixel clock that has been set and the data format (bits per pixel). The actual data load on the bus can slightly deviate from this value.

Input parameters

`hCam`	Camera handle

Return values

`INT` value	The total current bus bandwidth (in MByte/s)

Related functions

- `is_PixelClock()`

© 2015 Thorlabs Scientific Imaging

4.3.56 is_GetVsyncCount

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_GetVsyncCount (HIDS hCam, long* pIntr, long* pActIntr)
```

Description

`is_GetVsyncCount()` reads out the VSYNC counter. It will be incremented by 1 each time the sensor starts capturing an image.

Input parameters

`hCam`	Camera handle
`pIntr`	Current VSYNC count
`pActIntr`	Current Frame SYNC count

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetFramesPerSecond()`

4.3.57 is_HasVideoStarted

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_HasVideoStarted (HIDS hCam, BOOL* pbo)
```

Description

Using `is_HasVideoStarted()`, you can check whether the image digitizing process has started. This function is helpful when the `is_FreezeVideo()` function was called with the `IS_DONT_WAIT` parameter.

Input parameters

`hCam`	Camera handle
`pbo`	Returns the digitizing status: 0 = Image capturing has not started yet. 1 = Image capturing has started.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_FreezeVideo()`
- `is_IsVideoFinish()`

© 2015 Thorlabs Scientific Imaging

4.3.58 is_HotPixel

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_HotPixel (HIDS hCam, UINT nCommand, void* pParam, UINT nSizeOfParam)
```

Description

`is_HotPixel()` configures the correction of sensor hot pixels. The correction is performed by the software. The hot pixel list is stored in the camera's non-volatile EEPROM. Some sensor models can also correct hot pixels directly in the sensor.

For further information on hot pixel correction, please refer to Basics: Hot pixels.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

Attention

This correction will not work with subsampling or with binning factors greater than 2.

Note

Previous hot pixel functions

The `is_HotPixel()` function comprises all hot pixel correction functions. The following uc480 API commands are therefore obsolete:

- `is_SetBadPixelCorrection()`
- `is_SetBadPixelCorrectionTable()`
- `is_LoadBadPixelCorrectionTable()`
- `is_SaveBadPixelCorrectionTable()`

See also Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_HOTPIXEL_DISABLE_CORRECTION`	Disables hot pixel correction (Example 1)
`IS_HOTPIXEL_ENABLE_CAMERA_CORRECTION`	Enables hot pixel correction using the hot pixel list(s) stored in the camera EEPROM.
`IS_HOTPIXEL_ENABLE_SOFTWARE_USER_CORRECTION`	Enables hot pixel correction using the user's hot pixel list stored in the computer. This requires the user's hot pixel list to be set (`IS_HOTPIXEL_SET_SOFTWARE_USER_LIST`)
`IS_HOTPIXEL_ENABLE_SENSOR_CORRECTION`	Enables sensor's own hot pixel correction function (if available).
`IS_HOTPIXEL_DISABLE_SENSOR_CORRECTION`	Disables the sensor's own hot pixel correction function.

hCam	Camera handle
IS_HOTPIXEL_GET_CORRECTION_MODE	Returns the currently set hot pixel correction mode (Example 2)
IS_HOTPIXEL_GET_SUPPORTED_CORRECTION_MODES	Returns the supported hot pixel correction modes.The return value is a bitmask with the following constants (combined by OR): • IS_HOTPIXEL_ENABLE_CAMERA_CORRECTION: Hot pixel correction is possible via the hot pixel list in the camera EEPROM. • IS_HOTPIXEL_ENABLE_SOFTWARE_USER_CORRECTION: Hot pixel correction is possible via the user-defined hot pixel list. • IS_HOTPIXEL_ENABLE_SENSOR_CORRECTION: Hot pixel correction is possible via the sensor-internal hot pixel correction.
IS_HOTPIXEL_GET_SOFTWARE_USER_LIST_EXISTS	Indicates whether the user-defined hot pixel list exists in the computer (Example 3)
IS_HOTPIXEL_GET_SOFTWARE_USER_LIST_NUMBER	Returns the number of hot pixels in the user-defined hot pixel list stored in the computer.
IS_HOTPIXEL_GET_SOFTWARE_USER_LIST	Returns the user-defined hot pixel list stored in the computer.
IS_HOTPIXEL_SET_SOFTWARE_USER_LIST	Sets the user-defined hot pixel list that is stored in the computer.
IS_HOTPIXEL_SAVE_SOFTWARE_USER_LIST IS_HOTPIXEL_SAVE_SOFTWARE_USER_LIST_UNICODE	Saves the user-defined hot pixel list to a file. The function can also be used with Unicode file names. (Example 4)
IS_HOTPIXEL_LOAD_SOFTWARE_USER_LIST IS_HOTPIXEL_LOAD_SOFTWARE_USER_LIST_UNICODE	Loads the user-defined hot pixel list from a file. The function can also be used with Unicode file names.
IS_HOTPIXEL_GET_CAMERA_FACTORY_LIST_EXISTS	Indicates whether the factory-set hot pixel list exists.
IS_HOTPIXEL_GET_CAMERA_FACTORY_LIST_NUMBER	Returns the number of hot pixels in the factory-set hot pixel list.
IS_HOTPIXEL_GET_CAMERA_FACTORY_LIST	Returns the factory-set hot pixel list.
IS_HOTPIXEL_GET_CAMERA_USER_LIST_EXISTS	Indicates whether the user-defined hot pixel list exists in the camera EEPROM.
IS_HOTPIXEL_GET_CAMERA_USER_LIST_NUMBER	Returns the number of hot pixels in the user-defined hot pixel list stored in the camera EEPROM.
IS_HOTPIXEL_GET_CAMERA_USER_LIST	Returns the user-defined hot pixel list stored in the camera EEPROM.
IS_HOTPIXEL_SET_CAMERA_USER_LIST	Sets the user-defined hot pixel list stored in the camera EEPROM (Example 5)
IS_HOTPIXEL_DELETE_CAMERA_USER_LIST	Deletes the user-defined hot pixel list from the camera EEPROM.
IS_HOTPIXEL_GET_CAMERA_USER_LIST_MAX_NUMBER	Returns the maximum number of hot pixels that the user can store in the camera EEPROM.

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
IS_HOTPIXEL_GET_MERGED_CAMERA_LIST_NUMBER	Returns the number of hot pixels in a merged list that combines the entries from the factory-set hot pixel list with those of the user-defined hot pixels list stored in the camera EEPROM.
IS_HOTPIXEL_GET_MERGED_CAMERA_LIST	Returns the merged list (Example 6)
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Example 1

```
// Enable/disable correction
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_DISABLE_CORRECTION, NULL, NULL);
nRet = is_HotPixel(hCam, IS_HOTPIXEL_ENABLE_CAMERA_CORRECTION, NULL, NULL);
nRet = is_HotPixel(hCam, IS_HOTPIXEL_ENABLE_SOFTWARE_USER_CORRECTION, NULL, NULL);
nRet = is_HotPixel(hCam, IS_HOTPIXEL_ENABLE_SENSOR_CORRECTION, NULL, NULL);
nRet = is_HotPixel(hCam, IS_HOTPIXEL_DISABLE_SENSOR_CORRECTION, NULL, NULL);
```

Example 2

```
// Read out current mode
INT nMode = 0;
```

```
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_CORRECTION_MODE,
                       (void*)&nMode, sizeof(nMode));
// Query supported modes
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_SUPPORTED_CORRECTION_MODES,
                       (void*)&nMode, sizeof(nMode));
```

Example 3

```
// Query user-defined hot pixel list
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_SOFTWARE_USER_LIST_EXISTS, NULL, NULL);
if (nRet == IS_SUCCESS)
{
    // Query the number of hot pixels in the user-defined list
    INT nNumber = 0;
    nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_SOFTWARE_USER_LIST_NUMBER,
                       (void*)&nNumber, sizeof(nNumber));
    if (nRet == IS_SUCCESS)
    {
        // Allocate sufficient memory. Each hot pixel needs two WORDS
        // memory space.
        // Additional memory space of one WORD per hot pixel is required for numbering.
        WORD *pList = new WORD[1 + 2 * nNumber];
        nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_SOFTWARE_USER_LIST,
                           (void*)pList, (1 + 2 * nNumber) * sizeof(WORD));

        // Change a value and save the list.
        // The number of the hot pixel has to be specified in pList[0]
        pList[1] = 100;
        nRet = is_HotPixel(hCam, IS_HOTPIXEL_SET_SOFTWARE_USER_LIST,
                           (void*)pList, (1 + 2 * nNumber) * sizeof(WORD));

        // Delete unneeded list
        delete [] pList;
    }
}
```

 © 2015 Thorlabs Scientific Imaging

Example 4

```
// Save user-defined list to file
char File1[100];
ZeroMemory(File1, sizeof(File1));
strcpy(File1, "c:\\test.txt");

nRet = is_HotPixel(hCam, IS_HOTPIXEL_LOAD_SOFTWARE_USER_LIST, (void*)File1, 0);

nRet = is_HotPixel(hCam, IS_HOTPIXEL_SAVE_SOFTWARE_USER_LIST, (void*)File1, 0);

// Unicode
wchar_t File2[100];
ZeroMemory(File2, sizeof(File2));
wcscpy(File2, L"c:\\test.txt");

nRet = is_HotPixel(hCam, IS_HOTPIXEL_LOAD_SOFTWARE_USER_LIST_UNICODE, (void*)File2, 0);

nRet = is_HotPixel(hCam, IS_HOTPIXEL_SAVE_SOFTWARE_USER_LIST_UNICODE, (void*)File2, 0);
```

Example 5

```
// Save user-defined list to the camera EEPROM
INT nNumber = 0;
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_CAMERA_USER_LIST_MAX_NUMBER,
                       (void*)&nNumber , sizeof(nNumber));
if (nRet == IS_SUCCESS)
{
    // Write the maximum number of hot pixels to EEPROM
    WORD *pList = new WORD[1 + 2 * nNumber];
    pList[0] = nNumber;
    for (int i = 0; i < nNumber; i++)
    {
        pList[1 + 2 * i] = x_value;
        pList[2 + 2 * i] = y_value;
    }

    nRet = is_HotPixel(hCam, IS_HOTPIXEL_SET_CAMERA_USER_LIST,
                        (void*)pList, (1 + 2 * nNumber) * sizeof(WORD));

    delete [] pList;

    // Delete user-defined EEPROM list
    nRet = is_HotPixel(hCam, IS_HOTPIXEL_DELETE_CAMERA_USER_LIST, NULL, NULL);
}
```

Example 6

```
// Return combined list
INT nNumber = 0;
INT nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_MERGED_CAMERA_LIST_NUMBER,
                        (void*)&nNumber , sizeof(nNumber));
if (nRet == IS_SUCCESS)
{
     // Allocate sufficient memory. Each hot pixel needs two WORDS
     // memory space.
     // Additional memory space of one WORD per hot pixel is required for numbering.
     WORD *pList = new WORD[1 + 2 * nNumber];
    nRet = is_HotPixel(hCam, IS_HOTPIXEL_GET_MERGED_CAMERA_LIST,
                        (void*)pList, (1 + 2 * nNumber) * sizeof(WORD));

    // Delete unneeded list
    delete [] pList;
}
```

4.3.59 is_ImageFile

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ImageFile (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

`is_ImageFile()` loads and save an image from or to a file. The image must be BMP, JPEG or PNG format. The image is loaded into the active image memory or read-out from the active image memory.

Note

When saving an image `is_FreezeVideo()` sjould not be called with the `IS_DONT_WAIT` parameter, because the image acquisition might not be completed at the time of saving.

The bitmap is stored with the color depth that was used when allocating the image memory (in DIB mode) or that was set for the current color mode (in Direct3D mode). You can save images with a bit depth of more than 8 bit in the PNG format. 12 bit formats are converted into 16 bit. JPEG files are always saved with a color depth of 8 or 24 bits.

Note

In Direct3D or OpenGL mode, overlay data is not saved.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

The following functions are obsolete by the `is_ImageFile()` function:

- `is_LoadImage()`
- `is_LoadImageMem()`
- `is_SaveImage()`
- `is_SaveImageMem()`
- `is_SaveImageEx()`
- `is_SaveImageMemEx()`

See also: Obsolete functions

Input parameters

hCam	Camera handle
⊟ nCommand	

© 2015 Thorlabs Scientific Imaging

IS_IMAGE_FILE_CMD_LOAD	Loads an image file (bmp, jpg, png) (Example 1) The function can be used with UNICODE file names.
IS_IMAGE_FILE_CMD_SAVE	Saves an image file (bmp, jpg, png) (Example 2) The function can be used with UNICODE file names.
pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the IMAGE_FILE_PARAMS structure

wchar_t	pwchFileName	Name of the file to be loaded/saved (Unicode). If NULL is passed, the "Open file"/"Save as" dialog opens.
UINT	nFileType	File type to be saved: • IS_IMG_BMP • IS_IMG_JPG • IS_IMG_PNG
UINT	nQuality	Sets the image quality for JPEG and PNG (and therefore the compression). The higher the value, the better the quality is: • 100 = maximum quality with minimum compression • If the parameter is set to 0, the the default value of 75 is used. For BMP the parameter is ignored.
char	ppcImageMem	**When loading:** Pointer to an image memory and pointer to the corresponding ID. If both pointers are NULL the image is loaded into the active image memory. If both pointers are valid a new memory is allocated. This memory must be released with is_FreeImageMem(). **When saving:** Pointer to an image memory and pointer to the corresponding ID. If both pointers are NULL the image is saved from the active image memory. If both pointers are valid the corresponding memory is used.
UINT	pnImageID	
BYTE	reserved[32]	reserved

Return values

IS_FILE_READ_INVALID_BMP_ID	The specified file is not a valid bitmap file.
IS_FILE_READ_OPEN_ERROR	The file cannot be opened.
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_GetImageMem()
- is_SetImageMem()

© 2015 Thorlabs Scientific Imaging

Example 1

```
IMAGE_FILE_PARAMS ImageFileParams;

ImageFileParams.pwchFileName = NULL;
ImageFileParams.pnImageID = NULL;
ImageFileParams.ppcImageMem = NULL;
ImageFileParams.nQuality = 0;

// Load bitmap into active memory (with file open dialog)
ImageFileParams.nFileType = IS_IMG_BMP;
INT nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_LOAD, (void*)&ImageFileParams,
                        sizeof(ImageFileParams));

// Load jpeg into active memory (with file open dialog)
ImageFileParams.nFileType = IS_IMG_JPG;
nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_LOAD, (void*)&ImageFileParams,
                    sizeof(ImageFileParams));

// Alloc image memory and load bitmap into it (without file open dialog)
char* pcMemory = NULL;
UINT nID = 0;
ImageFileParams.pwchFileName = L"c:\\test.bmp";
ImageFileParams.pnImageID = &nID;
ImageFileParams.ppcImageMem = &pcMemory;
ImageFileParams.nFileType = IS_IMG_BMP;
nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_LOAD, (void*)&ImageFileParams,
                    sizeof(ImageFileParams));
```

Example 2

```
IMAGE_FILE_PARAMS ImageFileParams;

ImageFileParams.pwchFileName = NULL;
ImageFileParams.pnImageID = NULL;
ImageFileParams.ppcImageMem = NULL;
ImageFileParams.nQuality = 0;

// Save bitmap from active memory to file (with file open dialog)
ImageFileParams.nFileType = IS_IMG_BMP;
INT nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_SAVE, (void*)&ImageFileParams,
                        sizeof(ImageFileParams));

// Save jpeg from active memory with quality 80 (without file open dialog)
ImageFileParams.pwchFileName = L"c:\\test.jpg";
ImageFileParams.nFileType = IS_IMG_JPG;
ImageFileParams.nQuality = 80;
nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_SAVE, (void*)&ImageFileParams,
                    sizeof(ImageFileParams));

// Save png from special memory with quality 50 (with file open dialog)
 ImageFileParams.pwchFileName = NULL; ImageFileParams.pnImageID = &nID;   // valid ID
ImageFileParams.ppcImageMem = &pcMemory;     // valid buffer
ImageFileParams.nFileType = IS_IMG_PNG;
ImageFileParams.nQuality = 50;
nRet = is_ImageFile(m_hCam, IS_IMAGE_FILE_CMD_SAVE, (void*)&ImageFileParams,
                    sizeof(ImageFileParams));
```

4.3.60 is_ImageFormat

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ImageFormat (HIDS hCam,
                    UINT nCommand,
                    void *pParam,
                    UINT nSizeOfParam)
```

Description

Using `is_ImageFormat()`, you can query a list of possible image sizes and set a new image format if supported by your DCx Camera model. This is useful for sensors that do not support a free selection of the area of interest or image format. Using the AOI, binning/subsampling or scaling functions, the driver sets the selected image format to achieve the best possible image quality. For a complete list of available image formats see table "Image formats" below.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IMGFRMT_CMD_GET_LIST`	Returns a list of all image formats supported by the sensor. ⊟ More details You can query the number of entries in the list with `IMGFRMT_CMD_GET_NUM_ENTRIES`. • `pParam`: Pointer to list of type `IMAGE_FORMAT_LIST`. The list must be preallocated as specified below. • `nSizeOfParam`: Size of the list Size of (`IMAGE_FORMAT_LIST` + (number of list entries - 1) * Size of (`IMAGE_FORMAT_INFO`))
`IMGFRMT_CMD_GET_NUM_ENTRIES`	Returns the number of entries in the list. ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the number of list entries. • `nSizeOfParam`: 4
`IMGFRMT_CMD_SET_FORMAT`	Sets the desired image format. ⊟ More details • `pParam`: Pointer to variable of type `UINT` passing the format ID of the desired image format. • `nSizeOfParam`: 4

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
IMGFRMT_CMD_GET_ARBITRARY_AOI_SUPPORTED	Returns if the sensor supports a free selection of the area of interest (AOI). More details • pParam: Pointer to variable of type UINT indicating if free AOI selection is supported: 0 = No free AOI supported 1 = Free AOI supported • nSizeOfParam: 4
pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the IMAGE_FORMAT_LIST list of image formats

UINT	nSizeOfListEntry	Must be preset with the size of a list entry in bytes
UINT	nNumListElements	Must be preset with the number of list entries (from IMGFRMT_CMD_GET_NUM_ENTRIES)
UINT	nReserved[4]	Reserved
IMAGE_FORMAT_INFO	FormatInfo[0]	First entry in the list. After having been filled by IMGFRMT_CMD_GET_LIST, the list contains additional entries FormatInfo[1]… FormatInfo[nNumListElements-1].

Contents of the list entry IMAGE_FORMAT_INFO

INT	nFormatID	Format ID of the specified image format (see table "Image formats" below)
UINT	nWidth	Width of the area of interest
UINT	nHeight	Height of the area of interest
UINT	nX0	Start point of the area of interest (X)
UINT	nY0	Start point of the area of interest (Y)
UINT	nSupportedCaptureModes	Image capture modes supported for this format (see table below)
UINT	nBinningMode	Binning mode used
UINT	nSubsamplingMode	Subsampling mode used
IS_CHAR	strFormatName[64]	Description of the format
double	dSensorScalerFactor	Scaling factor used (only sensors that support scaling).
UINT	nReserved[24]	Reserved

Possible values for CAPTUREMODE

CAPTMODE_SINGLE	Freerun mode, single frame (freerun snap)
CAPTMODE_FREERUN	Freerun mode, continuous (freerun live)
CAPTMODE_TRIGGER_SOFT_SINGLE	Software triggered mode, single frame

CAPTMODE_TRIGGER_SOFT_CONTINUOUS	Software triggered mode, continuous
CAPTMODE_TRIGGER_HW_SINGLE	Hardware triggered mode, single frame
CAPTMODE_TRIGGER_HW_CONTINUOUS	Hardware triggered mode, continuous

For further information on the image capture modes, see also in the Basics: Operating modes chapter.

© 2015 Thorlabs Scientific Imaging

Image formats of CMOS sensors

Format ID	Resolution	Name	Camera model		
			DCC1645C	DCC1545M	DCC1240x
1	3264x2448	(8M)			
2	3264x2176	(8M 3:2)			
3	3264x1836	(8M 16:9)			
4	2592x1944	(5M)			
5	2048x1536	(3M)			
6	1920x1080	(Full HD 16:9)			
7	1632x1224	(2M)			
8	1280x960	(1.2M 4:3)	X	X	X
9	1280x720	(HD 16:9)	X	X	X
11	960x480	(WVGA 2:1)	X	X	X
12	800x480	(WVGA)	X	X	X
13	640x480	(VGA)	X	X	X
14	640x360	(VGA 16:9)	X	X	X
15	400x240	(WQVGA)	X	X	X
16	352x288	(CIF)	X	X	X
17	288x352	(CIF Portrait)	X	X	X
18	320x240	(QVGA)	X	X	X
19	240x320	(QVGA Portrait)	X	X	X
20	1600x1200	(UXGA)			
21	3840x2748	(10M)			
22	1920x1080	(Full HD 16:9, HQ)			
23	2560x1920	(5M)			
24	768x576	(CCIR)	X	X	X
25	1280x1024	(1.3M SXGA)	X	X	X
26	2448x2048	(5M)			
27	1024x768	(XGA)	X	X	X
28	1024x1024	(1M)	X	X	
29	800x600	(SVGA)	X	X	X
30	1360x1024	(1.4M 4:3)			

Image formats of CCD sensors

Format ID	Resolution	Name	Camera model	
			DCU-223x	DCU-224x
1	3264x2448	(8M)		
2	3264x2176	(8M 3:2)		
3	3264x1836	(8M 16:9)		
4	2592x1944	(5M)		
5	2048x1536	(3M)		
6	1920x1080	(Full HD 16:9)		
7	1632x1224	(2M)		
8	1280x960	(1.2M 4:3)		X
9	1280x720	(HD 16:9)		X
11	960x480	(WVGA 2:1)	X	X
12	800x480	(WVGA)	X	X
13	640x480	(VGA)	X	X
14	640x360	(VGA 16:9)	X	
15	400x240	(WQVGA)	X	
16	352x288	(CIF)	X	
17	288x352	(CIF Portrait)	X	
18	320x240	(QVGA)	X	
19	240x320	(QVGA Portrait)	X	
20	1600x1200	(UXGA)		
21	3840x2748	(10M)		
22	1920x1080	(Full HD 16:9, HQ)		
23	2560x1920	(5M)		
24	768x576	(CCIR)	X	X
25	1280x1024	(1.3M SXGA)		X
26	2448x2048	(5M)		
27	1024x768	(XGA)	X	X
28	1024x1024	(1M)		X
29	800x600	(SVGA)	X	X
30	1360x1024	(1.4M 4:3)		

Return values

`IS_CANT_ADD_TO_SEQUENCE`	The image memory is already included in the sequence and cannot be added again.
`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.

© 2015 Thorlabs Scientific Imaging

IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_INVALID_PIXEL_CLOCK	This setting is not available for the currently set pixel clock frequency.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- is_AOI()
- is_SetBinning()
- is_SetSubSampling()

Example

```
HIDS hCam;
char strCamFileName[256];
int  nRet;

// Get number of available formats and size of list
UINT count;
UINT bytesNeeded = sizeof(IMAGE_FORMAT_LIST);
nRet = is_ImageFormat(hCam, IMGFRMT_CMD_GET_NUM_ENTRIES, &count, 4);
bytesNeeded += (count - 1) * sizeof(IMAGE_FORMAT_INFO);
void* ptr = malloc(bytesNeeded);

// Create and fill list
IMAGE_FORMAT_LIST* pformatList = (IMAGE_FORMAT_LIST*) ptr;
pformatList->nSizeOfListEntry = sizeof(IMAGE_FORMAT_INFO);
pformatList->nNumListElements = count;
nRet = is_ImageFormat(hCam, IMGFRMT_CMD_GET_LIST, pformatList, bytesNeeded);

// Activate trigger mode for capturing high resolution images (USB uEye XS)
nRet = is_StopLiveVideo(hCam, IS_WAIT);
nRet = is_SetExternalTrigger(hCam, IS_SET_TRIGGER_SOFTWARE);

// Prepare for creating image buffers
char* pMem = NULL;
int memID = 0;

// Set each format and then capture an image
IMAGE_FORMAT_INFO formatInfo;
for (int i = 0; i < count; i++)
{
     formatInfo = pformatList->FormatInfo[i];
     int width = formatInfo.nWidth;
     int height = formatInfo.nHeight;

     // Allocate image mem for current format, set format
     nRet = is_AllocImageMem(hCam, width, height, 24, &pMem, &memID);
     nRet = is_SetImageMem(hCam, pMem, memID);
     nRet = is_ImageFormat(hCam, IMGFRMT_CMD_SET_FORMAT, &formatInfo.nFormatID, 4);

     // Capture image
     nRet = is_FreezeVideo(hCam, IS_WAIT);
}
```

 © 2015 Thorlabs Scientific Imaging

4.3.61 is_InitCamera

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_InitCamera (HIDS* phf, HWND hWnd)
```

Description

`is_InitCamera()` starts the driver and establishes the connection to the camera. After successful initialization, this function assigns the camera handle. All subsequent functions require this handle as the first parameter.

When using Direct3D or OpenGL for image display, you can pass a handle to the output window.

Notes

1. Multi-camera environments

When using multiple cameras in parallel operation on a single system, you should assign a unique camera ID to each camera. To initialize or select a camera with `is_InitCamera()`, the `phCam` handle must previously have been set to the desired camera ID.

To initialize or select the next available camera without specifying a camera ID, `phCam` has to be preset with 0.

2. Thread safety

We recommend that you call the following functions exclusively from a single thread in order to avoid unpredictable behavior of the application.

- `is_InitCamera()`
- `is_SetDisplayMode()`
- `is_ExitCamera()`

See also General: Thread programming

Input parameters

`phCam`	Pointer to the camera handle When you call this function, the pointer value has the following meaning: 0: The first available camera will be initialized or selected. 1-254: The camera with the specified camera ID will be initialized or selected.
`*phCam | IS_USE_DEVICE_ID`	The camera is opened using the device ID instead of the camera ID. For details on device ID please refer to the `is_GetCameraList()` chapter.
`*phCam | IS_ALLOW_STARTER_FW_UPLOAD`	During initialization of the camera, this parameter checks whether a new version of the starter firmware is required. If it is, the new starter firmware is updated automatically (only GigE uEye SE/RE/CP cameras). To ensure backward compatibility of applications, always call `is_InitCamera()` without the `IS_ALLOW_STARTER_FW_UPLOAD` parameter first. Only if an error occurs, call the function with this parameter set (see

	Example below).
hWnd	Pointer to the window where the Direct3D image will be displayed If hWnd = NULL, DIB mode will be used for image display.

Return values

IS_ALL_DEVICES_BUSY	All cameras are in use
IS_BAD_STRUCTURE_SIZE	An internal structure has an incorrect size.
IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CANT_OPEN_REGISTRY	Error opening a Windows registry key
IS_CANT_READ_REGISTRY	Error reading settings from the Windows registry
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_CRC_ERROR	A CRC error-correction problem occurred while reading the settings.
IS_DEVICE_ALREADY_PAIRED	The device is already paired.
IS_DEVICE_NOT_COMPATIBLE	The device is not compatible to the drivers.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
IS_ERROR_CPU_IDLE_STATES_CONFIGURATION	The configuration of the CPU idle has failed.
IS_FILE_WRITE_OPEN_ERROR	File cannot be opened for writing or reading.
IS_INCOMPATIBLE_SETTING	Because of other incompatible settings the function is not possible.
IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_DEVICE_ID	The device ID is invalid. Valid IDs start from 1 for USB cameras.
IS_INVALID_EXPOSURE_TIME	This setting is not available for the currently set exposure time.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle

© 2015 Thorlabs Scientific Imaging

IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_INVALID_PIXEL_CLOCK	This setting is not available for the currently set pixel clock frequency.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_IMAGE_MEM_ALLOCATED	The driver could not allocate memory.
IS_NO_IR_FILTER	No IR filter available
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_STARTER_FW_UPLOAD_NEEDED	The camera's starter firmware is not compatible with the driver and needs to be updated.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- `is_ExitCamera()`
- `is_EnableAutoExit()`
- `is_GetCameraList()`
- `is_SetCameraID()`
- `is_GetCameraInfo()`

Example

```
//Open camera with ID 1
HIDS hCam = 1;
INT nRet = is_InitCamera (&hCam, NULL);
```

```
if (nRet != IS_SUCCESS)
{
  //Check if GigE uEye SE needs a new starter firmware
  if (nRet == IS_STARTER_FW_UPLOAD_NEEDED)
  {
    //Calculate time needed for updating the starter firmware
    INT nTime;
    is_GetDuration (hCam, IS_SE_STARTER_FW_UPLOAD, &nTime);
    /*
    e.g. have progress bar displayed in separate thread
   */

    //Upload new starter firmware during initialization
    hCam =  hCam | IS_ALLOW_STARTER_FW_UPLOAD;
    nRet = is_InitCamera (&hCam, NULL);

    /*
     end progress bar
    */
  }
}
```

Sample programs

- uc480MultipleCameraScan (C++)
- uc480Console (C++)
- uc480C# Demo (C#)

© 2015 Thorlabs Scientific Imaging

4.3.62 is_InitEvent

USB 2.0 USB 3.0	-

Syntax

```
INT is_InitEvent (HIDS hCam, HANDLE hEv, INT which)
```

Description

`is_InitEvent()` initializes the event handle for the specified event object. This registers the event object in the uc480 kernel driver.

Attention

Using USB cameras under Windows

The following events require a Windows message loop. This message loop has to be executed by the thread that loads the uc480 API. The message loop is usually provided by the application window. In some cases, the message loop might not be created automatically (e.g. in console applications). In this case you will need to implement the message loop yourself.

This applies to the following uc480 events:

- `IS_SET_EVENT_REMOVE`
- `IS_SET_EVENT_REMOVAL`
- `IS_SET_EVENT_DEVICE_RECONNECTED`
- `IS_SET_EVENT_NEW_DEVICE`

If no message loop exists, a USB camera will not be automatically detected after reconnecting.

Input parameters

`hCam`	Camera handle
`hEv`	Event handle created by the `CreateEvent()` Windows API function.
`which`	ID of the event to be initialized (see `is_EnableEvent()`)

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_EnableEvent()`
- `is_DisableEvent()`
- `is_ExitEvent()`

Example

```
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

//Enable frame event, start image capture and wait for event
is_InitEvent(hCam, hEvent, IS_SET_EVENT_FRAME);
is_EnableEvent(hCam, IS_SET_EVENT_FRAME);
is_FreezeVideo(hCam, IS_DONT_WAIT);
DWORD dwRet = WaitForSingleObject(hEvent, 1000);
if (dwRet == WAIT_TIMEOUT)
{
    /* wait timed out */
}
else if (dwRet == WAIT_OBJECT_0)
{
    /* event signalled */
}
is_DisableEvent(hCam, IS_SET_EVENT_FRAME);
is_ExitEvent(hCam, IS_SET_EVENT_FRAME);
CloseHandle(hEvent);
```

Sample programs

- SimpleLive (C++)
- uc480Event (C++)

© 2015 Thorlabs Scientific Imaging

4.3.63 is_InitImageQueue

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_InitImageQueue (HIDS hCam, INT nMode)
```

Description

`is_InitImageQueue()` enables the queue mode for existing image memory sequences. New images will be added to the end of the queue on arrival (FIFO principle). The image memory sequence has to be created with `is_AddToSequence()` prior to calling is_InitImageQueue(). With `is_WaitForNextImage()` you can query the pointer and sequence ID of the first (i.e. oldest) image in the sequence.

Note

Image memory sequences can also be used without queue mode. In this case the current image memory has to be queried with `is_GetActSeqBuf()` on every frame event. Disadvantage of this proceeding is that at very high frame rates it may happen that additional images arrive between the frame event and accessing/locking the memory. The images arriving in this period will be skipped when you query the current image.

When the queue mode is used (`is_InitImageQueue()`), however, you can be sure to always receive the oldest image which has not yet been queried. In addition, image memories are automatically locked immediately after receiving the image. This prevents images from being overwritten when very high frame rates and few image memories are used.

Input parameters

`hCam`	Camera handle
`nMode`	Queue mode. Currently only `nMode = 0` is supported.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_ExitImageQueue()`
- `is_WaitForNextImage()`
- `is_AddToSequence()`

Example

```
// A previously initialized camera continuously captures images
// until a timeout or transfer error occurs.
// Note: image memories have to be allocated before this
```

```
is_InitImageQueue (m_hCam, 0);
INT nMemID = 0;
char *pBuffer = NULL;

while (IS_SUCCESS == is_WaitForNextImage(m_hCam, 1000, &pBuffer, &nMemID))
{
    is_SaveImageMem (m_hCam, "image.bmp", pBuffer, nMemID);
    is_UnlockSeqBuf (m_hCam, nMemID, pBuffer);
}
is_ExitImageQueue (m_hCam);
```


© 2015 Thorlabs Scientific Imaging

4.3.64 is_InquireImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_InquireImageMem (HIDS hCam, char* pcMem, int nID,
                        int* pnX, int* pnY,
                        int* pnBits, int* pnPitch);
```

Description

`is_InquireImageMem()` reads out the properties of an allocated image memory.

Input parameters

`hCam`	Camera handle
`pMem`	Pointer to the starting address of the image memory as allocated by `is_AllocImageMem()`
`NID`	ID of the image memory as allocated by `is_AllocImageMem()`
`pnX`	Returns the width used to define the image memory. You can also pass `NULL` instead.
`pnY`	Returns the height used to define the image memory. You can also pass `NULL` instead.
`pnBits`	Returns the bit width used to define the image memory. You can also pass `NULL` instead.
`pnPitch`	Returns the line increment of the image memory. You can also pass `NULL` instead.

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`

- `is_SetImageMem()`
- `is_SetAllocatedImageMem()`
- `is_GetColorDepth()`

© 2015 Thorlabs Scientific Imaging

4.3.65 is_IO

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_IO(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

With the `is_IO()` function you control all flash and trigger functions and the additional digital outputs (GPIOs) of some DCx Camera models. For information on GPIO wiring, please refer to the Electrical Specifications chapter.

Additionally you can toggle the color of the LED on the back of the DCU22x and DCC1240x camera housing.

Note

GPIOs are available only for DCC3240x cameras. The GPIOs are not provided with optocouplers and use TTL/LVCMOS voltages. For information on GPIO wiring, please refer to the Electrical specifications chapter.

- Rolling shutter cameras:
  Using `is_IO()`, you can determine the times required to implement a global flash function for rolling shutter cameras. This way, a rolling shutter camera can also be used as a global shutter camera provided that no ambient light falls on the sensor outside the flash period.
  If the exposure time is set too short so that no global flash operation is possible, the function returns `IS_NO_SUCCESS`.

  Note

  To use a rolling shutter camera with the global start function, first call the `is_SetGlobalShutter()` function. Otherwise, incorrect values will be returned for `Delay` and `Duration`.

- Global shutter cameras:
  In freerun mode, the exposure of global shutter cameras is delayed if the exposure time is not set to the maximum value. `is_IO()` determines the required delay in order to synchronize exposure and flash operation. In triggered mode, the return values for delay and flash duration are 0, since no delay is necessary before exposure starts.

For further information, please refer to the chapters Camera basics: Shutter methods, Digital input/output (trigger/flash) and Operating modes.

Attention

Accuracy of flash synchronization

The following parameters have an influence on the camera's internal timing:

- Image geometry (CMOS and CCD sensors)
- Pixel clock (CMOS and CCD sensors)
- Exposure time (CCD sensors)

If you change any of these parameters, you will have to set the flash duration and flash delay parameters once again.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value,

`pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

The following functions are obsolete by the `is_IO()` function:

- `is_GetGlobalFlashDelays()`
- `is_SetFlashDelay()`
- `is_SetFlashStrobe()`
- `is_SetIO()`
- `is_SetIOMask()`
- `is_SetLED()`

See also: Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
GPIO	
`IS_IO_CMD_GPIOS_GET_SUPPORTED`	Returns the supported GPIO ports (Example 1) • `IO_FLASH_GPIO_PORT_MASK (IO_FLASH_MODE_GPIO_1 | IO_FLASH_MODE_GPIO_2)`
`IS_IO_CMD_GPIOS_GET_SUPPORTED_INPUTS`	Returns the supported GPIO inputs (Example 1)
`IS_IO_CMD_GPIOS_GET_SUPPORTED_OUTPUTS`	Returns the supported GPIO outputs (Example 1)
`IS_IO_CMD_GPIOS_GET_DIRECTION`	Returns the input/output mask of the GPIOs (Example 2)
`IS_IO_CMD_GPIOS_SET_DIRECTION`	Set the GPIO on input/output (Example 2) • `IO_FLASH_MODE_GPIO_1`: Sets GPIO 1 as outout. • `IO_FLASH_MODE_GPIO_2`: Sets GPIO 2 as output.
`IS_IO_CMD_GPIOS_GET_STATE`	Returns the state of the GPIO (High, Low) (Example 2)
`IS_IO_CMD_GPIOS_SET_STATE`	Sets the state of the GPIOs if they are defined as output (High, Low) (Example 2)
`IS_IO_CMD_GPIOS_GET_CONFIGURATION`	Returns the configuration of a GPIO port (Example 10)
`IS_IO_CMD_GPIOS_SET_CONFIGURATION`	Sets the configuration of a GPIO port (Example 11, Example 12)
Flash	
`IS_IO_CMD_FLASH_GET_SUPPORTED_GPIOS`	Returns the GPIOs which can be used for flash output (Example 6)
`IS_IO_CMD_FLASH_GET_MODE`	Returns the current flash mode, see below (Example 7)
`IS_IO_CMD_FLASH_SET_MODE`	Sets the flash mode (Example 7) • `IO_FLASH_MODE_OFF`: Disables the digital output.

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	• IO_FLASH_MODE_TRIGGER_LO_ACTIVE: Enables the flash strobe in trigger mode. The digital output is set to low level for the flash duration. • IO_FLASH_MODE_TRIGGER_HI_ACTIVE: Enables the flash strobe in trigger mode. The digital output is set to high level for the flash duration. • IO_FLASH_MODE_CONSTANT_HIGH: Statically sets the digital output to high level (HIGH). • IO_FLASH_MODE_CONSTANT_LOW: Statically sets the digital output to low level (LOW). • IO_FLASH_MODE_FREERUN_LO_ACTIVE: Enables the flash strobe in freerun mode. The digital output is set to low level for the flash duration. • IO_FLASH_MODE_FREERUN_HI_ACTIVE: Enables the flash strobe in freerun mode. The digital output is set to high level for the flash duration.
IS_IO_CMD_FLASH_GET_GLOBAL_PARAMS	Returns the parameters for the global exposure window (Example 4)
IS_IO_CMD_FLASH_APPLY_GLOBAL_PARAMS	Returns the parameters for the global exposure window and sets them as flash parameters (Examplel 4)
IS_IO_CMD_FLASH_GET_PARAMS	Returns the current values for flash delay and duration (Example 5)
IS_IO_CMD_FLASH_SET_PARAMS	Sets the current values for flash delay and duration (Example 5)
IS_IO_CMD_FLASH_GET_PARAMS_MIN	Returns the minimum possible values for flash delay and duration (Example 5)
IS_IO_CMD_FLASH_GET_PARAMS_MAX	Returns the maximum possible values for flash delay and duration (Example 5)
IS_IO_CMD_FLASH_GET_PARAMS_INC	Returns the increments for flash delay and duration (Example 5)
IS_IO_CMD_FLASH_GET_GPIO_PARAMS_MIN	Returns the minimum possible parameters for the GPIOs as shorter flash delay and flash duration are possible when using the GPIOs for flash (Example 13).
IS_IO_CMD_FLASH_SET_GPIO_PARAMS	Sets the flash delay and flash duration and allows the minimum values for GPIOs. **Attention:** For values below 20 µs an unpredictable behavior can occur when flashing is done via the normal flash pin (Example 13).
Pulse-width modulation	
IS_IO_CMD_PWM_GET_SUPPORTED_GPIOS	Returns the GPIOs which can be used for pulse-width modulation (PWM) (Example 6)
IS_IO_CMD_PWM_GET_MODE	Returns the current PWM mode (Example 9)
IS_IO_CMD_PWM_SET_MODE	Sets the current PWM mode (Example 9) • IS_FLASH_MODE_PWM: Sets the flash output as

`hCam`	Camera handle
	output for PWM mode. • `IO_GPIO_1`: Sets GPIO 1 as output. • `IO_GPIO_2`: Sets GPIO 2 as output.
`IS_IO_CMD_PWM_GET_PARAMS`	Returns the current values of the PWM parameters (Example 8)
`IS_IO_CMD_PWM_SET_PARAMS`	Sets the current values of the PWM parameters (Example 8)
`IS_IO_CMD_PWM_GET_PARAMS_MIN`	Returns the minimum possible values for PWM parameters (Example 8)
`IS_IO_CMD_PWM_GET_PARAMS_MAX`	Returns the maximum possible values for PWM parameters (Example 8)
`IS_IO_CMD_PWM_GET_PARAMS_INC`	Returns the increments of the PWM parameters (Example 8)
LED	
`IS_IO_CMD_LED_GET_STATE`	Returns the state of the LED (Example 3)
`IS_IO_CMD_LED_SET_STATE`	Sets the state of the LED (Example 3) • `IO_LED_STATE_1`: Sets LED to orange. • `IO_LED_STATE_2`: Sets LED to green.
`IS_IO_CMD_LED_TOGGLE_STATE`	Toggles between the LED states (Example 3)
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

© 2015 Thorlabs Scientific Imaging

Contents of the IO_FLASH_PARAMS structure

INT	s32Delay	Flash delay (in µs)
UINT	u32Duration	Flash duration (in µs) If 0 is passed, the flash output will be active until the end of the exposure time. For sensors with Global Start Shutter this is the time until the end of exposure of the first sensor row.

Contents of the IO_PWM_PARAMS structure

double	dblFrequency_Hz	Frequency of the pulse-width modulation (PWM) 1.0…10 000 Hz
double	dbl_DutyCycle	Duty cycle of the pulse-width modulation 0.0…1.0 (1.0 corresponds to 100 %)

Contents of the IO_GPIO_CONFIGURATION structure

UINT	u32Gpio	Sets the GPIO whose configuration is to be read or set (IO_GPIO_1, IO_GPIO_2). So this value must be initialized before the GPIO configuration is read or set.
UINT	u32Caps	When reading the configuration: ORed bitmask of the supported GPIO modes (IS_GPIO_INPUT | IS_GPIO_OUTPUT…).
UINT	u32Configuration	• When reading the configuration: returns the current set configuration • When setting the configuration: sets the configuration
UINT	u32State	• When reading the configuration: returns the current state of the GPIO (0 = Low, 1 = High). • When setting the configuration: sets the state of the GPIO (0 = Low, 1 = High).
UINT	u32Reserved[12]	Reserved

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this

	function or setting.
IS_SUCCESS	Function executed successfully
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.
IS_TRIGGER_NOT_ACTIVATED	The function is not possible as trigger is disabled.

© 2015 Thorlabs Scientific Imaging

Example 1

```
INT nRet = IS_SUCCESS;

UINT nSupportedIOs = 0;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_GET_SUPPORTED,
             (void*)&nSupportedIOs, sizeof(nSupportedIOs));

UINT nSupportedInputs = 0;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_GET_SUPPORTED_INPUTS,
             (void*)&nSupportedInputs, sizeof(nSupportedInputs));

UINT nSupportedOutputs = 0;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_GET_SUPPORTED_OUTPUTS,
             (void*)&nSupportedOutputs, sizeof(nSupportedOutputs));
```

Example 2

```
INT nRet = IS_SUCCESS;

UINT nDirection = 0;

// Get direction
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_GET_DIRECTION,
             (void*)&nDirection, sizeof(nDirection));

// Set GPIO1 and GPIO2 to output
nDirection = IO_GPIO_1 | IO_GPIO_2;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_SET_DIRECTION,
             (void*)&nDirection, sizeof(nDirection));

// Get the current state of the GPIOs
UINT nCurrentState = 0;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_GET_STATE,
             (void*)&nCurrentState, sizeof(nCurrentState));

// Set GPIO1 to high, GPIO2 to low
nCurrentState = IO_GPIO_1;
nRet = is_IO(m_hCam, IS_IO_CMD_GPIOS_SET_STATE,
             (void*)&nCurrentState, sizeof(nCurrentState));
```

Example 3

```
INT nRet = IS_SUCCESS;

UINT nLED = 0;

// Get the current state of the LED
nCurrentState = 0;
nRet = is_IO(m_hCam, IS_IO_CMD_LED_GET_STATE,
             (void*)&nCurrentState, sizeof(nCurrentState));

// Set LED to state 1 (red)
nCurrentState = IO_LED_STATE_1;
nRet = is_IO(m_hCam, IS_IO_CMD_LED_SET_STATE,
             (void*)&nCurrentState, sizeof(nCurrentState));

// Toggle LED state to green
nRet = is_IO(m_hCam, IS_IO_CMD_LED_TOGGLE_STATE, NULL, 0);
```

Example 4

```
INT nRet = IS_SUCCESS;

// Read the global flash params
IO_FLASH_PARAMS flashParams;
```

```
INT nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_GLOBAL_PARAMS,
                 (void*)&flashParams, sizeof(flashParams));
if (nRet == IS_SUCCESS)
{
    INT nDelay    = flashParams.s32Delay;
    UINT nDuration = flashParams.u32Duration;
}

// Read the global flash params and set the flash params to these values
INT nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_APPLY_GLOBAL_PARAMS, NULL, 0);
```


© 2015 Thorlabs Scientific Imaging

Example 5

```
INT nRet = IS_SUCCESS;

IO_FLASH_PARAMS flashParams;

// Get the minimum values for flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_PARAMS_MIN,
             (void*)&flashParams, sizeof(flashParams));
if (nRet == IS_SUCCESS)
{
    INT nFlashDelayMin    = flashParams.s32Delay;
    UINT nFlashDurationMin = flashParams.u32Duration;
}

// Get the maximum values for flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_PARAMS_MAX,
             (void*)&flashParams, sizeof(flashParams));
if (nRet == IS_SUCCESS)
{
    INT nFlashDelayMax    = flashParams.s32Delay;
    UINT nFlashDurationMax = flashParams.u32Duration;
}

// Get the increment for flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_PARAMS_INC,
             (void*)&flashParams, sizeof(flashParams));
if (nRet == IS_SUCCESS)
{
   UINT nFlashDelayInc    = flashParams.s32Delay;
    UINT nFlashDurationInc = flashParams.u32Duration;
}

// Get the current values for flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_PARAMS,
             (void*)&flashParams, sizeof(flashParams));
if (nRet == IS_SUCCESS)
{
    INT nCurrentFlashDelay    = flashParams.s32Delay;
    UINT nCurrentFlashDuration = flashParams.u32Duration;
}

// Set the current values for flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_PARAMS,
             (void*)&flashParams, sizeof(flashParams));
```

Example 6

```
INT nRet = IS_SUCCESS;

// Get all GPIOs that can be used as flash output
UINT nGPIOs_Flash = 0;
INT nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_SUPPORTED_GPIOS,
                 (void*)&nGPIOs_Flash, sizeof(nGPIOs_Flash));

// Get all GPIOs that can be used for the PWM
UINT nGPIOs_PWM = 0;
INT nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_SUPPORTED_GPIOS,
                 (void*)&nGPIOs_PWM, sizeof(nGPIOs_PWM));
```

Example 7

```
INT nRet = IS_SUCCESS;

// Disable flash
UINT nMode = IO_FLASH_MODE_OFF;
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_MODE, (void*)&nMode, sizeof(nMode));
```

```
// Set the flash to a constant low output
nMode = IO_FLASH_MODE_CONSTANT_LOW;
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_MODE, (void*)&nMode, sizeof(nMode));

// Set the flash to a high active pulse for each image in the trigger mode
nMode = IO_FLASH_MODE_TRIGGER_HI_ACTIVE;
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_MODE, (void*)&nMode, sizeof(nMode));

// Get the current flash mode
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_MODE, (void*)&nMode, sizeof(nMode));
```


© 2015 Thorlabs Scientific Imaging

Example 8

```
INT nRet = IS_SUCCESS;

IO_PWM_PARAMS m_pwmParams;

// Get the minimum values of the PWM parameters
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_PARAMS_MIN,
             (void*)&m_pwmParams, sizeof(m_pwmParams));
if (nRet == IS_SUCCESS)
{
    double dblFrequencyMin = m_pwmParams.dblFrequency_Hz;
    double dblDutyCycleMin = m_pwmParams.dblDutyCycle;
}

// Get the maximum values of the PWM parameters
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_PARAMS_MAX,
             (void*)&m_pwmParams, sizeof(m_pwmParams));
if (nRet == IS_SUCCESS)
{
    double dblFrequencyMax = m_pwmParams.dblFrequency_Hz;
    double dblDutyCycleMax = m_pwmParams.dblDutyCycle;
}

// Get the increment of the PWM parameters
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_PARAMS_INC,
             (void*)&m_pwmParams, sizeof(m_pwmParams));
if (nRet == IS_SUCCESS)
{
    double dblFrequencyInc = m_pwmParams.dblFrequency_Hz;
    double dblDutyCycleInc = m_pwmParams.dblDutyCycle;
}

// Get the current values of the PWM parameters
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_PARAMS,
             (void*)&m_pwmParams, sizeof(m_pwmParams));
if (nRet == IS_SUCCESS)
{
    double dblFrequency = m_pwmParams.dblFrequency_Hz;
    double dblDutyCycle = m_pwmParams.dblDutyCycle;
}

// Set the current values of the PWM parameters (1 KHz, 50% duty cycle)
m_pwmParams.dblFrequency_Hz = 1000;
m_pwmParams.dblDutyCycle = 0.5;
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_SET_PARAMS,
             (void*)&m_pwmParams, sizeof(m_pwmParams));
```

Example 9

```
INT nRet = IS_SUCCESS;

// Set GPIO1 as PWM output
UINT nMode = IO_GPIO_1;
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_SET_MODE,
             (void*)&nMode, sizeof(nMode));

// Set GPIO1, GPIO2 and the flash pin as PWM output
nMode = IO_GPIO_1 | IO_GPIO_2 | IS_FLASH_MODE_PWM;
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_SET_MODE, (void*)&nMode, sizeof(nMode));

// Get the current PWM mode
nRet = is_IO(m_hCam, IS_IO_CMD_PWM_GET_MODE, (void*)&nMode, sizeof(nMode));
```

Example 10

```
INT nRet = IS_SUCCESS;

IO_GPIO_CONFIGURATION gpioConfiguration;

// Read information about GPIO1
gpioConfiguration.u32Gpio = IO_GPIO_1;

nRet = is_IO(hCam, IS_IO_CMD_GPIOS_GET_CONFIGURATION, (void*)&gpioConfiguration,
             sizeof(gpioConfiguration) );

if (nRet == IS_SUCCESS)
{
  if ((gpioConfiguration.u32Caps & IS_GPIO_PWM) != 0)
  {
    // GPIO1 supports PWM
  }
  if ((gpioConfiguration.u32Caps & IS_GPIO_FLASH) != 0)
  {
    // GPIO1 supports Flash
  }
  if (gpioConfiguration.u32Configuration == IS_GPIO_OUTPUT)
  {
    // GPIO1 is currently configured as output
    if (gpioConfiguration.u32State == 1)
    {
      // GPIO1 is currently output HIGH
    }
  }
}
```

Example 11

```
INT nRet = IS_SUCCESS;

IO_GPIO_CONFIGURATION gpioConfiguration;

// Set configuration of GPIO1 (OUTPUT LOW)
gpioConfiguration.u32Gpio = IO_GPIO_1;
gpioConfiguration.u32Configuration = IS_GPIO_OUTPUT;
gpioConfiguration.u32State = 0;

nRet = is_IO(hCam, IS_IO_CMD_GPIOS_SET_CONFIGURATION, (void*)&gpioConfiguration,
             sizeof(gpioConfiguration));
```

Example 12

```
INT nRet = IS_SUCCESS;

IO_GPIO_CONFIGURATION gpioConfiguration;

// Set configuration of GPIO1 (COM-port TX)
// GPIO1 configured as RX is not supported!
gpioConfiguration.u32Gpio = IO_GPIO_1;
gpioConfiguration.u32Configuration = IS_GPIO_COMPORT_TX;

// GPIO2 will be configured as IS_GPIO_COMPORT_RX automatically!
nRet = is_IO(hCam, IS_IO_CMD_GPIOS_SET_CONFIGURATION, (void*)&gpioConfiguration,
             sizeof(gpioConfiguration));

// The following code leads to the same setting
// Set configuration of GPIO2 (COM-port RX)
gpioConfiguration.u32Gpio = IO_GPIO_2;
gpioConfiguration.u32Configuration = IS_GPIO_COMPORT_RX;

// GPIO1 will be configured as IS_GPIO_COMPORT_TX automatically!
nRet = is_IO(hCam, IS_IO_CMD_GPIOS_SET_CONFIGURATION, (void*)&gpioConfiguration,
             sizeof(gpioConfiguration));
```

 © 2015 Thorlabs Scientific Imaging

Example 13

```
INT nRet = IS_SUCCESS;

IO_FLASH_PARAMS flashParams;
// Get the minimum values for the GPIO flash delay and flash duration
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_GET_GPIO_PARAMS_MIN, (void*)&flashParams,
             sizeof(flashParams));

// Set the minimum values for flash delay and flash duration. Be careful: The normal flash does not wor
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_GPIO_PARAMS, (void*)&flashParams,
             sizeof(flashParams));
```

4.3.66 is_IsVideoFinish

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_IsVideoFinish (HIDS hCam, INT* pbo)
```

Description

Using `is_IsVideoFinish()`, you can check whether an image has been captured and stored completely in the image memory. This function is helpful if the `is_FreezeVideo()` function was called with the `IS_DONT_WAIT` parameter.

By setting the `*pbo==IS_CAPTURE_STATUS` parameter before calling `is_IsVideoFinish()`, you can also check whether a transfer or post-processing error occurred.

Input parameters

`hCam`	Camera handle
`pbo`	By setting `*pbo != IS_CAPTURE_STATUS` before calling the function, `pbo` contains the following digitizing status: • `IS_VIDEO_NOT_FINISH` = Digitizing of the image is not completed yet. • `IS_VIDEO_FINISH` = Digitizing of the image is completed. By setting `*pbo == IS_CAPTURE_STATUS` before calling the function, `pbo` contains the following digitizing status: • `IS_VIDEO_NOT_FINISH` = Digitizing of the image is not completed yet. • `IS_VIDEO_FINISH` = Digitizing of the image is completed. • `IS_CAPTURE_STATUS` = Transfer error or conversion problem (e.g. destination memory is invalid) The parameter `IS_CAPTURE_STATUS` replaces the previous parameter `IS_TRANSFER_FAILED.` The parameter `IS_TRANSFER_FAILED` was moved into the new header file `uc480_deprecated.h`, which contains all obsolete function definitions and constants. If necessary the header file `uc480_deprecated.h` can be included in addition to the header file `uc480.h.`

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_FreezeVideo()`

© 2015 Thorlabs Scientific Imaging

- `is_HasVideoStarted()`

4.3.67 is_LockSeqBuf

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_LockSeqBuf (HIDS hCam, INT nNum, char* pcMem)
```

Description

`is_LockSeqBuf()` locks write access to an image memory within a sequence. In the capturing process, locked image memories will be skipped in the sequence list of image memories to be used. This way, you can avoid that image data which are required for further processing will be overwritten by newly captured data. Full access to the image memory is still guaranteed. You can lock any number of image memories at the same time.

Using the `is_UnlockSeqBuf()` function, you can re-enable write access to the image memory.

Input parameters

`hCam`	Camera handle
`nNum`	Number of the image memory to be locked (`1…max`) or `IS_IGNORE_PARAMETER`: The image memory will be identified by its starting address only.
`pcMem`	Starting address of the image memory to be locked

Attention

`nNum` indicates the location in the sequence list, not the memory ID assigned using `is_AllocImageMem()`.

Return values

`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_UnlockSeqBuf()`
- `is_AddToSequence()`
- `is_SetImageMem()`
- `is_SetAllocatedImageMem()`

© 2015 Thorlabs Scientific Imaging

4.3.68 is_LUT

USB 2.0 USB 3.0 GigE	USB 2.0 USB 3.0 GigE

Syntax

```
INT is_LUT(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParams);
```

Description

Using `is_LUT()`, you can enable a hardware or software LUT for uEye cameras. This LUT which will be applied to the image in the camera. A number of predefined LUTs are available. Alternatively, you define your own LUT. It is possible to define a LUT without enabling it at the same time. You can also query the current LUT used by the camera.

Each lookup table (LUT) for the uEye contains modification values for the image brightness and contrast parameters. When a LUT is used, each brightness value in the image will be replaced by a value from the table. LUTs are typically used to enhance the image contrast or the gamma curve. The values must be in the range between 0.0 and 1.0. A linear LUT containing 64 equidistant values between 0.0 and 1.0 has no effect on the image.

For further information on LUTs, please refer to the LUT properties section.

Note: The following functions are obsolete by the `is_LUT()` function:

- `is_GetCameraLUT()`
- `is_SetCameraLUT()`

See also: Obsolete functions

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	

IS_LUT_CMD_SET_ENABLED	Enables/disables the LUT (Example 1): • IS_LUT_DISABLED: LUT is disabled. • IS_LUT_ENABLED: LUT is enabled. If no other LUT is defined the linear LUT IS_CAMERA_LUT_IDENTITY is set.
IS_LUT_CMD_SET_MODE	Sets the calculation mode (Example 2): • IS_LUT_MODE_ID_DEFAULT: Default mode: If supported the LUT/gamma is done on hardware otherwise in software • IS_LUT_MODE_ID_FORCE_HARDWARE: LUT/gamma in hardware (if not possible: IS_LUT_STATE_ID_NOT_SUPPORTED) • IS_LUT_MODE_ID_FORCE_SOFTWARE: LUT/gamma in software (if not possible: IS_LUT_STATE_ID_NOT_SUPPORTED)
IS_LUT_CMD_GET_STATE	Returns the current status information on the LUT (Example 3): • IS_LUT_STATE_ID_INACTIVE: LUT is inactive (No gamma and no LUT is set) • IS_LUT_STATE_ID_NOT_SUPPORTED: LUT is not supported with the current settings • IS_LUT_STATE_ID_HARDWARE_LUT: LUT is calculated in hardware • IS_LUT_STATE_ID_HARDWARE_GAMMA: Gamma is calculated in hardware • IS_LUT_STATE_ID_HARDWARE_LUTANDGAMMA: LUT and gamma are calculated in hardware • IS_LUT_STATE_ID_SOFTWARE_LUT: LUT is calculated in software • IS_LUT_STATE_ID_SOFTWARE_GAMMA: Gamma is calculated in software • IS_LUT_STATE_ID_SOFTWARE_LUTANDGAMMA: LUT and gamma are calculated in software
IS_LUT_CMD_GET_SUPPORT_INFO	Returns the current support information of the LUT (Example 3)
IS_LUT_CMD_SET_USER_LUT	Sets the user-defined values of the LUT (Example 4)
IS_LUT_CMD_GET_USER_LUT	Returns the user-defined values of the LUT (unchanged) (Example 5)
IS_LUT_CMD_GET_COMPLETE_LUT	Returns the user-defined values of the LUT after the values for gamma, contrast and brightness are included (Example 5)
IS_LUT_CMD_GET_PRESET_LUT	Returns the predefined LUT (Example 6): • IS_LUT_PRESET_ID_IDENTITY: linear LUT, no image modifications • IS_LUT_PRESET_ID_NEGATIVE: inverts the image • IS_LUT_PRESET_ID_GLOW1: false-color display of the image • IS_LUT_PRESET_ID_GLOW2: false-color display of the image

pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the IS_LUT_CONFIGURATION_PRESET_64 structure

IS_LUT_PRESET	predefinedLutID	ID of the predefined LUT
IS_LUT_CONFIGURATION_64	lutConfiguration	Predefined LUT

Contents of the IS_LUT_CONFIGURATION_64 structure

DOUBLE	dblValues[3][IS_LUT_64]	Defines a LUT with 64 knee points. This results in 32 sections with a start and end point each.
BOOL	bAllChannelsAreEqual	If TRUE, the same LUT is applied to all three channels.

Contents of the IS_LUT_STATE structure

BOOL	bLUTEnabled	LUT is enabled
INT	nLUTStateID	ID of the LUT status information
INT	nLUTModeID	ID of the LUT mode
INT	nLUTBits	Used bits of the LUT

Contents of the IS_LUT_SUPPORT_INFO structure

BOOL	bSupportLUTHardware	Hardware LUT is supported
BOOL	bSupportLUTSoftware	Software LUT is supported
INT	nBitsHardware	Used bits of the hardware LUT
INT	nBitsSoftware	Used bits of the software LUT
INT	nChannelsHardware	Supported channels for hardware LUT
INT	nChannelsSoftware	Supported channels for software LUT

Return values

IS_FILE_READ_OPEN_ERROR	The file cannot be opened.
IS_FILE_WRITE_OPEN_ERROR	File cannot be opened for writing or reading.
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Example 1

```
/* Enable the last set LUT */
IS_LUT_ENABLED_STATE nLutEnabled = IS_LUT_ENABLED;
INT nRet = is_LUT(hCam, IS_LUT_CMD_SET_ENABLED, (void*) &nLutEnabled, sizeof(nLutEnabled));
```

Example 2

```
/* Force LUT and gamma to be included in software (former gamma behavior for cameras with USB3) */
IS_LUT_MODE nLutMode = IS_LUT_MODE_ID_FORCE_SOFTWARE;
INT nRet = is_LUT(hCam, IS_LUT_CMD_SET_MODE, &nLutMode , sizeof(nLutMode));
```

Example 3

```
/* Readout the current LUT state */
IS_LUT_STATE lutState;
INT nRet = is_LUT(hCam, IS_LUT_CMD_GET_STATE, (void*) &lutState, sizeof(lutState));

/* Readout the current LUT support information */
IS_LUT_SUPPORT_INFO lutSupportInfo;
nRet = is_LUT(hCam, IS_LUT_CMD_GET_SUPPORT_INFO, (void*) &lutSupportInfo, sizeof(lutSupportInfo));
```

Example 4

```
/* Set user-defined LUT */
IS_LUT_CONFIGURATION_64 userdefinedLUT;
...
... setze Werte in userdefinedLUT.dblValues[][] ...
...
INT nRet = is_LUT(hCam, IS_LUT_CMD_SET_USER_LUT, (void*) &userdefinedLUT, sizeof(userdefinedLUT));
```

Example 5

```
/* Readout of the set LUT */
/* User-defined LUT (without included gamma) */
IS_LUT_CONFIGURATION_64 userLUT;
INT nRet = is_LUT(hCam, IS_LUT_CMD_GET_USER_LUT, (void*) &userLUT, sizeof(userLUT));

/* Complet LUT (with included gamma) */
IS_LUT_CONFIGURATION_64 completeLUT;
nRet = is_LUT(hCam, IS_LUT_CMD_GET_COMPLETE_LUT, (void*) &completeLUT, sizeof(completeLUT));
```

Example 6

```
/* Set predefined LUT "Glow1" */
/* Firstly load GLOW1 */
IS_LUT_CONFIGURATION_PRESET_64 presetLUT;
presetLUT.predefinedLutID = IS_LUT_PRESET_ID_GLOW1;
INT nRet = is_LUT(hCam, IS_LUT_CMD_GET_PRESET_LUT, (void*) &presetLUT, sizeof(presetLUT));

/* Secondly set GLOW1 */
if (IS_SUCCESS == nRet)
{
     nRet = is_LUT(hCam, IS_LUT_CMD_SET_USER_LUT, (void*) &presetLUT, sizeof(presetLUT));
}
```

Example 7

```
/* Load the "lutFile.xml" file and set LUT */
wchar_t* pFilename = L"lutFile.xml";
INT nRet = is_LUT(hCam, IS_LUT_CMD_LOAD_FILE, (void*) pFilename , NULL);

/* Save the current set LUT into the "lutFile2.xml" file */
wchar_t* pFilename2 = L"lutFile2.xml";
nRet = is_LUT(hCam, IS_LUT_CMD_SAVE_FILE, (void*) pFilename2 , NULL);
```

4.3.69 is_Measure

USB 2.0 USB 3.0	USB 2.0 USB 3.0

© 2015 Thorlabs Scientific Imaging

Syntax

```
INT is_Measure(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

The function allows the measurement of the sharpness in a defined AOI of the current image. To get a sharpness value the edges in the image are evaluated. The sharpness can only be indicated as a relative value as it depends on the edges in the current image. An image with less edges will reach the sharpness value of an image with a lot of edges.

The higher the value, the better the sharpness. The value can be used in comparative measurements to detect changes in the image acquisition of the same object, e.g. caused by readjusted lenses.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	
`IS_MEASURE_CMD_SHARPNESS_AOI_SET`	Sets an AOI in which the sharpness is measured. In the image are up to 5 AOIs possible. These AOIs can also overlap. (Example 1)
`IS_MEASURE_CMD_SHARPNESS_AOI_INQUIRE`	Returns information of the AOI, e.g. the sharpness(Example 2)
`IS_MEASURE_CMD_SHARPNESS_AOI_SET_PRESET`	Sets different predefined AOIs in the image (Example 3)
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Content of the MEASURE_SHARPNESS_AOI_INFO structure

`UINT`	`u32NumberAOI`	ID of the AOI
`UINT`	`u32SharpnessValue`	Relative sharpness value in the defined AOI
`IS_RECT`	`rcAOI`	Position and size of the AOI: • `s32X`: X position • `s32Y`: Y position • `s32Width`: AOI width • `s32Height`: AOI height

Content of the MEASURE_SHARPNESS_AOI_PRESETS enumeration

`IS_MEASURE_SHARPNESS_AOI_PRESET_1`	Predefined AOI for the harpness measurement (in each of the four image corner and in the center)

Return values

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Example 1

```
INT nRet = IS_SUCCESS;

/* Create info object */
MEASURE_SHARPNESS_AOI_INFO measureSharpnessInfo;

/* Set values of AOI_0: The position and size of the AOI equals the whole image */
measureSharpnessInfo.u32NumberAOI    = 0;
measureSharpnessInfo.rcAOI.s32X      = 0;
measureSharpnessInfo.rcAOI.s32Y      = 0;
measureSharpnessInfo.rcAOI.s32Width  = m_s32MaxImageWidth;
measureSharpnessInfo.rcAOI.s32Height = m_s32MaxImageHeight;

/* Set AOI_0 */
nRet = is_Measure(m_hCam, IS_MEASURE_CMD_SHARPNESS_AOI_SET, (void*)&measureSharpnessInfo,
                  sizeof(measureSharpnessInfo));

/* Set values of AOI_1 */
measureSharpnessInfo.u32NumberAOI    = 1;
measureSharpnessInfo.rcAOI.s32X      = 50;
measureSharpnessInfo.rcAOI.s32Y      = 50;
measureSharpnessInfo.rcAOI.s32Width  = 200;
measureSharpnessInfo.rcAOI.s32Height = 200;

/* Set AOI_1 */
nRet = is_Measure(m_hCam, IS_MEASURE_CMD_SHARPNESS_AOI_SET, (void*)&measureSharpnessInfo,
                  sizeof(measureSharpnessInfo));
```

Example 2

```
INT nRet = IS_SUCCESS;

/* Create info object */
MEASURE_SHARPNESS_AOI_INFO measureSharpnessInfo;

/* Get values of AOI_0 */
measureSharpnessInfo.u32NumberAOI = 0;

nRet = is_Measure(m_hCam, IS_MEASURE_CMD_SHARPNESS_AOI_INQUIRE, (void*)&measureSharpnessInfo,
                  sizeof(measureSharpnessInfo));
if (nRet == IS_SUCCESS)
{
    UINT s32Sharpness = measureSharpnessInfo.u32SharpnessValue;
}
```

Example 3

 © 2015 Thorlabs Scientific Imaging

```
INT nRet = IS_SUCCESS;

/* Set preset */
UINT nPreset = IS_MEASURE_SHARPNESS_AOI_PRESET_1;

nRet = is_Measure(m_hCam, IS_MEASURE_CMD_SHARPNESS_AOI_SET_PRESET, (void*)&nPreset,
                  sizeof(nPreset));
```

4.3.70 is_ParameterSet

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ParameterSet(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

The `is_ParameterSet()` function saves the current camera parameters to a file or to the EEPROM of the camera and loads the parameter set from a file or the EEPROM.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Only camera-specific ini files can be loaded. The uc480_parameter_file section in the appendix describes the structure of a uc480 ini file.

Attention

When loading an ini file, make sure that the image size (AOI) and color depth parameters in the ini file match those in the allocated memory. Otherwise, display errors may occur.

Note

The following functions are obsolete by the `is_ParameterSet()` function:

- `is_SaveParameters()`
- `is_LoadParameters()`

See also: Obsolete functions

Input parameter

`hCam`	Camera handle
⊟ `nCommand`	

© 2015 Thorlabs Scientific Imaging

IS_PARAMETERSET_CMD_LOAD_EEPROM	Loads a camera parameter set from the EEPROM (Example 1) The parameter sets in the EEPROM of the camera can be loaded via special file names: • \\cam\\set1 oder /cam/set1
IS_PARAMETERSET_CMD_LOAD_FILE	Loads a camera parameter set from a file (Example 2) You must pass the path to the ini file as Unicode string. You can pass either a relative or an absolute path. If you pass NULL the "Open file" dialog opens.
IS_PARAMETERSET_CMD_SAVE_EEPROM	Saves a camera parameter set in the EEPROM (Example 3) The parameter sets in the EEPROM of the camera can be saved with special file names: • \\cam\\set1 oder /cam/set1
IS_PARAMETERSET_CMD_SAVE_FILE	Saves a camera parameter set in a file (Example 4) You must pass the path to the ini file as Unicode string. You can pass either a relative or an absolute path. If you pass NULL the "Save as" dialog opens.
IS_PARAMETERSET_CMD_GET_NUMBER_SUPPORTED	Returns the number of supported parameter sets in the camera EEPROM (Example 5). At the moment this is "1" for all cameras.

pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Return values

IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_CameraStatus()

Example 1

```
INT nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_LOAD_EEPROM, NULL, NULL);
```

Example 2

```
// Load parameters from file (open filebox)
INT nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_LOAD_FILE, NULL, NULL);

// Load parameters from specified file
nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_LOAD_FILE, L"file.ini", NULL);
```

Example 3

```
INT nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_SAVE_EEPROM, NULL, NULL);
```

Example 4

```
// Save parameters to file (open filebox)
INT nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_SAVE_FILE, NULL, NULL);

// Save parameters to specified file
nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_SAVE_FILE, L"file.ini", NULL);
```

Example 5

```
// Get the number of supported parameter sets in the camera EEPROM
UINT nNumber;
INT nRet = is_ParameterSet(m_hCam, IS_PARAMETERSET_CMD_GET_NUMBER_SUPPORTED, (void*)
                           &nNumber, sizeof(nNumber));
```


© 2015 Thorlabs Scientific Imaging

4.3.71 is_PixelClock

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_PixelClock(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

The function returns the adjustable pixel clock range sets the piel clock. Due to an excessive pixel clock for USB cameras, images may get lost during the transfer. If you change the pixel clock on-the-fly, the current image capturing process will be aborted.

The pixel clock limit values can vary, depending on the camera model and operating mode. For detailed information on the pixel clock range of a specific camera model, please refer to the Camera and sensor data chapter.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

Note

Note: The following functions are obsolete by the `is_PixelClock()` function:

- `is_SetPixelClock()`
- `is_GetPixelClockRange()`

See also: Obsolete functions

Input parameter

`hCam`	Camera handle
⊟ `nCommand`	
`IS_PIXELCLOCK_CMD_GET_NUMBER`	Returns the number of discrete pixel clock which are supported by the camera (Example 1).
`IS_PIXELCLOCK_CMD_GET_LIST`	Returns the list with discrete pixel clocks.
`IS_PIXELCLOCK_CMD_GET_RANGE`	Returns the range for the pixel clock (Example 2)
`IS_PIXELCLOCK_CMD_GET_DEFAULT`	Returns the default pixel clock (Example 4)
`IS_PIXELCLOCK_CMD_GET`	Returns the current set pixel clock in MHz (Example 3)
`IS_PIXELCLOCK_CMD_SET`	Sets the pixel clock in MHz (Example 4)
`pParam`	Pointer to a function parameter, whose function depends on `nCommand`.
`cbSizeOfParam`	Size (in bytes) of the memory area to which `pParam` refers.

Return values

`IS_INVALID_MODE`	Camera is in standby mode, function not allowed

IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_GetFramesPerSecond()
- is_GetFrameTimeRange()
- is_Exposure()
- is_SetOptimalCameraTiming()
- is_SetFrameRate()
- is_SetAutoParameter()
- is_SetBinning()
- is_SetSubSampling()
- is_AOI()

Example 1

```
UINT nNumberOfSupportedPixelClocks = 0;
INT nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_GET_NUMBER,
                         (void*)&nNumberOfSupportedPixelClocks,
                         sizeof(nNumberOfSupportedPixelClocks));
if ((nRet == IS_SUCCESS) && (m_nNumberOfSupportedPixelClocks > 0))
{
    // No camera has more than 150 different pixel clocks.
    // Of course, the list can be allocated dynamically
    UINT nPixelClockList[150];
    ZeroMemory(&nPixelClockList, sizeof(nPixelClockList));

    nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_GET_LIST,
                         (void*)nPixelClockList,
                         nNumberOfSupportedPixelClocks * sizeof(UINT));
}
```

Example 2

```
UINT nRange[3];
ZeroMemory(nRange, sizeof(nRange));

// Get pixel clock range
INT nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_GET_RANGE, (void*)nRange, sizeof(nRange));
if (nRet == IS_SUCCESS)
{
    UINT nMin = nRange[0];
    UINT nMax = nRange[1];
    UINT nInc = nRange[2];
}
```

Example 3

```
UINT nPixelClock;

// Get current pixel clock
nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_GET, (void*)&nPixelClock, sizeof(nPixelClock));
```

 © 2015 Thorlabs Scientific Imaging

Example 4

```
UINT nPixelClockDefault;

// Get default pixel clock
INT nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_GET_DEFAULT,
                         (void*)&nPixelClockDefault, sizeof(nPixelClockDefault));
if (nRet == IS_SUCCESS)
{
    // Set this pixel clock
    nRet = is_PixelClock(hCam, IS_PIXELCLOCK_CMD_SET,
                         (void*)&nPixelClockDefault, sizeof(nPixelClockDefault));
}
```

4.3.72 is_ReadEEPROM

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ReadEEPROM (HIDS hCam, INT Adr, char* pcString, INT Count)
```

Description

Using `is_ReadEEPROM()`, you can read the contents of the camera EEPROM. Besides the hard-coded factory information, the EEPROM of the DCx Camera can hold 64 bytes of user data.

Input parameters

`hCam`	Camera handle
`Adr`	Starting address for data reads Value range: `0...63`
`pcString`	Pointer to the buffer for the data to read (min. size = `Count`)
`Count`	Number of characters to read

Return values

`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_CRC_ERROR`	A CRC error-correction problem occurred while reading the settings.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_WriteEEPROM()`

© 2015 Thorlabs Scientific Imaging

Example

```
char buffer[64];
is_ReadEEPROM( hCam, 0x00, buffer, 64 );
```

4.3.73 is_RenderBitmap

USB 2.0 USB 3.0	-

Syntax

```
INT is_RenderBitmap (HIDS hCam, INT nMemID, HWND hwnd, INT nMode)
```

Description

Using `is_RenderBitmap()`, you can output an image from an image memory in the specified window. For the display, Windows bitmap functionality is used. The image is displayed in the format you specified when allocating the image memory.

The `bitspixel` parameter of the `is_AllocImageMem()` function defines the color depth and display type. RGB16 and RGB15 require the same amount of memory but can be distinguished by the `bitspixel` parameter.

Attention

`is_RenderBitmap()` can render Y8 and RGB formats. For displaying YUV/YCbCr formats please use the `is_DirectRenderer()` function (see also Color and memory formats).

Input parameters

hCam	Camera handle
nMemID	ID of the image memory whose contents is to be displayed
hwnd	Output window handle
nMode	
IS_RENDER_NORMAL	The image is rendered normally. It will be displayed in 1:1 scale as stored in the image memory.
IS_RENDER_FIT_TO_WINDOW	The image size is adjusted to fit the output window.
IS_RENDER_DOWNSCALE_1_2	Displays the image at 50 % of its original size.
IS_RENDER_PLANAR_COLOR_RED	Renders the red color component of the planar format in red.
IS_RENDER_PLANAR_COLOR_GREEN	Renders the green color component of the planar format in green.
IS_RENDER_PLANAR_COLOR_BLUE	Renders the blue color component of the planar format in blue.
IS_RENDER_PLANAR_MONO_RED	Renders the red color component of the planar format in gray shades.
IS_RENDER_PLANAR_MONO_GREEN	Renders the green color component of the planar format in gray shades.
IS_RENDER_PLANAR_MONO_BLUE	Renders the blue color component of the planar format in gray shades.
The following options can be linked by a logical OR using the nMode parameter:	

© 2015 Thorlabs Scientific Imaging

IS_RENDER_MIRROR_UPDOWN	Mirrors the displayed image along the horizontal axis.

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- `is_AllocImageMem()`
- `is_SetColorMode()`
- `is_SetDisplayMode()`
- `is_DirectRenderer()`

Example

Fit image to window and display it upside down:

```
is_RenderBitmap (hCam, nMemID, hwnd, IS_RENDER_FIT_TO_WINDOW | IS_RENDER_MIRROR_UPDOWN);
```

Sample programs

- SimpleAcquire (C++)
- SimpleLive (C++)

4.3.74 is_ResetToDefault

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_ResetToDefault (HIDS hCam)
```

Description

`is_ResetToDefault()` resets all parameters to the camera-specific defaults as specified by the driver. By default, the camera uses full resolution, a medium speed and color level gain values adapted to daylight exposure.

Input parameters

`hCam`	Camera handle

Return values

`IS_CANT_ADD_TO_SEQUENCE`	The image memory is already included in the sequence and cannot be added again.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_DR_CANNOT_CREATE_SURFACE`	The image surface or overlay surface could not be created.
`IS_DR_CANNOT_CREATE_TEXTURE`	The texture could not be created.
`IS_DR_CANNOT_CREATE_VERTEX_BUFFER`	The vertex buffer could not be created.
`IS_DR_DEVICE_OUT_OF_MEMORY`	Not enough graphics memory available.
`IS_DR_LIBRARY_NOT_FOUND`	The DirectRenderer library could not be found.
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAPTURE_MODE`	The function can not be executed in the current camera operating mode (free run, trigger or standby).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.

© 2015 Thorlabs Scientific Imaging

IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_ParameterSet()

4.3.75 is_SetAllocatedImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetAllocatedImageMem (HIDS hCam, INT width, INT height, INT bitspixel,
                             char* pcImgMem, int* pid)
```

Description

Using `is_SetAllocatedImageMem()`, you can make a memory allocated by a user the active memory for storing digitized images in it. The allocated memory must be large enough and must always be locked globally.

Depending on the selected image format you need more than one byte per pixel for image memory:

```
unsigned int uBytesPerPixel = bitspixel/8;
if (uImageSize % bitspixel != 0)
{
    uBytesPerPixel++;
}
unsigned int uImageSize = width * height * uBytesPerPixel;
```

You can call the `is_AddToSequence()` function to add a memory which was set using `is_SetAllocatedImageMem()` to a sequence.

The address of this memory will be passed to the uc480 driver. For this, you can use the `is_SetAllocatedImageMem()` function. In addition, you need to specify the image size, just as you do when calling `is_AllocImageMem()`. The returned memory ID is required by other functions for memory access.

The memory area must be removed from the driver management again using the `is_FreeImageMem()` function. Please note that this does not release the memory. You then need to make sure that the memory will be released again.

After `is_SetAllocatedImageMem` you must call `is_SetImageMem` or `is_AddToSequence` in order that the image caption can be carried out in the image memory.

Input parameters

hCam	Camera handle
width	Image width
height	Image height
bitspixel	Image color depth (bits per pixel)
pcImgMem	Pointer to the starting address of the allocated memory
pid	Returns the ID of this memory.

Return values

IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid

 © 2015 Thorlabs Scientific Imaging

	range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully

Related functions

- is_AllocImageMem()
- is_FreeImageMem()
- is_AddToSequence()
- is_SetImageMem()
- is_GetColorDepth()
- is_GetImgMemPitch()

Example Windows

```
HANDLE hMem = GlobalAlloc(0, uImageSize);
char* pcMem = (char*)GlobalLock(hMem);
INT nRet = is_SetAllocatedImageMem(hCam, uWidth, uHeight, uBitspixel, pcMem, &iMemID);
[…]
nRet = is_FreeImageMem(hCam, pcMem, iMemID);
GlobalUnlock(hMem);
GlobalFree(hMem);
```

Example Linux

```
char* pcMem = (char*)malloc(uImageSize);
int iRet = mlock(pcMem, uImageSize);
INT nRet = is_SetAllocatedImageMem(hCam, uWidth, uHeight, uBitspixel, pcMem, &iMemID);
[…]
nRet = is_FreeImageMem(hCam, pcMem, iMemID);
iRet = munlock(pcMem, uImageSize);
free(pcMem);
```


© 2015 Thorlabs Scientific Imaging

4.3.76 is_SetAutoParameter

USB 2.0 USB 3.0 GigE	USB 2.0 USB 3.0 GigE

Syntax

```
INT is_SetAutoParameter (HIDS hCam, INT param, double* pval1, double* pval2)
```

Description

Using `is_SetAutoParameter()`, you can control the automatic gain, exposure shutter, frame rate and white balance control values.

For further information on automatic control, please refer to the Automatic image control chapter.

- Control is only active as long as the camera is capturing images.
- A manual change of the exposure time and gain settings disables the auto functions.
- When the **auto exposure shutter** function is enabled, you cannot modify the pixel clock frequency.
- The **auto frame rate** function is only available when the auto shutter control is on. Auto frame rate and auto gain control cannot be used simultaneously.
- The **auto gain** function can only be used for cameras with master gain control. Auto white balance is only available for cameras with hardware RGB gain control.
- The **sensor's internal auto features** are not supported by the sensors of DCx camera models.

Attention

Automatic controls when using very high frame rates

Using very high frame rates can cause that too many control commands are sent to the camera. When using frame rates higher than 100 fps you should increase the value for `IS_SET_AUTO_SKIPFRAMES`. Thus, less image will be used for the automatic controls which takes load off the camera.

Input parameters

`hCam`	Camera handle
`param`	Configure auto function
⊟ Enabling auto functions and querying the status	
`IS_SET_ENABLE_AUTO_GAIN`	Enables/disables the auto gain control function ⊟ Control parameters `pval1` = 1 enables control, 0 disables control
`IS_GET_ENABLE_AUTO_GAIN`	Returns the current auto gain setting or white level adjustment ⊟ Control parameters `pval1`: returns the current setting
`IS_SET_ENABLE_AUTO_SENSOR_GAIN`	Enables/disables the internal auto gain control function of the sensor*1 ⊟ Control parameters

hCam	Camera handle
	pval1 = 1 enables control, 0 disables control
IS_GET_ENABLE_AUTO_SENSOR_GAIN	Returns the current auto gain setting of the sensor*1 Control parameters pval1: returns the current setting
IS_SET_ENABLE_AUTO_SHUTTER	Enables/disables the auto exposure shutter function. Control parameters pval1 = 1 enables control, 0 disables control
IS_GET_ENABLE_AUTO_SHUTTER	Returns the current auto exposure shutter setting. Control parameters pval1: returns the current setting
IS_SET_ENABLE_AUTO_SENSOR_SHUTTER	Enables/disables the sensor's internal auto exposure shutter function*1 Control parameters pval1 = 1 enables control, 0 disables control
IS_GET_ENABLE_AUTO_SENSOR_SHUTTER	Returns the sensor's current auto exposure shutter setting*1 Control parameters pval1: returns the current setting
IS_SET_ENABLE_AUTO_WHITEBALANCE	Enables/disables the auto white balance function. Control parameters pval1 = 1 enables control, 0 disables control
IS_GET_ENABLE_AUTO_WHITEBALANCE	Returns the current auto white balance setting. Control parameters pval1: returns the current setting
IS_SET_ENABLE_AUTO_SENSOR_WHITEBALANCE	Enables/disables the sensor's internal auto white balance function*1 Control parameters pval1: white balance mode (see is_GetAutoInfo()): • WB_MODE_DISABLE • WB_MODE_AUTO • WB_MODE_ALL_PULLIN • WB_MODE_INCANDESCENT_LAMP • WB_MODE_FLUORESCENT_DL • WB_MODE_OUTDOOR_CLEAR_SKY • WB_MODE_OUTDOOR_CLOUDY
IS_GET_ENABLE_AUTO_SENSOR_WHITEBALANCE	Returns the sensor's current auto white balance setting*1 Control parameters pval1: returns the current setting
IS_SET_ENABLE_AUTO_FRAMERATE	Enables/disables the auto frame rate function.

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	⊟ Control parameters `pval1` = 1 enables control, 0 disables control
`IS_GET_ENABLE_AUTO_FRAMERATE`	Returns the current auto frame rate setting. ⊟ Control parameters `pval1`: returns the current setting
`IS_SET_ENABLE_AUTO_SENSOR_FRAMERATE`	Enables/disables the sensor's internal auto frame rate function*1 ⊟ Control parameters `pval1` = 1 enables control, 0 disables control
`IS_GET_ENABLE_AUTO_SENSOR_FRAMERATE`	Returns the sensor's current auto frame rate setting*1 ⊟ Control parameters `pval1`: returns the current setting
⊟ Adjusting auto gain control/auto exposure shutter	
`IS_SET_AUTO_REFERENCE`	Sets the setpoint for auto gain control/auto exposure shutter. ⊟ Control parameters `pval1`: defines the setpoint (average brightness of the image); the following rule applies independently of the image bit depth: • 0 = black • 128 = 50% gray (default) • 255 = white ⊟ Note on the sensor's internal control functionality When using the sensor's internal control functionality, you can only used values in a range between [44...235]. The increment in this range is 4. Smaller values are automatically set to 44, larger values to 235.
`IS_GET_AUTO_REFERENCE`	Returns the set point for auto gain control/auto exposure shutter. ⊟ Control parameters `pval1`: returns the current setting
`IS_SET_AUTO_GAIN_MAX`	Sets the upper limit for auto gain control. ⊟ Control parameters `pval1`: valid value for gain (0...100)
`IS_GET_AUTO_GAIN_MAX`	Returns the upper limit for auto gain control. ⊟ Control parameters `pval1`: returns the current setting
`IS_SET_AUTO_SHUTTER_MAX`	Sets the upper limit for auto exposure shutter. ⊟ Control parameters `pval1`: valid exposure value (0 sets the value continuously to max. exposure)

hCam	Camera handle
IS_GET_AUTO_SHUTTER_MAX	Returns the upper limit for auto exposure shutter. Control parameters pval1: returns the current setting
IS_SET_AUTO_BRIGHTNESS_ONCE	Enables/disables the automatic disable for automatic brightness control (gain and exposure time)*2 Control parameters pval1 = 1 enables control, 0 disables control
IS_GET_AUTO_BRIGHTNESS_ONCE	Returns the automatic disable status*2 Control parameters pval1: returns the current setting
Speed and hysteresis	
IS_SET_AUTO_SPEED	Sets the speed value for the auto function. Control parameters pval1: defines the control speed (0...100)
IS_GET_AUTO_SPEED	Returns the speed value for the auto function. Control parameters pval1: returns the current setting
IS_SET_AUTO_SKIPFRAMES	Sets the number of frames to be skipped during automatic control. Control parameters pval1: defines the number of frames to be skipped during automatic control (default: 4)
IS_GET_AUTO_SKIPFRAMES	Returns the number of frames to be skipped during automatic control. Control parameters pval1: returns the current setting
IS_GET_AUTO_SKIPFRAMES_RANGE	Returns the permissible range for the number of frames to be skipped. Control parameters • pval1: returns the minimum permitted value • pval2: returns the maximum permitted value
IS_SET_AUTO_HYSTERESIS	Sets the hysteresis value for auto exposure shutter and auto gain control. Control parameters pval1: defines the hysteresis value (default: 2)
IS_GET_AUTO_HYSTERESIS	Returns the hysteresis value for auto exposure shutter and auto gain control. Control parameters pval1: returns the current setting
IS_GET_AUTO_HYSTERESIS_RANGE	Returns the permissible range for the hysteresis value. Control parameters • pval1: returns the minimum permitted value

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	• pval2: returns the maximum permitted value
Photometric settings for auto gain control/auto exposure shutter	
IS_SET_SENS_AUTO_SHUTTER_PHOTOM	Sets the photometry mode for auto exposure shutter. Control parameters pval1: defines which fields of view are used for auto exposure shutter (see is_GetAutoInfo()): • AS_PM_NONE • AS_PM_SENS_CENTER_WEIGHTED • AS_PM_SENS_CENTER_SPOT • AS_PM_SENS_PORTRAIT • AS_PM_SENS_LANDSCAPE
IS_GET_SENS_AUTO_SHUTTER_PHOTOM	Returns the photometry mode for auto exposure shutter. Control parameters pval1: returns the current setting
IS_GET_SENS_AUTO_SHUTTER_PHOTOM_DEF	Returns the default photometry mode for auto exposure shutter. Control parameters pval1: returns the default
IS_SET_SENS_AUTO_GAIN_PHOTOM	Sets the photometry mode for auto gain control. Control parameters pval1: defines which fields of view are used for auto gain control (see is_GetAutoInfo()): • AG_PM_NONE • AG_PM_SENS_CENTER_WEIGHTED • AG_PM_SENS_CENTER_SPOT • AG_PM_SENS_PORTRAIT • AG_PM_SENS_LANDSCAPE
IS_GET_SENS_AUTO_GAIN_PHOTOM	Returns the photometry mode for auto gain control. Control parameters pval1: returns the current setting
IS_GET_SENS_AUTO_GAIN_PHOTOM_DEF	Returns the default photometry mode for auto gain control. Control parameters pval1: returns the default
Adjusting auto white balance	
IS_SET_AUTO_WB_OFFSET	Sets the offset values for the red and blue channels. Control parameters • pval1: defines the red level offset (-50...50) • pval2: defines the blue level offset (-50...50)
IS_GET_AUTO_WB_OFFSET	Returns the offset values for the red and blue

hCam	Camera handle
	channels. Control parameters • pval1: returns the red level offset (-50...50) • pval2: returns the blue level offset (-50...50)
IS_SET_AUTO_WB_GAIN_RANGE	Sets the color gain limits for auto white balance. Control parameters • pval1: sets the lowest gain value • pval2: sets the highest gain value
IS_GET_AUTO_WB_GAIN_RANGE	Returns the color gain limits for auto white balance. Control parameters • pval1: returns the minimum permitted gain value • pval2: returns the maximum permitted gain value
IS_SET_AUTO_WB_ONCE	Sets automatic disabling of auto white balance*2 Control parameters pval1 = 1 enables control, 0 disables control
IS_GET_AUTO_WB_ONCE	Returns the automatic disable status*2 Control parameters pval1: returns the current setting
Speed and hysteresis	
IS_SET_AUTO_WB_SPEED	Sets the speed for auto white balance. Control parameters pval1: defines the control speed (0...100)
IS_GET_AUTO_WB_SPEED	Returns the speed for auto white balance. Control parameters pval1: returns the current setting
IS_SET_AUTO_WB_HYSTERESIS	Sets the hysteresis value for auto white balance. Control parameters pval1: defines the hysteresis value (default: 2)
IS_GET_AUTO_WB_HYSTERESIS	Returns the hysteresis value for auto white balance. Control parameters pval1: returns the current setting
IS_GET_AUTO_WB_HYSTERESIS_RANGE	Returns the permissible range for the hysteresis value. Control parameters • pval1: returns the minimum permitted value • pval2: returns the maximum permitted value
IS_SET_AUTO_WB_SKIPFRAMES	Sets the number of frames to be skipped during automatic control. Control parameters

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
	pval1: defines the number (default: 4)
IS_GET_AUTO_WB_SKIPFRAMES	Returns the number of frames to be skipped during automatic control. Control parameters pval1: returns the current setting
IS_GET_AUTO_WB_SKIPFRAMES_RANGE	Returns the permissible range for the number of frames to be skipped. Control parameters • pval1: returns the minimum permitted value • pval2: returns the maximum permitted value
Default values for auto gain control/auto exposure shutter	
NULL is passed for the pval1, pval2 parameters.	
IS_DEFAULT_AUTO_BRIGHT_REFERENCE	Default setpoint for auto gain control and auto exposure shutter.
IS_MIN_AUTO_BRIGHT_REFERENCE	Minimum setpoint for auto gain control and auto exposure shutter.
IS_MAX_AUTO_BRIGHT_REFERENCE	Maximum setpoint for auto gain control and auto exposure shutter.
IS_DEFAULT_AUTO_SPEED	Default value for auto speed.
IS_MAX_AUTO_SPEED	Maximum value for auto speed
Default values for Auto White Balance	
NULL is passed for the pval1, pval2 parameters.	
IS_MIN_WB_OFFSET	Minimum value for auto white balance offset.
IS_MAX_WB_OFFSET	Maximum value for auto white balance offset.
IS_DEFAULT_AUTO_WB_SPEED	Default value for auto white balance speed.
IS_MIN_AUTO_WB_SPEED	Minimum value for auto white balance speed.
IS_MAX_AUTO_WB_SPEED	Maximum value for auto white balance speed.
pval1	Control parameter, can have a variable value depending on the corresponding auto function
pval2	Control parameter, can have a variable value depending on the corresponding auto function

*1 Not all sensors support this feature (see information box)

*2 Not with use of sensor's internal control functionality

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INCOMPATIBLE_SETTING	Because of other incompatible settings the function is not possible.

IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_INVALID_WB_BINNING_MODE	Mono binning/mono sub-sampling do not support automatic white balance.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_ParameterSet()
- is_GetAutoInfo()
- is_SetHardwareGain()
- is_SetHWGainFactor()
- is_Exposure()
- is_SetFrameRate()
- is_AOI()

Examples

```
//Enable auto gain control:
double dEnable = 1;
int ret = is_SetAutoParameter (hCam, IS_SET_ENABLE_AUTO_GAIN, &dEnable, 0);

//Set brightness setpoint to 128:
double nominal = 128;
int ret = is_SetAutoParameter (hCam, IS_SET_AUTO_REFERENCE, &nominal, 0);

//Return shutter control limit:
double maxShutter;
int ret = is_SetAutoParameter (hCam, IS_GET_AUTO_SHUTTER_MAX, &maxShutter, 0);
```

 © 2015 Thorlabs Scientific Imaging

4.3.77 is_SetBinning

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetBinning (HIDS hCam, INT mode)
```

Description

Using `is_SetBinning()`, you can enable the binning mode both in horizontal and in vertical direction. This way, the image size in the binning direction can be reduced without scaling down the area of interest. Depending on the sensor used, the sensitivity or the frame rate can be increased while binning is enabled.

To enable horizontal and vertical binning at the same time, you can link the horizontal and vertical binning parameters by a logical OR.

The adjustable binning factors of each sensor are listed in the Camera and sensor data chapter.

Note

Some sensors allow a higher pixel clock setting if binning or subsampling has been activated. If you set a higher pixel clock and then reduce the binning/subsampling factors again, the driver will automatically select the highest possible pixel clock for the new settings.

Attention

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing `is_SetBinning()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()`
- `is_Exposure()`
- If you are using the DCx Camera's flash function: `is_IO()`

Note

For the models DCC1240x Binning can be used only combined for the horizontal and the vertical direction. Please see also the information in section DCC1240x / DCC3240x Application Notes.

Input parameters

`hCam`	Camera handle
⊟ `mode`	
`IS_BINNING_DISABLE`	Disables binning.
`IS_BINNING_2X_VERTICAL`	Enables vertical binning with factor 2.
`IS_BINNING_3X_VERTICAL`	Enables vertical binning with factor 3.
`IS_BINNING_4X_VERTICAL`	Enables vertical binning with factor 4.
`IS_BINNING_5X_VERTICAL`	Enables vertical binning with factor 5.
`IS_BINNING_6X_VERTICAL`	Enables vertical binning with factor 6.
`IS_BINNING_8X_VERTICAL`	Enables vertical binning with factor 8.
`IS_BINNING_16X_VERTICAL`	Enables vertical binning with factor 16.

hCam	Camera handle
IS_BINNING_2X_HORIZONTAL	Enables horizontal binning with factor 2.
IS_BINNING_3X_HORIZONTAL	Enables horizontal binning with factor 3.
IS_BINNING_4X_HORIZONTAL	Enables horizontal binning with factor 4.
IS_BINNING_5X_HORIZONTAL	Enables horizontal binning with factor 5.
IS_BINNING_6X_HORIZONTAL	Enables horizontal binning with factor 6.
IS_BINNING_8X_HORIZONTAL	Enables horizontal binning with factor 8.
IS_BINNING_16X_HORIZONTAL	Enables horizontal binning with factor 16.
IS_GET_BINNING	Returns the current setting.
IS_GET_BINNING_FACTOR_VERTICAL	Returns the vertical binning factor.
IS_GET_BINNING_FACTOR_HORIZONTAL	Returns the horizontal binning factor.
IS_GET_SUPPORTED_BINNING	Returns the supported binning modes.
IS_GET_BINNING_TYPE	Indicates whether the camera uses color-proof binning (IS_BINNING_COLOR) or not (IS_BINNING_MONO)

Return values

When used with IS_GET_BINNING	Current setting
When used with IS_GET_BINNING_FACTOR_VERTICAL IS_GET_BINNING_FACTOR_HORIZONTAL	Current setting: Returns the current factor as integer value (2, 3, 4, 5, 6, 8, 16)
When used with IS_GET_BINNING_TYPE	Returns IS_BINNING_COLOR if the camera uses color-proof binning; otherwise, IS_BINNING_MONO is returned.
When used with IS_GET_SUPPORTED_BINNING	Returns the supported binning modes linked by logical ORs.
IS_BAD_STRUCTURE_SIZE	An internal structure has an incorrect size.
IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to

 © 2015 Thorlabs Scientific Imaging

	store the image in the desired format.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_INVALID_PIXEL_CLOCK	This setting is not available for the currently set pixel clock frequency.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- `is_SetSubSampling()`
- `is_AOI()`
- `is_SetImagePos()`
- `is_PixelClock()`

4.3.78 is_SetCameraID

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetCameraID (HIDS hCam, INT nID)
```

Description

Using `is_SetCameraID()`, you can assign a unique camera ID to a camera. Thus, it is possible to access the camera directly with the `is_InitCamera()` function.

The camera ID is stored in the non-volatile memory of the camera. The factory default camera ID is 1. The camera ID can also be changed in the uc480 Camera Manager.

Input parameters

`hCam`	Camera handle
⊟ `nID`	
`1...254`	New camera ID
`IS_GET_CAMERA_ID`	Returns the current ID.

Return values

When used together with `IS_GET_CAMERA_ID`	Current ID
`IS_ACCESS_VIOLATION`	An internal error has occured. The camera ID cannot be changed because the camera is running in the boot-boost mode.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MODE`	Camera is in standby mode, function not allowed
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support

 © 2015 Thorlabs Scientific Imaging

	this function or setting.
IS_NULL_POINTER	Invalid array
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_InitCamera()
- is_GetCameraInfo()
- is_CameraStatus()

4.3.79 is_SetColorConverter

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetColorConverter (HIDS hCam, INT ColorMode, INT ConvertMode)
```

Description

Using `is_SetColorConverter()`, you can select the type of Bayer conversion for color cameras. Software conversion is done on the PC. The use of a larger filter mask results in a higher image quality, but increases the computational load. For further information, please refer to the Camera basics: Color filters chapter.

Note

Software conversion with the large filter mask should only be used for sensors whose green pixels have the same sensitivity. This applies to all DCU22xX CCD cameras.

For all other sensors, we recommend using the standard filter mask.

Attention

While free run mode is active, you cannot change the color conversion type. To do so, you must first stop the capturing process using `is_StopLiveVideo()` or set the camera to trigger mode (see `is_SetExternalTrigger()`).

Input parameters

`hCam`	Camera handle
`ColorMode`	Color mode for which the converter is to be set. For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats section.
`ConvertMode`	Conversion mode selection
`IS_CONV_MODE_NONE`	No conversion
`IS_CONV_MODE_SOFTWARE`	Only for monochrome cameras, if you want to add a gamma
`IS_CONV_MODE_SOFTWARE_3X3`	Software conversion using the standard filter mask (default)
`IS_CONV_MODE_SOFTWARE_5X5`	Software conversion using a large filter mask
`IS_CONV_MODE_HARDWARE_3X3`	(Not applicable to DCx Cameras)
`IS_CONV_MODE_OPENCL_3X3`	Software conversion using the standard filter mask, but conversion is done on the graphic board
`IS_CONV_MODE_OPENCL_5X5`	(Not applicable to DCx Cameras)

Return values

`IS_INVALID_COLOR_FORMAT`	Parameter `ColorMode` invalid or not supported
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because

© 2015 Thorlabs Scientific Imaging

	no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_INVALID_COLOR_FORMAT	Invalid color format
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_IR_FILTER	No IR filter available
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SUCCESS	Function executed successfully

Related functions

- `is_GetColorConverter()`
- `is_SetColorMode()`
- `is_Convert()`

4.3.80 is_SetColorCorrection

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetColorCorrection (HIDS hCam, INT nEnable, double* factors)
```

Description

For color cameras, `is_SetColorCorrection()` enables color correction in the uc480 driver. This enhances the rendering of colors for cameras with color sensors. Color correction is a digital correction based on a color matrix which is adjusted individually for each sensor.

Note

After changing this parameter, perform manual or automatic white balancing in order to obtain correct color rendering (see also `is_SetAutoParameter()`).

Input parameters

`hCam`	Camera handle
⊟ `nEnable`	
`IS_CCOR_ENABLE_NORMAL`	Enables simple color correction. This parameter replaces `IS_CCOR_ENABLE`.
`IS_CCOR_ENABLE_BG40_ENHANCED`	Enables color correction for cameras with optical IR filter glasses of the BG40 type.
`IS_CCOR_ENABLE_HQ_ENHANCED`	Enables color correction for cameras with optical IR filter glasses of the HQ type.
`IS_CCOR_SET_IR_AUTOMATIC`	Enables color correction for cameras with optical IR filter glasses. The glass type is set automatically as specified in the camera EEPROM.
`IS_CCOR_DISABLE`	Disables color correction.
`IS_GET_CCOR_MODE`	Returns the current setting.
`IS_GET_SUPPORTED_CCOR_MODE`	Returns all supported color correction modes. See the Return values section.
`IS_GET_DEFAULT_CCOR_MODE`	Returns the default color correction mode.
`factors`	Sets the strength of the color correction between 0.0 (no correction) and 1.0 (strong correction).

Return values

When used together with `IS_GET_CCOR_MODE`	Current setting
When used together with `IS_GET_SUPPORTED_CCOR_MODE`	When used for color cameras and together with `IS_GET_SUPPORTED_CCOR_MODE`, this parameter returns the supported values linked by a logical OR: • `IS_CCOR_ENABLE_NORMAL`

© 2015 Thorlabs Scientific Imaging

	• IS_CCOR_ENABLE_BG40_ENHANCED • IS_CCOR_ENABLE_HQ_ENHANCED When used for monochrome cameras, the system returns 0.
When used together with IS_GET_DEFAULT_CCOR_MODE	When used for color cameras and together with IS_GET_DEFAULT_CCOR_MODE, this parameter returns the default color correction mode: • IS_CCOR_ENABLE_NORMAL • IS_CCOR_ENABLE_HQ_ENHANCED When used for monochrome cameras, the system returns 0.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_IR_FILTER	No IR filter available
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetColorConverter()
- is_SetColorMode()
- is_SetAutoParameter()

4.3.81 is_SetColorMode

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetColorMode (HIDS hCam, INT Mode)
```

Description

`is_SetColorMode()` sets the color mode to be used when image data are saved or displayed by the graphics card. For this purpose, the allocated image memory must be large enough to accommodate the data with the selected color mode. When images are transferred directly to the graphics card memory, make sure that the display settings match the color mode settings. Otherwise, the images will be displayed with altered colors or are not clearly visible.

Notes

Display Modes

This function is only supported in the bitmap (DIB) display mode. Use the `is_SetDisplayMode()` function to display other color formats in Direct3D or OpenGL mode.

Bit Depth

Color formats with a bit depth of more than 8 bits per channel are not supported by DCx camera models.

RGB15/16

For the RGB16 and RGB15 data formats, the MSBs of the internal 8-bit R, G and B colors are used.

Input parameters

`hCam`	Camera handle
`Mode`: Color mode to be set. For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats section.	
`IS_CM_MONO16`	Grayscale (16), for monochrome and color cameras, LUT/gamma active
`IS_CM_MONO12`	Grayscale (12), for monochrome and color cameras, LUT/gamma active
`IS_CM_MONO8`	Grayscale (8), for monochrome and color cameras, LUT/gamma active
`IS_CM_SENSOR_RAW16`	Raw sensor data (16), for monochrome and color cameras, LUT/gamma active
`IS_CM_SENSOR_RAW12`	Raw sensor data (12), for monochrome and color cameras, LUT/gamma active
`IS_CM_SENSOR_RAW8`	Raw sensor data (8), for monochrome and color cameras, LUT/gamma active
`IS_CM_RGB12_PACKED`	RGB36 (12 12 12), for monochrome and color cameras, LUT/gamma active.
`IS_CM_RGB10_PACKED`	RGB30 (10 10 10), for monochrome and color cameras, LUT/gamma active

© 2015 Thorlabs Scientific Imaging

hCam	Camera handle
IS_CM_RGB8_PACKED	RGB24 (8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_RGBA12_PACKED	RGB48 (12 12 12), for monochrome and color cameras, LUT/gamma active
IS_CM_RGBA8_PACKED	RGB32 (8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_RGBY8_PACKED	RGBY (8 8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_BGR12_PACKED	BGR36 (12 12 12), for monochrome and color cameras, LUT/gamma active
IS_CM_BGR10_PACKED	BGR30 (10 10 10), for monochrome and color cameras, LUT/gamma active
IS_CM_BGR8_PACKED	BGR24 (8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_BGRA12_PACKED	BGR48 (12 12 12), for monochrome and color cameras, LUT/gamma active
IS_CM_BGRA8_PACKED	BGR32 (8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_BGRY8_PACKED	BGRY (8 8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_RGB8_PLANAR	Planar RGB (8) for monochrome and color cameras, LUT/gamma active
IS_CM_BGR565_PACKED	BGR16 (5 6 5), for monochrome and color cameras, LUT/gamma active
IS_CM_BGR5_PACKED	BGR15 (5 5 5), for monochrome and color cameras, LUT/gamma active
IS_CM_UYVY_PACKED	YUV 4:2:2 (8 8), for monochrome and color cameras, LUT/gamma active
IS_CM_CBYCRY_PACKED	YC_bC_r 4:2:2 (8 8), for monochrome and color cameras, LUT/gamma active
IS_GET_COLOR_MODE	Returns the current setting.

Return values

When used together with IS_GET_COLOR_MODE	Current setting
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_COLOR_FORMAT	Invalid color format

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_IR_FILTER	No IR filter available
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_SetDisplayMode()
- is_SetColorConverter()
- is_SetColorCorrection()
- is_GetColorDepth()
- is_AllocImageMem()
- is_RenderBitmap()

Note on obsolete parameters

The following parameters for color formats are obsolete. Only the new parameters should be used:

Old parameter	New parameter
IS_SET_CM_RGB32	IS_CM_BGRA8_PACKED
IS_SET_CM_RGB24	IS_CM_BGR8_PACKED
IS_SET_CM_RGB16	IS_CM_BGR565_PACKED
IS_SET_CM_RGB15	IS_CM_BGR555_PACKED
IS_SET_CM_Y8	IS_CM_MONO8
IS_SET_CM_BAYER	IS_CM_BAYER_RG8
IS_SET_CM_UYVY	IS_CM_UYVY_PACKED
IS_SET_CM_UYVY_MONO	IS_CM_UYVY_MONO_PACKED
IS_SET_CM_UYVY_BAYER	IS_CM_UYVY_BAYER_PACKED
IS_SET_CM_CBYCRY	IS_CM_CBYCRY_PACKED
IS_SET_CM_RGBY	IS_CM_BGRY8_PACKED
IS_SET_CM_RGB30	IS_CM_BGR10V2_PACKED

© 2015 Thorlabs Scientific Imaging

Old parameter	New parameter
IS_SET_CM_Y12	IS_CM_MONO12
IS_SET_CM_BAYER12	IS_CM_BAYER_RG12
IS_SET_CM_Y16	IS_CM_MONO16
IS_SET_CM_BAYER16	IS_CM_BAYER_RG16
IS_CM_BGR10V2_PACKED	IS_CM_BGR10_PACKED
IS_CM_RGB10V2_PACKED	IS_CM_RGB10_PACKED
IS_CM_BGR555_PACKED	IS_CM_BGR5_PACKED
IS_CM_BAYER_RG8	IS_CM_SENSOR_RAW8
IS_CM_BAYER_RG12	IS_CM_SENSOR_RAW12
IS_CM_BAYER_RG16	IS_CM_SENSOR_RAW16

4.3.82 is_SetDisplayMode

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetDisplayMode (HIDS hCam, INT Mode)
```

Description

Using `is_SetDisplayMode()`, you can set the way in which images will be displayed on the screen.

For live videos including overlays, you can use the Direct3D or OpenGL mode. These modes are not supported by all graphics cards. The graphics card must have sufficient extended memory because the overlay mode requires additional memory up to the size needed for the current screen resolution.

For further information on the display modes of the DCx camera, see the How to proceed: Image display section.

Attention

The Direct3D display mode is not available on Linux operating systems.

Note

We recommend that you call the following functions exclusively from a single thread in order to avoid unpredictable behaviour of the application.

- `is_InitCamera()`
- `is_SetDisplayMode()`
- `is_ExitCamera()`

See also Programming: Thread programming

Input parameters

`hCam`	Camera handle
`Mode`	
`IS_SET_DM_DIB`	Captures an image in system memory (RAM). Using `is_RenderBitmap()`, you can define the image display (default).
`IS_SET_DM_DIRECT3D`	Image display in Direct3D mode
`IS_SET_DM_DIRECT3D | IS_SET_DM_MONO`	Monochrome image display in Direct3D mode
`IS_SET_DM_DIRECT3D | IS_SET_DM_BAYER`	Raw Bayer format image display in Direct3D mode
`IS_SET_DM_OPENGL`	Image display in OpenGL mode
`IS_SET_DM_OPENGL | IS_SET_DM_MONO`	Monochrome image display in OpenGL mode
`IS_SET_DM_OPENGL | IS_SET_DM_BAYER`	Raw Bayer format image display in OpenGL mode
`IS_GET_DISPLAY_MODE`	Returns the current setting.

© 2015 Thorlabs Scientific Imaging

Attention

The new Direct3D mode completely replaces the "BackBuffer" and "Overlay Surface" display modes from DirectDraw. It is advisable not to use these modes any longer (see also Obsolete functions). To activate the obsolete modes, do the following:

IS_SET_DM_DIRECTDRAW | IS_SET_DM_BACKBUFFER	Image display in DirectDraw BackBuffer mode
IS_SET_DM_DIRECTDRAW | IS_SET_DM_ALLOW_OVERLAY	Image display in DirectDraw Overlay Surface mode
IS_SET_DM_ALLOW_SCALING	Real-time scaling in Overlay Surface mode

Return values

When used with IS_GET_DISPLAY_MODE	Current setting
IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_COLOR_FORMAT	Invalid color format
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_IR_FILTER	No IR filter available
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.

IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_RenderBitmap()
- is_SetColorMode()
- is_DirectRenderer()

© 2015 Thorlabs Scientific Imaging

Example

```
is_SetDisplayMode (hCam, Mode);

//Bitmap mode (images are digitized and stored in system memory):
Mode = IS_SET_DM_DIB

//Direct3D mode
Mode = IS_SET_DM_DIRECT3D
```

4.3.83 is_SetDisplayPos

USB 2.0 USB 3.0	-

Syntax

```
INT is_SetDisplayPos (HIDS hCam, INT x, INT y)
```

Description

`is_SetDisplayPos()` allows you to move an area of interest when rendering images using `is_RenderBitmap()`. The function moves the camera image by the selected offset within the output window. The image memory remains unchanged.

Note

To set the size and position of an area of interest in memory, use the `is_AOI()` functions.

Input parameters

hCam	Camera handle
x	Offset in x direction, measured from the top left corner of the output window
y	Offset in y direction, measured from the top left corner of the output window

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- `is_AOI()`
- `is_RenderBitmap()`
- `is_SetDisplayMode()`

© 2015 Thorlabs Scientific Imaging

4.3.84 is_SetErrorReport

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetErrorReport (HIDS hCam, INT Mode)
```

Description

Using `is_SetErrorReport()`, you can enable/disable error event logging. If error reporting is enabled, errors will automatically be displayed in a dialog box. Cancelling the dialog box disables the error report. Even with disabled error reporting, you can still query errors using the `is_GetError()` function.

Note

`is_SetErrorReport()` can be called before calling `is_InitCamera()`.
You only need to enable the `is_SetErrorReport()` function once for all cameras in the application.

Input parameters

`hCam`	Camera handle Or 0 if no camera has been initialized yet
⊟ `Mode`	
`IS_DISABLE_ERR_REP`	Disables error reporting.
`IS_ENABLE_ERR_REP`	Enables error reporting.
`IS_GET_ERR_REP_MODE`	Current status of error reporting.

Return values

When used with `IS_GET_ERR_REP_MODE`	Current setting
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetError()`
- `is_CaptureStatus()`
- `is_CameraStatus()`

4.3.85 is_SetExternalTrigger

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetExternalTrigger (HIDS hCam, INT nTriggerMode)
```

Description

Using `is_SetExternalTrigger()`, you can activate the trigger mode. If the camera is in standby mode, it quits this mode and activates trigger mode.

In hardware trigger mode, image capture is delayed for each function call until the selected trigger event has occurred.

In software trigger mode, an image is captured immediately when `is_FreezeVideo()` is called, or a continuous triggered capture is started when `is_CaptureVideo()` is called. In hardware trigger mode, you can use the `is_ForceTrigger()` command to trigger an image capture even if no electric signal is present.

When you disable the trigger functionality, you can query the signal level at the trigger input. This option causes the camera to change to freerun mode.

For further information on the image capture modes of the DCx camera, see How to proceed: Image capture.

Input parameters

`hCam`	Camera handle

⊟ `nTriggerMode`

	Trigger mode	Trigger event
`IS_SET_TRIGGER_OFF`	Off	-
`IS_SET_TRIGGER_HI_LO`	Hardware trigger	Falling signal edge
`IS_SET_TRIGGER_LO_HI`	Hardware trigger	Rising signal edge
`IS_SET_TRIGGER_PRE_HI_LO`	(Not supported by DCx Cameras)	
`IS_SET_TRIGGER_PRE_LO_HI`	(Not supported by DCx Cameras)	
`IS_SET_TRIGGER_HI_LO_SYNC`	(Not supported by DCx Cameras)	
`IS_SET_TRIGGER_LO_HI_SYNC`	(Not supported by DCx Cameras)	
`IS_SET_TRIGGER_SOFTWARE`	Software trigger	Call of `is_FreezeVideo()` (single frame mode) Call of `is_CaptureVideo()` (continuous mode)
`IS_GET_EXTERNALTRIGGER`	Returns the trigger mode setting	
`IS_GET_TRIGGER_STATUS`	Returns the current signal level at the trigger input	
`IS_GET_SUPPORTED_TRIGGER_MODE`	Returns the supported trigger modes	

© 2015 Thorlabs Scientific Imaging

Return values

When used with IS_GET_EXTERNALTRIGGER	Returns the current setting
When used with IS_GET_TRIGGER_STATUS	Returns the current signal level at the trigger input
When used with IS_GET_SUPPORTED_TRIGGER_MODE	Returns the supported modes linked by logical ORs
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_CaptureVideo()
- is_FreezeVideo()
- is_ForceTrigger()
- is_SetTriggerCounter()
- is_SetTriggerDelay()
- is_IO()

Example

```
//Enable trigger mode and set high-active flash mode.
is_SetExternalTrigger(hCam, IS_SET_TRIGGER_SOFTWARE);

// Set the flash to a high active pulse for each image in the trigger mode
UINT nMode = IO_FLASH_MODE_TRIGGER_HI_ACTIVE;
is_IO(m_hCam, IS_IO_CMD_FLASH_SET_MODE, (void*)&nMode, sizeof(nMode));

is_FreezeVideo(hCam, IS_WAIT);
```

Sample programs

- uc480 Simple Trigger (C++)
- uc480 IO (C++)

4.3.86 is_SetFrameRate

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetFrameRate (HIDS hCam, double FPS, double* newFPS)
```

Description

Using `is_SetFrameRate()`, you can set the sensor frame rate in freerun mode (live mode). Since this value depends on the sensor timing, the exposure time actually used may slightly deviate from the value set here. After you have called the function, the actual frame rate is returned through the `newFPS` parameter.

If the frame rate is set too high, it might not be possible to transfer every single frame. In this case, the effective frame rate may vary from the set value.

For minimum and maximum frame rates as well as other sensor-based dependencies, please refer to Camera and sensor data chapter.

Note

Newer driver versions sometimes allow an extended value range for the frame rate setting. We recommend to query the value range every time and set the frame rate explicitly.

Changes to the frame rate affect the value ranges of the exposure time. After executing `is_SetFrameRate()`, calling the function `is_Exposure()` is recommended in order to keep the defined camera settings.

Attention

The use of the following functions will affect the frame rate:

- `is_PixelClock()`
- `is_SetOptimalCameraTiming()`
- `is_AOI()` (if the image size is changed)
- `is_SetSubSampling()`
- `is_SetBinning()`

Changes made to the window size or the read-out timing (pixel clock frequency) also affect the defined frame rate. For this reason, you need to call `is_SetFrameRate()` again after such changes.

Attention

To be able to set the default frame rate, you have to set a pixel clock equal to or higher than the default pixel clock.

Input parameters

`hCam`	Camera handle
`FPS`	Desired frame rate in frames per second (fps)
`IS_GET_FRAMERATE`	Returns the set frame rate in the `newFPS` parameter. To query the frame rate actually reached by the camera, use `is_GetFramesPerSecond()`.
`IS_GET_DEFAULT_FRAMERATE`	Returns the default frame rate.
`newFPS`	Returns the frame rate actually set.

 © 2015 Thorlabs Scientific Imaging

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_GetFramesPerSecond()
- is_GetFrameTimeRange()
- is_PixelClock()
- is_SetOptimalCameraTiming()
- is_Exposure()
- is_SetAutoParameter()
- is_AOI()
- is_SetSubSampling()
- is_SetBinning()
- is_CaptureVideo()

4.3.87 is_SetGainBoost

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetGainBoost (HIDS hCam, INT mode)
```

Description

In some cameras, `is_SetGainBoost()` enables an additional analog hardware gain boost feature on the sensor.

Input parameters

`hCam`	Camera handle
⊟ `mode`	
`IS_GET_GAINBOOST`	Returns the current state of the gain boost function.
`IS_SET_GAINBOOST_ON`	Enables the gain boost function.
`IS_SET_GAINBOOST_OFF`	Disables the gain boost function.
`IS_GET_SUPPORTED_GAINBOOST`	Indicates whether the camera supports a gain boost feature or not.

Return values

Current setting when used together with `IS_GET_GAINBOOST`	Returns `IS_SET_GAINBOOST_ON` if the function is enabled, otherwise it returns `IS_SET_GAINBOOST_OFF`.
Current setting when used together with `IS_GET_SUPPORTED_GAINBOOST`	Returns `IS_SET_GAINBOOST_ON` if the function is supported, otherwise it returns `IS_SET_GAINBOOST_OFF`.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MODE`	Camera is in standby mode, function not allowed
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetHardwareGain()`
- `is_SetHWGainFactor()`
- `is_SetAutoParameter()`

© 2015 Thorlabs Scientific Imaging

4.3.88 is_SetGamma

USB 2.0 USB 3.0 GigE	USB 2.0 USB 3.0 GigE

Syntax

```
INT is_SetGamma (HIDS hCam, INT nGamma)
```

Description

`is_SetGamma()` enables digital gamma correction which applies a gamma characteristic to the image. When hardware color conversion is used on GigE uEye HE cameras the gamma correction is performed in the camera hardware as well. When the color conversion is performed in the PC (software conversion) the gamma correction is performed in software.

Notes

1. When the color format is set to Raw Bayer the gamma correction can not be used.
2. Typical values for gamma range between 1.6 and 2.2.

Input parameters

`hCam`	Camera handle
`nGamma`	Gamma value to be set, multiplied by 100 (Range: 1…1000. Default = 100, corresponds to a gamma value of 1.0)
`IS_GET_GAMMA`	Returns the current setting.

Return values

When used with `IS_GET_GAMMA`	Current setting
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_SUCCESS`	Function executed successfully

See also:

- Basics: Characteristics and LUT
- Basics: Color filter (Bayer filter)
- Programming: `is_SetColorConverter()`

© 2015 Thorlabs Scientific Imaging

4.3.89 is_SetHardwareGain

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetHardwareGain (HIDS hCam, INT nMaster, INT nRed, INT nGreen, INT nBlue)
```

Description

`is_SetHardwareGain()` controls the sensor gain channels. These can be set between 0 % and 100 % independently of of each other. The actual gain factor obtained for the value 100 % depends on the sensor and is specified in Camera and sensor data chapter.

You can use the `is_GetSensorInfo()` function to query the available gain controls.

Notes

1. Sensor Gain

A signal gain will also result in a noise gain. High gain settings are therefore not recommended.

We suggest the following gain settings:

1. Enable the gain boost function `is_SetGainBoost()`.
2. If required, adjust the gain setting with `is_SetHardwareGain()`

New gain settings might only become effective when the next image is captured. This depends on the time when the gain settings are changed.

2. Linearity of Sensor Gain

On DCx Cameras, you can set the gain factor in increments from 0 to 100. These increments are not graduated linearly throughout the range due to the sensor. The increments will typically be greater in the upper range than in the lower range.

The maximum gain factor settings also vary from sensor to sensor.

3. Default Settings for RGB gains

The default setting values for the red, green and blue channel gain factors depend on the color correction matrix that has been set. If you select a different color correction matrix, the returned default values might change (see also `is_SetColorCorrection()`).

Input parameters

`hCam`	Camera handle
`nMaster`	Sets the overall gain factor (0...100).
`IS_IGNORE_PARAMETER`	The master gain factor will not be changed.
`IS_GET_MASTER_GAIN`	Returns the master gain factor.
`IS_GET_RED_GAIN`	Returns the red channel gain factor.
`IS_GET_GREEN_GAIN`	Returns the green channel gain factor.
`IS_GET_BLUE_GAIN`	Returns the blue channel gain factor.
`IS_GET_DEFAULT_MASTER`	Returns the default master gain factor.
`IS_GET_DEFAULT_RED`	Returns the default red channel gain factor.
`IS_GET_DEFAULT_GREEN`	Returns the default green channel gain factor.
`IS_GET_DEFAULT_BLUE`	Returns the default blue channel gain factor.

hCam	Camera handle
IS_SET_ENABLE_AUTO_GAIN	Enables the auto gain functionality (see also is_SetAutoParameter()). You can disable the auto gain functionality by setting a value for nMaster.
nRed	Sets the red channel gain factor (0...100).
IS_IGNORE_PARAMETER	The red channel gain factor will not be changed.
nGreen	Sets the green channel gain factor (0...100).
IS_IGNORE_PARAMETER	The green channel gain factor will not be changed.
nBlue	Sets the blue channel gain factor (0...100).
IS_IGNORE_PARAMETER	The blue channel gain factor will not be changed.

Return values

When used with IS_GET_MASTER_GAIN IS_GET_RED_GAIN IS_GET_GREEN_GAIN IS_GET_BLUE_GAIN	Current setting
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_SetHWGainFactor()

© 2015 Thorlabs Scientific Imaging

- is_GetSensorInfo()
- is_SetGainBoost()
- is_SetAutoParameter()

4.3.90 is_SetHWGainFactor

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetHWGainFactor (HIDS hCam, INT nMode, INT nFactor)
```

Description

`is_SetHWGainFactor()` uses gain factors to control sensor gain channels. These channels can be set independently of each other. The `is_SetHardwareGain()` does not use factors for setting the gain channels, but standardized values between 0 and 100. The actual gain factor is sensor-dependent and can be found in Camera and sensor data chapter.

You can use the `is_GetSensorInfo()` function to query the available gain controls.

Depending on the time when the gain settings are changed, these changes might only become effective when the next image is captured.

Input parameters

`hCam`	Camera handle
⊟ `nMode`	
`IS_GET_MASTER_GAIN_FACTOR`	Returns the master gain factor.
`IS_GET_RED_GAIN_FACTOR`	Returns the red channel gain factor.
`IS_GET_GREEN_GAIN_FACTOR`	Returns the green channel gain factor.
`IS_GET_BLUE_GAIN_FACTOR`	Returns the blue channel gain factor.
`IS_SET_MASTER_GAIN_FACTOR`	Sets the master gain factor.
`IS_SET_RED_GAIN_FACTOR`	Sets the red channel gain factor.
`IS_SET_GREEN_GAIN_FACTOR`	Sets the green channel gain factor.
`IS_SET_BLUE_GAIN_FACTOR`	Sets the blue channel gain factor.
`IS_GET_DEFAULT_MASTER_GAIN_FACTOR`	Returns the default master gain factor.
`IS_GET_DEFAULT_RED_GAIN_FACTOR`	Returns the default red channel gain factor.
`IS_GET_DEFAULT_GREEN_GAIN_FACTOR`	Returns the default green channel gain factor.
`IS_GET_DEFAULT_BLUE_GAIN_FACTOR`	Returns the default blue channel gain factor.
`IS_INQUIRE_MASTER_GAIN_FACTOR`	Converts the index value for the master gain factor.
`IS_INQUIRE_RED_GAIN_FACTOR`	Converts the index value for the red channel gain factor.
`IS_INQUIRE_GREEN_GAIN_FACTOR`	Converts the index value for the green channel gain factor.
`IS_INQUIRE_BLUE_GAIN_FACTOR`	Converts the index value for the blue channel gain factor.
`nFactor`	Gain value (100 = gain factor 1, i. e. no effect)

For converting a gain value from the `is_SetHardwareGain()` function, you can set the `nMode` parameter to one of the `IS_INQUIRE_x_FACTOR` values. In this case, the value range for `nFactor` is

 © 2015 Thorlabs Scientific Imaging

between 0 and 100.

To set the gain using `IS_SET_..._GAIN_FACTOR`, you must set the `nFactor` parameter to an integer value in the range from 100 to the maximum value. By calling `IS_INQUIRE_x_FACTOR` and specifying the value 100 for `nFactor`, you can query the maximum value. A gain value of 100 means no gain, a gain value of 200 means gain to the double level (factor 2), etc.

Return values

When used with IS_GET_MASTER_GAIN_FACTOR IS_GET_RED_GAIN_FACTOR IS_GET_GREEN_GAIN_FACTOR IS_GET_BLUE_GAIN_FACTOR	Current setting
When used with IS_SET_MASTER_GAIN_FACTOR IS_SET_RED_GAIN_FACTOR IS_SET_GREEN_GAIN_FACTOR IS_SET_BLUE_GAIN_FACTOR	Defined setting
When used with IS_GET_DEFAULT_MASTER_GAIN_FACTOR IS_GET_DEFAULT_RED_GAIN_FACTOR IS_GET_DEFAULT_GREEN_GAIN_FACTOR IS_GET_DEFAULT_BLUE_GAIN_FACTOR	Default setting
When used with IS_INQUIRE_MASTER_GAIN_FACTOR IS_INQUIRE_RED_GAIN_FACTOR IS_INQUIRE_GREEN_GAIN_FACTOR IS_INQUIRE_BLUE_GAIN_FACTOR	Converted gain index
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully

Related functions

- is_SetHardwareGain()
- is_SetGainBoost()
- is_SetAutoParameter()
- is_GetSensorInfo()

Example

```
//Set master gain factor to 3.57:
INT ret = is_SetHWGainFactor (hCam, IS_SET_MASTER_GAIN_FACTOR, 357);
//ret has the value 363 for the UI-1460-C

//Query the maximum gain factor for the red channel:
ret = is_SetHWGainFactor (hCam, IS_INQUIRE_RED_GAIN_FACTOR, 100);
//ret has the value 725 for the UI-1460-C
```

4.3.91 is_SetImageMem

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetImageMem (HIDS hCam, char* pcImgMem, INT id)
```

Description

`is_SetImageMem()` makes the specified image memory the active memory. Only an active image memory can receive image data. When you call `is_FreezeVideo()`, the captured image is stored in the image buffer designated by `pcImgMem` and `id`. For `pcImgMem`, you must pass a pointer which was created by `is_AllocImageMem()`, passing any other pointer will result in an error message. You may pass the same pointer multiple times.

Notes

- In the Direct3D or OpenGL modes, there is no need to set an image memory.
- If you want the application to be compatible with the FALCON SDK, make sure to call `is_SetImageSize()` after `is_SetImageMem()`.

Input parameters

`hCam`	Camera handle
`pcImgMem`	Pointer to the starting position in the memory.
`id`	ID of this memory.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_AllocImageMem()`
- `is_FreeImageMem()`
- `is_AddToSequence()`
- `is_SetAllocatedImageMem()`
- `is_GetColorDepth()`
- `is_GetImageMem()`
- `is_GetImageMemPitch()`

© 2015 Thorlabs Scientific Imaging

4.3.92 is_SetOptimalCameraTiming

USB 2.0 USB 3.0	–

Syntax

```
INT is_SetOptimalCameraTiming (HIDS hCam, INT Mode, INT Timeout,
                               INT* pMaxPxlClk, double* pMaxFrameRate)
```

Description

Using `is_SetOptimalCameraTiming()`, you can determine the highest possible pixel clock frequency for the current configuration. This function sets the pixel clock for which no transfer errors will occur during the Timeout period. Moreover, it returns the highest frame rate available for this pixel clock frequency.

`is_SetOptimalCameraTiming()` can only be executed in free-run mode (`is_CaptureVideo()`). If the return value is ≠ `IS_SUCCESS`, no clock setting will be made.

Attention

The function should be executed in a separate thread and run in the background to allow for the computational load caused by additional color conversions, etc. Otherwise, it will not be able to return the optimum values.

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing `is_SetOptimalCameraTiming()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()`
- `is_Exposure()`
- If you are using the DCx Camera's flash function: `is_IO()`

Input parameters

hCam	Camera handle
⊟ Mode	
IS_BEST_PCLK_RUN_ONCE	The function makes one attempt to determine the optimum pixel clock and returns immediately.
Timeout [4000...20000]	Sets the period (in milliseconds) during which no transfer error may occur. The adjustable range is between 4 and 20 seconds. The higher the value you set for this parameter, the more stable the determined pixel clock value will be. This, in turn, increases the runtime of the function correspondingly.
pMaxPxlClk	Returns the maximum pixel clock frequency (in MHz).
pMaxFrameRate	Returns the maximum frame rate (in fps).

Return values

IS_AUTO_EXPOSURE_RUNNING	This setting cannot be changed while automatic exposure time control is enabled.

IS_INVALID_IMAGE_SIZE	Invalid image size This value is returned if e.g. the function is called with active binning or subsampling
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- is_PixelClock()
- is_SetFrameRate()
- is_SetAutoParameter()
- is_CaptureVideo()

© 2015 Thorlabs Scientific Imaging

4.3.93 is_SetRopEffect

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetRopEffect (HIDS hCam, INT effect, INT param, INT reserved)
```

Description

`is_SetRopEffect()` enables functions for real-time image geometry modification (Rop = raster operation).

Input parameters

`hCam`	Camera handle
⊟ `effect`	
`IS_SET_ROP_MIRROR_UPDOWN`	Mirrors the image along the horizontal axis.
`IS_SET_ROP_MIRROR_LEFTRIGHT`	Mirrors the image along the vertical axis. Depending on the sensor, this operation is performed in the camera or in the PC software.
`IS_GET_ROP_EFFECT`	Returns the current settings.
`param`	Turns the Rop effect on/off. 0 = Turn off 1 = Turn on
`reserved`	Reserved. 0 must be passed.

Return values

When used with `IS_GET_ROP_EFFECT`	Current setting
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAPTURE_MODE`	The function can not be executed in the current camera operating mode (free run, trigger or standby).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MODE`	Camera is in standby mode, function not allowed
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.

IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_SetBinning()
- is_SetSubSampling()
- is_AOI()

© 2015 Thorlabs Scientific Imaging

4.3.94 is_SetSaturation

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetSaturation (HIDS hCam, INT ChromU, INT ChromV)
```

Description

Using `is_SetSaturation()`, you can set the software color saturation.

Note

In the YUV format, color information (i.e. the color difference signals) is provided by the U and V channels. In the U channel, this information results from the difference between the blue level and Y (luminance), in the V channel from the difference between the red level and Y.

For use in other color formats than YUV, U and V are converted using a driver matrix.

Input parameters

`hCam`	Camera handle
`ChromU`	U saturation: value multiplied by 100. Range: `[IS_MIN_SATURATION … IS_MAX_SATURATION]`
`IS_GET_SATURATION_U`	Returns the current value for the U saturation.
`ChromV`	V saturation: value multiplied by 100. Range: `[IS_MIN_SATURATION … IS_MAX_SATURATION]`
`IS_GET_SATURATION_V`	Returns the current value for the V saturation.

Return values

When used with
`IS_GET_SATURATION_U`
`IS_GET_SATURATION_V`
	Current setting

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetColorMode()`
- `is_SetColorCorrection()`
- `is_SetColorConverter()`

4.3.95 is_SetSensorScaler

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetSensorScaler (HIDS hCam, UINT nMode, double dblFactor)
```

Description

`is_SetSensorScaler()` enables internal image scaling for some sensors. This allows to reduce the image resolution by adjustable factors. Thus, the amount of data from high resolution sensors can be reduced.

Note

- Internal image scaling is only supported DCC1240x and DCC3240x series cameras.
- The use of the internal scaler has no effect on the attainable frame rate.

Input parameters

`hCam`	Camera handle
⊟ `nMode:` Function mode	
`IS_ENABLE_SENSOR_SCALER`	Enable image scaling
`IS_ENABLE_SENSOR_SCALER | IS_ENABLE_ANTI_ALIASING`	Enable image scaling with smoothed edges (anti-aliasing effect)
`dblFactor`	Scaling factor

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_NULL_POINTER`	Invalid array
`IS_SUCCESS`	Function executed successfully

© 2015 Thorlabs Scientific Imaging

IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- is_GetSensorScalerInfo()

Example

```
SENSORSCALERINFO Info;
INT nRet;
double dblNewFactor;

// Query information on image scaling
nRet = is_GetSensorScalerInfo (hCam, &Info,
                               sizeof(Info));

// Enable scaling with anti aliasing
dblNewFactor = Info.dblMinFactor + Info.dblFactorIncrement;
nRet = is_SetSensorScaler (hCam, IS_ENABLE_SENSOR_SCALER |
                               IS_ENABLE_ANTI_ALIASING, dblNewFactor);
```


© 2015 Thorlabs Scientific Imaging

4.3.96 is_SetSensorTestImage

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetSensorTestImage (HIDS hCam, INT TestImage, INT Param)
```

Description

`is_SetSensorTestImage()` enables a test image function in the sensor. You can select different test images. The test images supported by a particular camera can be queried using the `is_GetSupportedTestImages()` function. For some test images, the `Param` parameter provides additional options. If the test image does not support additional parameters, `Param` will be ignored.

Attention

Manually changing the pixel clock will disable the test image mode.

Input parameters

`hCam`	Camera handle
`TestImage`	The test image to be set. See also `is_GetSupportedTestImages()`.
`Param`	Additional parameter for used to modify the test image. Not available for all test images.

Return values

`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_CAPTURE_RUNNING`	A capturing operation is in progress and must be terminated before you can start another one.
`IS_INVALID_BUFFER_SIZE`	The image memory has an inappropriate size to store the image in the desired format.
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_ACTIVE_IMG_MEM`	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a

	sequence using the `is_AddToSequence()` function.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_NULL_POINTER`	Invalid array
`IS_SUCCESS`	Function executed successfully
`IS_TIMED_OUT`	A timeout occurred. An image capturing process could not be terminated within the allowable period.
`IS_TRIGGER_ACTIVATED`	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- `is_GetSupportedTestImages()`
- `is_GetTestImageValueRange()`

© 2015 Thorlabs Scientific Imaging

4.3.97 is_SetSubSampling

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetSubSampling (HIDS hCam, INT mode)
```

Description

Using `is_SetSubSampling()`, you can enable sub-sampling mode both in horizontal and in vertical directions. This allows you to reduce the image size in the sub-sampling direction without scaling down the area of interest. In order to simultaneously enable horizontal and vertical sub-sampling, the horizontal and vertical sub-sampling parameters can by linked by a logical OR.

Some monochrome sensors are limited by their design to mere color sub-sampling. In case of fine image structures, this can result in slight artifacts.

The adjustable sub-sampling factors of each sensor are listed in Camera and sensor data chapter.

Note

Some sensors allow a higher pixel clock setting when binning or subsampling is activated. If you set a higher pixel clock and then reduce the binning/subsampling factors again, the driver will automatically select the highest possible pixel clock for the new settings.

Attention

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing `is_SetSubSampling()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()`
- `is_Exposure()`
- If you are using the DCx Camera's flash function: `is_IO()`

Input parameters

hCam	Camera handle
⊟ mode	
IS_SUBSAMPLING_DISABLE	Disables sub-sampling.
IS_SUBSAMPLING_2X_VERTICAL	Enables vertical sub-sampling with factor 2.
IS_SUBSAMPLING_3X_VERTICAL	Enables vertical sub-sampling with factor 3.
IS_SUBSAMPLING_4X_VERTICAL	Enables vertical sub-sampling with factor 4.
IS_SUBSAMPLING_5X_VERTICAL	Enables vertical sub-sampling with factor 5.
IS_SUBSAMPLING_6X_VERTICAL	Enables vertical sub-sampling with factor 6.
IS_SUBSAMPLING_8X_VERTICAL	Enables vertical sub-sampling with factor 8.
IS_SUBSAMPLING_16X_VERTICAL	Enables vertical sub-sampling with factor 16.
IS_SUBSAMPLING_2X_HORIZONTAL	Enables horizontal sub-sampling with factor 2.
IS_SUBSAMPLING_3X_HORIZONTAL	Enables horizontal sub-sampling with factor 3.
IS_SUBSAMPLING_4X_HORIZONTAL	Enables horizontal sub-sampling with factor 4.

hCam	Camera handle
IS_SUBSAMPLING_5X_HORIZONTAL	Enables horizontal sub-sampling with factor 5.
IS_SUBSAMPLING_6X_HORIZONTAL	Enables horizontal sub-sampling with factor 6.
IS_SUBSAMPLING_8X_HORIZONTAL	Enables horizontal sub-sampling with factor 8.
IS_SUBSAMPLING_16X_HORIZONTAL	Enables horizontal sub-sampling with factor 16.
IS_GET_SUBSAMPLING	Returns the current setting.
IS_GET_SUBSAMPLING_FACTOR_VERTICAL	Returns the vertical sub-sampling factor
IS_GET_SUBSAMPLING_FACTOR_HORIZONTAL	Returns the horizontal sub-sampling factor
IS_GET_SUBSAMPLING_TYPE	Indicates whether the camera uses color-proof sub-sampling.
IS_GET_SUPPORTED_SUBSAMPLING	Returns the supported sub-sampling modes.

Return values

When used with IS_GET_SUBSAMPLING	Current setting: Returns an ORing of the defined constants from ueye.h, e.g. IS_SUBSAMPLING_2X_HORIZONTAL
When used with IS_GET_SUBSAMPLING_FACTOR_VERTICAL IS_GET_SUBSAMPLING_FACTOR_HORIZONTAL	Current setting: Returns the current factor as integer value (2, 3, 4, 5, 6, 8, 16)
When used with IS_GET_SUBSAMPLING_TYPE	Current setting: Returns IS_SUBSAMPLING_COLOR if the camera uses color-proof sub-sampling, else IS_SUBSAMPLING_MONO
In Verbindung mit IS_GET_SUPPORTED_SUBSAMPLING	Returns the supported sub-sampling modes linked by logical ORs
IS_BAD_STRUCTURE_SIZE	An internal structure has an incorrect size.
IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.
IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.

 © 2015 Thorlabs Scientific Imaging

IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
IS_INVALID_MODE	Camera is in standby mode, function not allowed
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_INVALID_PIXEL_CLOCK	This setting is not available for the currently set pixel clock frequency.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the `is_SetImageMem()` function or create a sequence using the `is_AddToSequence()` function.
IS_NO_SUCCESS	General error message
IS_NOT_CALIBRATED	The camera does not contain any calibration data.
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_NULL_POINTER	Invalid array
IS_OUT_OF_MEMORY	No memory could be allocated.
IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.

Related functions

- `is_SetBinning()`

- `is_AOI()`
- `is_ImageFormat()`
- `is_PixelClock()`

© 2015 Thorlabs Scientific Imaging

4.3.98 is_SetTimeout

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetTimeout (HIDS hCam, UINT nMode, UINT Timeout)
```

Description

Using `is_SetTimeout()`, you can change user-defined timeout values of the uc480 API. If no user-defined timeout is set, the default value of the uc480 API is used for the relevant timeout.

For further information, please refer to the How to proceed: Timeout values for image capture section.

Note

The user-defined timeout only applies to the specified camera at runtime of the program.

Input parameters

`hCam`	Camera handle
`nMode`	Selects the timeout value to be set
`IS_TRIGGER_TIMEOUT`	Sets the timeout value for triggered image capture
`Timeout`	Timeout value in 10 ms. Value range [0; 4...429496729] (corresponds to 40 ms to approx. 1193 hours) 0 = use default value of the uc480 API For 1...3, the value 4 is used.

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_GetTimeout()`
- `is_CaptureVideo()`
- `is_FreezeVideo()`
- `is_SetExternalTrigger()`

Example

```
// Set user-defined timeout to 120 seconds
is_SetTimeout(hCam, IS_TRIGGER_TIMEOUT, 12000);
```


© 2015 Thorlabs Scientific Imaging

4.3.99 is_SetTriggerCounter

USB 2.0	USB 2.0

Syntax

```
INT is_SetTriggerCounter (HIDS hCam, INT nValue)
```

Description

`is_SetTriggerCounter()` returns the number of images captured in hardware or software trigger mode.

Note

In freerun mode, the counter always returns 0 even when images were captured.

Input parameters

`hCam`	Camera handle
⊟ `nValue`	
`IS_GET_TRIGGER_COUNTER`	Returns the current count for triggered image captures
Other values	Resets the counter for triggered image captures

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_SetExternalTrigger()`
- `is_CameraStatus()`

4.3.100 is_SetTriggerDelay

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_SetTriggerDelay (HIDS hCam, INT nTriggerDelay)
```

Description

Using `is_SetTriggerDelay()`, you can set the delay time between the arrival of a trigger signal and the start of exposure. The trigger signal can be initiated by hardware or by software.

The delay time set here adds to the delay caused by the sensor. The delay times of each sensor are listed in Camera and sensor data chapter.

Input parameters

`hCam`	Camera handle
`nTriggerDelay`	Time by which the image capture is delayed (in µs) 0 = deactivate trigger delay
`IS_GET_TRIGGER_DELAY`	Returns the currently set delay time.
`IS_GET_MIN_TRIGGER_DELAY`	Returns the minimum adjustable value.
`IS_GET_MAX_TRIGGER_DELAY`	Returns the maximum adjustable value.
`IS_GET_TRIGGER_DELAY_GRANULARITY`	Returns the resolution of the adjustable delay time.

Return values

When used with `IS_GET_TRIGGER_DELAY`	Current setting
`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_IO()`
- `is_SetExternalTrigger()`

© 2015 Thorlabs Scientific Imaging

4.3.101 is_StopLiveVideo

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_StopLiveVideo (HIDS hCam, INT Wait)
```

Description

`is_StopLiveVideo()` stops live mode or cancels a hardware triggered image capture in case the exposure has not yet started.

Input parameters

`hCam`	Camera handle
⊟ `Wait`	
`IS_WAIT`	The function waits until the image save is complete.
`IS_DONT_WAIT`	The function returns immediately. Digitizing the image is completed in the background.
`IS_FORCE_VIDEO_STOP`	Digitizing is stopped immediately.

Return values

`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully
`IS_TIMED_OUT`	A timeout occurred. An image capturing process could not be terminated within the allowable period.

Related functions

- `is_FreezeVideo()`
- `is_CaptureVideo()`
- `is_SetDisplayMode()`

4.3.102 is_Trigger

GigE USB 3.0	GigE USB 3.0

Syntax

```
INT is_Trigger (HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

Description

`is_Trigger()` activates the burst trigger mode in GigE and USB 3 uEye cameras. In burst trigger mode, the camera captures a series of images in rapid succession on receipt of a single trigger signal. The trigger signal can be generated by the software (`is_FreezeVideo()`) or transmitted via the digital input of the camera. The burst images are captured and transferred at maximum speed. The maximum speed depends on the pixel clock parameter (see `is_PixelClock()`) and the exposure time parameter (see `is_Exposure()`). `is_Transfer()` allows adjusting the latency of image data transfer.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `cbSizeOfParam` input parameter.

This function is currently only supported by the GigE and USB 3 uEye camera series.

Note on the uEye trigger functions

In a future version of the uEye API, all trigger functions will be combined in a single `is_Trigger()` function. The following functions will then be obsolete:

- `is_SetExternalTrigger()`
- `is_ForceTrigger()`
- `is_SetTriggerCounter()`
- `is_SetTriggerDelay()`
- `is_TriggerDebounce()`

Note on trigger delay in burst trigger mode

If you set a trigger delay with the `is_SetTriggerDelay()` function, the delay will only apply to the first image after each trigger signal.

Input parameters

hCam	Camera handle
⊟ nCommand	

© 2015 Thorlabs Scientific Imaging

IS_TRIGGER_CMD_GET_BURST_SIZE_SUPPORTED	Returns if the camera supports the burst trigger mode. More details • pParam: Pointer to variable of type UINT. • nSizeOfParam: 4 Example 1
IS_TRIGGER_CMD_GET_BURST_SIZE_RANGE	Returns the value range, the default value and the increment for the number of images in a burst. More details • pParam: Pointer to variable of type RANGE_OF_VALUES_U32 returning the values. • nSizeOfParam: sizeof (RANGE_OF_VALUES_U32)
IS_TRIGGER_CMD_GET_BURST_SIZE	Returns the currently set number of images in a burst. More details • pParam: Pointer to variable of type UINT returning the current value. • nSizeOfParam: 4 Example 2
IS_TRIGGER_CMD_SET_BURST_SIZE	Sets the number of images in a burst. More details • pParam: Pointer to variable of type UINT that passes the value to be set. • nSizeOfParam: 4

pParam	Pointer to a function parameter, whose function depends on nCommand.
cbSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Contents of the RANGE_OF_VALUES_U32 structure

UINT	u32Minimum	Minimum value
UINT	u32Maximum	Maximum value
UINT	u32Increment	Increment
UINT	u32Default	Default value

Return values

IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetExternalTrigger()
- is_ForceTrigger()
- is_SetTriggerCounter()
- is_SetTriggerDelay()
- is_TriggerDebounce()

Example 1

```
UINT nTriggerBurstSizeSupported = 0;
INT nRet = is_Trigger(m_hCam, IS_TRIGGER_CMD_GET_BURST_SIZE_SUPPORTED,
                      (void*)&nTriggerBurstSizeSupported, sizeof(nTriggerBurstSizeSupported));

if (nRet == IS_SUCCESS) {
  // Burst size supported
  if (nTriggerBurstSizeSupported == 1) {
    RANGE_OF_VALUES_U32 rangeBurstSize;
    nRet = is_Trigger(m_hCam, IS_TRIGGER_CMD_GET_BURST_SIZE_RANGE,
                      (void*)&rangeBurstSize, sizeof(rangeBurstSize));

    if (nRet == IS_SUCCESS) {
      UINT nMin, nMax;
      nMin = rangeBurstSize.u32Minimum;
      nMax = rangeBurstSize.u32Maximum;
    }
  }
}
```

Example 2

```
UINT nTriggerBurstSize = 0;
INT nRet = is_Trigger(m_hCam,
                      IS_TRIGGER_CMD_GET_BURST_SIZE,
                      (void*)&nTriggerBurstSize,
                      sizeof(nTriggerBurstSize)
                     );

nRet = is_Trigger(m_hCam,
                  IS_TRIGGER_CMD_SET_BURST_SIZE,
                  (void*)&nTriggerBurstSize,
                  sizeof(nTriggerBurstSize)
                 );
```

4.3.103 is_TriggerDebounce

GigE	GigE

Syntax

```
INT is_TriggerDebounce (HIDS hCam, UINT nCommand,
                        void *pParam, UINT nSizeOfParam)
```

Description

Using `is_TriggerDebounce()`, you can suppress disturbances at the digital input when you are running a GigE uEye camera in trigger mode. The signal at the digital input is only recognized as a trigger if the signal level remains constant at the target level for a user-selectable time. The signal edge and a delay (`DELAY_TIME`) can be set as parameters. It is recommend to use automatic signal edge selection.

Example: Mode set to "rising edge" (`TRIGGER_DEBOUNCE_MODE_RISING_EDGE`) and delay set to 50 µs. The camera will not trigger the image capture on the rising edge until the digital signal has remained at the high level for longer than 50 µs without interruption. If this is not the case, the signal

 © 2015 Thorlabs Scientific Imaging

is regarded as a disturbance and ignored.

The `nCommand` input parameter is used to select the function mode. The `pParam` input parameter depends on the selected function mode. If you select functions for setting or returning a value, `pParam` contains a pointer to a variable of the `UINT` type. The size of the memory area to which `pParam` refers is specified in the `nSizeOfParam` input parameter.

`is_TriggerDebounce()` delays the start of a triggered image capture by the selected time (`DELAY_TIME`).

This function is currently only supported by the GigE uEye camera series.

Input parameters

`hCam`	Camera handle
⊟ `nCommand`	

Setting the function mode	
`TRIGGER_DEBOUNCE_CMD_GET_SUPPORTED_MODES`	Returns the function modes supported by the camera. ⊟ More details • `pParam`: Pointer to a bit mask of type `UINT` The bit mask returns the supported modes, linked by logical ORs (see Function modes table). • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_SET_MODE`	Sets a function mode. ⊟ More details • `pParam`: Pointer to variable of type `UINT` that passes the value to be set. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_MODE`	Returns the set function mode. ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the current value. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_MODE_DEFAULT`	Returns the default mode. ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the default value. • `nSizeOfParam`: 4
Setting the delay time	
`TRIGGER_DEBOUNCE_CMD_SET_DELAY_TIME`	Sets a delay time (in µs). ⊟ More details • `pParam`: Pointer to variable of type `UINT` that passes the value to be set. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_DELAY_TIME`	Returns the set delay time (in µs). ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the current value. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_DELAY_TIME_MIN`	Returns the minimum value for the delay (in µs). ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the minimum value. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_DELAY_TIME_MAX`	Returns the maximum value for the delay (in µs). ⊟ More details • `pParam`: Pointer to variable of type `UINT` returning the maximum value. • `nSizeOfParam`: 4
`TRIGGER_DEBOUNCE_CMD_GET_DELAY_TIME_INC`	Returns the increment for setting the delay. ⊟ More details • `pParam`: Pointer to variable of type `UINT`

pParam	Pointer to a function parameter, whose function depends on nCommand.
nSizeOfParam	Size (in bytes) of the memory area to which pParam refers.

Function modes

TRIGGER_DEBOUNCE_MODE_NONE	Disables debouncing the digital input.
TRIGGER_DEBOUNCE_MODE_FALLING_EDGE	Debounces the digital input for falling edge signals.
TRIGGER_DEBOUNCE_MODE_RISING_EDGE	Debounces the digital input for rising edge signals.
TRIGGER_DEBOUNCE_MODE_BOTH_EDGES	Debounces the digital input for rising or falling edge signals.
TRIGGER_DEBOUNCE_MODE_AUTOMATIC	Debounces the digital input with automatic edge selection (recommended). The edge is selected based on the set trigger edge (see is_SetExternalTrigger()).

Return values

IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
IS_INVALID_CAMERA_HANDLE	Invalid camera handle
IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
IS_NO_SUCCESS	General error message
IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
IS_SUCCESS	Function executed successfully

Related functions

- is_SetExternalTrigger()
- is_SetTriggerDelay()

Example

```
INT value = TRIGGER_DEBOUNCE_MODE_AUTOMATIC;
is_TriggerDebounce(hCam, TRIGGER_DEBOUNCE_CMD_SET_MODE, (void*)&value, sizeof(value));
```

4.3.104 is_UnlockSeqBuf

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_UnlockSeqBuf (HIDS hCam, INT nNum, char* pcMem)
```

Description

Using `is_UnlockSeqBuf()`, you unlock a previously locked image memory in order to make it available again for storing captured images. The image memory is re-inserted at its previous position in the sequence list.

Input parameters

`hCam`	Camera handle
`nNum`	Number of the image memory to unlock. When you pass `IS_IGNORE_PARAMETER`, the image memory is only identified by its starting address. `nNum` identifies the position in the sequence list, not the memory ID assigned with `is_AllocImageMem()`.
`pcMem`	Starting address of the image memory

Return values

`IS_BAD_STRUCTURE_SIZE`	An internal structure has an incorrect size.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully

Related functions

- `is_LockSeqBuf()`
- `is_WaitForNextImage()`

© 2015 Thorlabs Scientific Imaging

4.3.105 is_WaitEvent

-	USB 2.0 USB 3.0

Syntax

```
INT is_WaitEvent (HIDS hCam, INT which, INT nTimeout)
```

Description

`is_WaitEvent()` allows waiting for uc480 events. The function indicates successful execution when the event has occurred within the specified timeout.

Input parameters

`hCam`	Camera handle
`which`	ID of the event (see `is_EnableEvent()`)
`nTimeout`	Time (in ms) that the function will wait for an event to occur. Using the constant `INFINITE` you can set the time for the timeout to infinity.

Return values

`IS_NO_SUCCESS`	General error message
`IS_SUCCESS`	Function executed successfully
`IS_TIMED_OUT`	A timeout occurred. An image capturing process could not be terminated within the allowable period. That means the specified timeout expired without the event having occurred.

Related functions

- `is_EnableEvent()`
- `is_DisableEvent()`

Example

```
//Activate and initialize FRAME event
is_EnableEvent(hCam, IS_SET_EVENT_FRAME);

//Start image capture and wait 1000 ms for event to occur
is_FreezeVideo(hCam, IS_DONT_WAIT);
INT nRet = is_WaitEvent(hCam, IS_SET_EVENT_FRAME, 1000);
if (nRet == IS_TIMED_OUT)
{
      /* wait timed out */
}
else if (nRet == IS_SUCCESS)
{
      /* event signalled */
}
is_DisableEvent(hCam, IS_SET_EVENT_FRAME);
```

4.3.106 is_WaitForNextImage

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_WaitForNextImage(HIDS hCam, UINT timeout, char** ppcMem, INT* imageID)
```

Description

`is_WaitForNextImage()` returns the pointer and sequence ID of the first (i.e. oldest) image in a memory sequence. The queue mode has to be enabled for the memory sequence (see `is_InitImageQueue()`). If the sequence does not contain images, `is_WaitForNextImage()` waits until a new image arrives or until the specified time has elapsed.

Note

Note that also image capture errors are added to the `ImageQueue` like images. If a call of `is_WaitForNextImage()` returns the `IS_CAPTURE_STATUS` return value then you can check by a new call of the function, if any further images were enqueued into the ImageQueue after the error.

Attention

Image memories in a sequence with queue mode are automatically locked. The image memories will have to be unlocked with `is_UnlockSeqBuf()` in order to be re-used in the sequence.

Input parameters

`hCam`	Camera handle
`timeout`	Timeout in ms. Range 0…2^{32}-1 If no images are in the sequence and no image arrives during the timeout, the function returns `IS_TIMED_OUT`.
`ppcMem`	Pointer to a variable which will receive the address of the last image in the sequence.
`imageID`	Pointer to a variable which will receive the sequence ID of the oldest image in the sequence.

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MEMORY_POINTER`	Invalid pointer or invalid memory ID
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.

© 2015 Thorlabs Scientific Imaging

IS_NO_SUCCESS	General error message
IS_SUCCESS	Function executed successfully
IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
IS_CAPTURE_STATUS	A transfer error occurred or no image memory was available for saving. The parameter IS_CAPTURE_STATUS replaces the previous parameter IS_TRANSFER_FAILED. The parameter IS_TRANSFER_FAILED was moved into the new header file uc480_deprecated.h, which contains all obsolete function definitions and constants. If necessary the header file uc480_deprecated.h can be included in addition to the header file uc480.h.

Related functions

- is_InitImageQueue()
- is_ExitImageQueue()
- is_UnlockSeqBuf()

See also:

- How to proceed: Image memory sequences
- How to proceed: Allocating image memory
- Transfer error: uc480 Viewer Image infos
- Transfer error: is_GetImageInfo()

4.3.107 is_WriteEEPROM

USB 2.0 USB 3.0	USB 2.0 USB 3.0

Syntax

```
INT is_WriteEEPROM (HIDS hCam, INT Adr, char* pcString, INT Count)
```

Description

Using `is_WriteEEPROM()`, you can write data to the EEPROM of the camera. Besides the hard-coded factory information, the EEPROM of the DCx Camera can hold 64 bytes of user data.

Input parameters

`hCam`	Camera handle
`Adr`	Starting address for data writes (0...63)
`pcString`	Pointer to the string containing the data to be written
`Count`	Number of characters to be written (1...64)

Return values

`IS_CANT_COMMUNICATE_WITH_DRIVER`	Communication with the driver failed because no driver has been loaded.
`IS_CANT_OPEN_DEVICE`	An attempt to initialize or select the camera failed (no camera connected or initialization error).
`IS_INVALID_CAMERA_TYPE`	The camera type defined in the .ini file does not match the current camera model.
`IS_INVALID_CAMERA_HANDLE`	Invalid camera handle
`IS_INVALID_MODE`	Camera is in standby mode, function not allowed
`IS_INVALID_PARAMETER`	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
`IS_IO_REQUEST_FAILED`	An IO request from the uc480 driver failed. Possibly the versions of the `uc480.dll` (API) and the driver file (`uc480_usb.sys`) do not match.
`IS_NO_SUCCESS`	General error message
`IS_NOT_CALIBRATED`	The camera does not contain any calibration data.
`IS_NOT_SUPPORTED`	The camera model used here does not support this function or setting.
`IS_NULL_POINTER`	Invalid array
`IS_SUCCESS`	Function executed successfully
`IS_TIMED_OUT`	A timeout occurred. An image capturing process could not be terminated within the allowable period.

 © 2015 Thorlabs Scientific Imaging

Related functions

- `is_ReadEEPROM()`

4.4 AVI Function Descriptions

The functions of the `uc480_tools.dll` enable you to save images captured with the DCx Camera as sequences to an AVI file. The How to proceed: Capturing AVIs chapter shows the command sequence for capturing an AVI video.

Notes

- **Video Resolution**

If the width or height of the video resolution is greater than 2048 pixels, some media players may not be able to play the video.

- **Video Compression**

To reduce the file size, the single frames are stored in the AVI container using an adjustable JPEG compression. It is possible to extract single frames from the AVI file.

4.4.1 isavi_AddFrame

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_AddFrame (INT nAviID, char* pcImageMem)
```

Description

`isavi_AddFrame()` adds a new frame to an AVI sequence.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function
`pcImageMem`	Pointer to the memory containing the image.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_COMPRESS`	The last image compression failed.
`IS_AVI_ERR_COMPRESSION_RUN`	The current image could not be processed since compression is still in progress.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.

Related functions

- `isavi_InitAVI()`

© 2015 Thorlabs Scientific Imaging

4.4.2 isavi_CloseAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_CloseAVI (INT nAviID)
```

Description

`isavi_CloseAVI()` closes an AVI file which was opened using `isavi_OpenAVI()`.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_OpenAVI()`
- `isavi_InitAVI()`
- `isavi_ExitAVI()`

4.4.3 isavi_DisableEvent

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_DisableEvent (INT nAviID, INT which)
```

Description

`isavi_DisableEvent()` disables the specified event. The disabled event no longer triggers an event notification when the associated event occurs.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`which`	Name of the event to be disabled.
`IS_AVI_SET_EVENT_FRAME_SAVED`	A new frame was saved to the AVI file.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found.

	Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_PARAMETER`	One of the submitted parameters is outside the valid range.

Related functions

- `isavi_EnableEvent()`

4.4.4 isavi_EnableEvent

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_EnableEvent (INT nAviID, INT which)
```

Description

`isavi_EnableEvent()` enables the specified event. The enabled event triggers an event notification when the associated event occurs.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`which`	Name of the event to be enabled.
`IS_AVI_SET_EVENT_FRAME_SAVED`	A new frame was saved to the AVI file.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_PARAMETER`	One of the submitted parameters is outside the valid range.

Related functions

- `isavi_DisableEvent()`

© 2015 Thorlabs Scientific Imaging

4.4.5 isavi_ExitAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_ExitAVI (INT nAviID)
```

Description

`isavi_ExitAVI()` terminates and deletes the instance of the uc480 AVI interface.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.

Related functions

- `isavi_InitAVI()`
- `isavi_OpenAVI()`
- `isavi_CloseAVI()`

4.4.6 isavi_ExitEvent

USB 2.0 USb 3.0	-

Syntax

```
INT isavi_ExitEvent (INT nAviID, INT which)
```

Description

`isavi_ExitEvent()` deletes the specified event. The deleted event no longer occurs and needs to be re-created using `isavi_InitEvent()` before it can be enabled and used.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`which`	Name of the event to be deleted.
`IS_AVI_SET_EVENT_FRAME_SAVED`	A new frame was saved to the AVI file.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_PARAMETER`	One of the submitted parameters is outside the valid range.

Related functions

- `isavi_InitEvent()`
- `isavi_EnableEvent()`
- `isavi_DisableEvent()`

© 2015 Thorlabs Scientific Imaging

4.4.7 isavi_GetAVIFileName

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_GetAVIFileName (INT nAviID, char* strName)
```

Description

Using `isavi_GetAVIFileName()`, you can read out the filename of the current AVI file. This function is helpful if an AVI file was opened with the `isavi_OpenAVI()` function and a Null parameter was specified.

Note

The functions `isavi_OpenAVI()` and `isavi_GetAVIFileName()` do not support UNICODE strings. For this purpose you use the functions `isavi_OpenAVIW()` and `isavi_GetAVIFileNameW()`.

If the AVI file was created using a UNICODE string, only the `isavi_GetAVIFileNameW()` function can return the right file string.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`strName`	Pointer to the memory location where the filename is written to. The allocated memory must be large enough to accommodate the full file path.
`NULL`	When `NULL` is passed the function returns the length of the filename.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetAVISize()`
- `isavi_OpenAVI()`

4.4.8 isavi_GetAVIFileNameW

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_GetAVIFileNameW (INT nAviID, wchar_t* strName)
```

Description

Using `isavi_GetAVIFileNameW()`, you can read out the filename of the current AVI file as UNICODE string. This function is helpful if an AVI file was opened with the `isavi_OpenAVIW()` function and a `NULL` parameter was specified.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`strName`	Pointer to the memory location where the filename is written to. The allocated memory must be large enough to accommodate the full file path as UNICODE string.
`NULL`	When `NULL` is passed the function returns the length of the filename.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetAVISize()`
- `isavi_OpenAVIW()`

© 2015 Thorlabs Scientific Imaging

4.4.9 isavi_GetAVISize

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_GetAVISize (INT nAviID, float* size)
```

Description

Use `isavi_GetAVISize()` to retrieve the size of the frame sequence saved to the current AVI file.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function
`size`	The size in kBytes

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetAVIFileName()`

4.4.10 isavi_GetnCompressedFrames

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_GetnCompressedFrames (INT nAviID, unsigned long* nFrames)
```

Description

Using `isavi_GetnCompressedFrames()`, you can read out the number of frames saved to the current AVI file.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`nFrames`	The number of frames

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetnLostFrames()`
- `isavi_ResetFrameCounters()`

© 2015 Thorlabs Scientific Imaging

4.4.11 isavi_GetnLostFrames

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_GetnLostFrames (INT nAviID, unsigned long* nFrames)
```

Description

Using `isavi_GetnLostFrames()`, you can read out the number of frames that have been discarded. A frame will be discarded if it cannot be processed because a compression operation in still in progress.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`nFrames`	The number of frames

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetnCompressedFrames()`
- `isavi_ResetFrameCounters()`

4.4.12 isavi_InitAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_InitAVI (INT* pnAviID, HIDS hCam)
```

Description

`isavi_InitAVI()` initializes an instance of the uc480 AVI interface. Multiple instances can be created simultaneously.

Input parameters

`pnAviID`	Pointer. Returns the instance ID which is needed for calling the other uc480 AVI functions.
`hCam`	Handle of a selected or initialized DCx camera.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_PARAMETER`	One of the submitted parameters is outside the valid range.
`IS_AVI_ERR_NO_CODEC_AVAIL`	The maximum number of instances allowed in this system has been reached. It is not possible to create another instance.
`IS_AVI_ERR_INVALID_UEYE`	No DCx camera was found.

Related functions

- `isavi_ExitAVI()`
- `isavi_OpenAVI()`
- `isavi_CloseAVI()`

© 2015 Thorlabs Scientific Imaging

4.4.13 isavi_InitEvent

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_InitEvent (INT nAviID, INT which)
```

Description

`isavi_InitEvent()` creates the specified event. This includes registering the event object in the uc480 AVI interface and creating an event handler. Before you can use a new event, you must enable it by calling `isavi_EnableEvent()`.

Input parameters

nAviID	Instance ID set by the `isavi_InitAVI()` function.
which	Name of the event to be created.
IS_AVI_SET_EVENT_FRAME_SAVED	A new frame was saved to the AVI file.

Return values

IS_AVI_NO_ERR	Function executed successfully.
IS_AVI_ERR_INVALID_ID	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
IS_AVI_ERR_EVENT_FAILED	The event could not be initialized. The Windows `SetEvent` function failed.
IS_AVI_ERR_PARAMETER	One of the submitted parameters is outside the valid range.

Related functions

- `isavi_ExitEvent()`
- `isavi_EnableEvent()`
- `isavi_DisableEvent()`

Example

Create and enable an event object for the "Frame saved" event:

```
HANDLE hEvent = CreateEvent( NULL, TRUE, FALSE, "" );
if ( hEvent != NULL )
{
  isavi_InitEvent( AviDest, hEvent, IS_AVI_SET_EVENT_FRAME_SAVED );
  isavi_EnableEvent( AviDest, IS_AVI_SET_EVENT_FRAME_SAVED );

    if ( WaitForSingleObject( hEvent, 1000 ) == WAIT_OBJECT_0 )
    {
      //Frame was captured successfully...
    }
  isavi_DisableEvent( AviDest, IS_AVI_SET_EVENT_FRAME_SAVED );
  isavi_ExitEvent( AviDest, IS_AVI_SET_EVENT_FRAME_SAVED );
}
```

4.4.14 isavi_OpenAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_OpenAVI (INT nAviID, const char* strFileName)
```

Description

`isavi_OpenAVI()` opens a new or existing AVI file.

Note

The functions `isavi_OpenAVI()` and `isavi_GetAVIFileName()` do not support UNICODE strings. For this purpose you use the functions `isavi_OpenAVIW()` and `isavi_GetAVIFileNameW()`.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`pFileName`	Pointer to the name to be assigned to the AVI file. If `NULL` is passed, the "Open File" dialog is displayed.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_CAPTURE_RUNNING`	Another capturing operation is in progress or an AVI file is still open.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.
`IS_AVI_ERR_NEW_FAILED`	No memory could be allocated for the AVI file.
`IS_AVI_ERR_CREATESTREAM`	No AVI stream could be created.

Related functions

- `isavi_GetAVIFileName()`
- `isavi_CloseAVI()`
- `isavi_InitAVI()`
- `isavi_ExitAVI()`

© 2015 Thorlabs Scientific Imaging

4.4.15 isavi_OpenAVIW

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_OpenAVIW (INT nAviID, const wchar_t* strFileName)
```

Description

`isavi_OpenAVIW()` opens a new or existing AVI file.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`pFileName`	Pointer to the name to be assigned to the AVI file. The file name is passed as UNICODE string. If `NULL` is passed, the "Open File" dialog is displayed.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_CAPTURE_RUNNING`	Another capturing operation is in progress or an AVI file is still open.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.
`IS_AVI_ERR_NEW_FAILED`	No memory could be allocated for the AVI file.
`IS_AVI_ERR_CREATESTREAM`	No AVI stream could be created.

Related functions

- `isavi_GetAVIFileNameW()`
- `isavi_CloseAVI()`
- `isavi_InitAVI()`
- `isavi_ExitAVI()`

4.4.16 isavi_ResetFrameCounters

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_ResetFrameCounters (INT nAviID)
```

Description

`isavi_ResetFrameCounters()` resets the counters for saved and discarded images.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.

Related functions

- `isavi_GetnCompressedFrames()`
- `isavi_GetnLostFrames()`

© 2015 Thorlabs Scientific Imaging

4.4.17 isavi_SetFrameRate

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_SetFrameRate (INT nAviID, double fr)
```

Description

`isavi_SetFrameRate()` sets the frame rate for AVI capturing. You can set the frame rate after opening the AVI file. This value does not have to be equal to the frame rate set for the DCx camera.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`fr`	The frame rate to be set. Default = 25.0

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_WRITE_INFO`	The AVI file could not be modified.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.

Related functions

- `isavi_SetImageQuality()`
- `isavi_SetImageSize()`

4.4.18 isavi_SetImageQuality

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_SetImageQuality (INT nAviID, INT q)
```

Description

`isavi_SetImageQuality()` indicates the quality for the frames to be compressed. You can change the image quality at any time; it then applies to all subsequent frames. For compression, the system uses the JPEG algorithm.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`q`	Image quality [1 = lowest ... 100 = highest]

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_INVALID_VALUE`	The `q` parameter is outside the range of 1…100.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.

Related functions

- `isavi_SetFrameRate()`
- `isavi_SetImageSize()`

© 2015 Thorlabs Scientific Imaging

4.4.19 isavi_SetImageSize

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_SetImageSize (INT nAviID, INT cMode, INT Width, INT Height,
                        INT PosX, INT PosY, INT LineOffset)
```

Description

`isavi_SetImageSize()` sets the size and position of the area of interest which will be saved to the AVI file. Only the defined area of interest of each frame will be saved. In addition, this function specifies the input color format of the frames. You define these settings only once for the entire video.

Note

The supported input color formats are RGB32, RGB24, Y8 and raw Bayer. The output file will always be in RGB24 format, regardless of the input data format. For further information on the structure of the different color formats, see the Appendix: Color and memory formats section.

Attention

When an area of interest is used, the width (`Width`) and height (`Height`) of the AOI must be at least 16 pixel. The AOI width must be a multiple of 8.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.
`cMode`	Color format of the input frames captured by the DCx Camera.
`Width`	Width of the entire frame or of the area of interest.
`Height`	Height of the entire frame or of the area of interest.
`PosX`	X position (offset) of the area of interest.
`PosY`	Y position (offset) of the area of interest.
`LineOffset`	Line increment. The line increment is the difference between the width of the entire frame (in pixel) and the area of interest (in pixel).

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.
`IS_AVI_ERR_CAPTURE_RUNNING`	Another capturing operation is in progress or an AVI file is still open.
`IS_AVI_ERR_ALLOC_MEMORY`	No memory could be allocated.
`IS_AVI_ERR_INVALID_CM`	The submitted color mode is not supported for AVI capturing.

IS_AVI_ERR_INVALID_SIZE	The submitted size is invalid.
IS_AVI_ERR_INVALID_POSITION	The submitted position is invalid.

Related functions

- isavi_SetFrameRate()
- isavi_SetImageQuality()

© 2015 Thorlabs Scientific Imaging

Example

```
// Query image buffer geometry
int nWidth, nWidth, nBits, nPitch;
is_InquireImageMem ( hCam, pLast, nImageID,
            &nWidth, &nHeight,
            &nBits, &nPitch);
INT nOffsetX = is_SetImagePos ( hCam, IS_GET_IMAGE_POS_X_ABS, 0);
INT nOffsetY = is_SetImagePos ( hCam, IS_GET_IMAGE_POS_Y_ABS, 0);

// Derive pixel pitch from buffer byte pitch
INT nPitchPx=0;
nPitchPx = (nPitch * 8 ) / nBits;

INT nAviWidth = nWidth /8 * 8; // Width must be multiple of 8
INT LineOffsetPx = nPitchPx - nAviWidth ;
isavi_SetImageSize( nAviId, m_cMode,
            nAviWidth, nHeight,
            nOffsetX, nOffsetY,
            LineOffsetPx);
```

4.4.20 isavi_StartAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_StartAVI (INT nAviID)
```

Description

`isavi_StartAVI()` starts the image capture thread.

Input parameters

nAviID	Instance ID set by the `isavi_InitAVI()` function.

Return values

IS_AVI_NO_ERR	Function executed successfully.
IS_AVI_ERR_INVALID_ID	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
IS_AVI_ERR_INVALID_FILE	The file has no valid AVI format.
IS_AVI_ERR_PLAY_RUNNING	A playback is already running.

Related functions

- `isavi_StopAVI()`
- `isavi_InitEvent()`
- `isavi_ExitAVI()`

4.4.21 isavi_StopAVI

USB 2.0 USB 3.0	-

Syntax

```
INT isavi_StopAVI (INT nAviID)
```

Description

`isavi_StopAVI()` stops the image capture thread. Subsequent calls of `isavi_AddFrame()` will be ignored.

Input parameters

`nAviID`	Instance ID set by the `isavi_InitAVI()` function.

Return values

`IS_AVI_NO_ERR`	Function executed successfully.
`IS_AVI_ERR_CAPTURE_NOT_RUNNING`	No capturing operation is running or no AVI file is opened.
`IS_AVI_ERR_INVALID_ID`	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using `isavi_ExitAVI()`.
`IS_AVI_ERR_INVALID_FILE`	The file has no valid AVI format.
`IS_AVI_ERR_PLAY_NOT_RUNNING`	No playback is running.

Related functions

- `isavi_StartAVI()`
- `isavi_InitEvent()`
- `isavi_ExitAVI()`

© 2015 Thorlabs Scientific Imaging

4.5 RAW function descriptions

The functions of the `uEye_tools.dll` enable you to save images captured with the uEye camera as an image sequences to an RAW file or open it.

4.5.1 israw_AddFrame

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_AddFrame(UINT unFileID, const char* pcData, UINT64 unTimestampDevice)
```

Description

`israw_AddFrame()` adds a new frame to an image sequence.

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`pcData`	Pointer to the image data
`unTimestampDevice`	Data timestamp

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_FILE_NOT_OPEN`	The file is not open.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_INVALID_ID`	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using `israw_ExitFile()`.
`IS_AVI_ERR_NOT_SUPPORTED`	The file was opened in "read-only" mode (see `israw_InitFile()`).

Related functions

- `israw_GetFrame()`

4.5.2 israw_CloseFile

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_CloseFile(UINT unFileID)
```

Description

`israw_CloseFile()` closes a RAW file which was opened using `israw_OpenFile()`.

Input parameters

unFileID	Instance ID set by the israw_InitFile() function

Return values

IS_AVI_NO_ERR	No error, function executed successfully.
IS_AVI_ERR_EXCEPTION	An error has occurred.
IS_AVI_ERR_GENERIC	Generic error
IS_AVI_ERR_INVALID_ID	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using israw_ExitFile().

Related functions

- israw_OpenFile()

4.5.3 israw_ExitFile

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_ExitFile(UINT unFileID)
```

Description

israw_ExitFile() terminates and deletes the instance of the uEye RAW interface.

Input parameters

unFileID	Instance ID set by the israw_InitFile() function

Return values

IS_AVI_NO_ERR	No error, function executed successfully.
IS_AVI_ERR_EXCEPTION	An error has occurred.
IS_AVI_ERR_GENERIC	Generic error

Related functions

- israw_InitFile()

4.5.4 israw_GetFrame

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_GetFrame(UINT unFileID, char* pData, UINT64* punTimestampDevice)
```


© 2015 Thorlabs Scientific Imaging

Description

`israw_GetFrame()` returns the image data from an image sequence.

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`pData`	Pointer to the image data
`punTimestampDevice`	Data timestamp

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_FILE_NOT_OPEN`	The file is not open.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_INVALID_ID`	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using `israw_ExitFile()`.
`IS_AVI_ERR_NOT_SUPPORTED`	The file was opened in "read-only" mode (see `israw_InitFile()`).
`IS_AVI_ERR_READ_FAILED`	The file could not be read.

Related functions

- `israw_SeekFrame()`

4.5.5 israw_GetImageInfo

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_GetImageInfo(UINT unFileID, UINT* punWidth, UINT* punHeight, UINT* punBitsPerPixel)
```

Description

`israw_GetImageInfo()` returns the width, height, and bit depth of the data which are saved in the RAW file.

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`punWidth`	Image width
`punHeight`	Image height
`punBitsPerPixel`	Bit depth (bits per pixel)

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.

IS_AVI_ERR_FILE_NOT_OPEN	The file is not open.
IS_AVI_ERR_GENERIC	Generic error
IS_AVI_ERR_INVALID_ID	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using israw_ExitFile().
IS_AVI_ERR_PARAMETER	One of the submitted parameters is outside the valid range.

Related functions

- israw_SetImageInfo()

4.5.6 israw_GetSize

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_GetSize(UINT unFileID, float* pfSize)
```

Description

israw_GetSize() returns the size of the saved image sequence in the current RAW file.

Input parameters

unFileID	Instance ID set by the israw_InitFile() function
pfSize	Size in kBytes

Return values

IS_AVI_NO_ERR	No error, function executed successfully.
IS_AVI_ERR_EXCEPTION	An error has occurred.
IS_AVI_ERR_FILE_NOT_OPEN	The file is not open.
IS_AVI_ERR_GENERIC	Generic error
IS_AVI_ERR_INVALID_ID	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using israw_ExitFile().

Related functions

- israw_OpenFile()

4.5.7 israw_InitFile

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_InitFile(UINT* punFileID, INT nAccessMode)
```

 © 2015 Thorlabs Scientific Imaging

Description

`israw_InitFile()` initializes an instance of the uEye RAW interface. Multiple instances can be created simultaneously.

Input parameters

`punFileID`	Pointer in which the instance ID is returned. This ID is needed for calling other functions.
`nAccessMode`	File access mode: • Read: `IS_FILE_ACCESS_MODE_READ` • Write: `IS_FILE_ACCESS_MODE_WRITE`

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_PARAMETER`	One of the submitted parameters is outside the valid range.
`IS_AVI_ERR_NEW_FAILED`	The maximum number of instances have been reached.

Related functions

- `israw_ExitFile()`

4.5.8 israw_OpenFile

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_OpenFile(UINT unFileID, const char* strFileName)
```

Description

`israw_OpenFile()` opens a new or existing RAW file.

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`strFileName`	RAW file name

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_FILE_NOT_OPEN`	The file is not open.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_INVALID_ID`	The indicated instance could not be found. Either the ID is invalid or the instance has

	already been deleted using `israw_ExitFile()`.
`IS_AVI_ERR_INVALID_VALUE`	No valid file name was selected.

Related functions

- `israw_CloseFile()`

4.5.9 israw_SeekFrame

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_SeekFrame(UINT unFileID, UINT unFrame)
```

Decsription

`israw_SeekFrame()` jumps to a specific image within an image sequence.

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`unFrame`	Image to be jumped to (index starts with "0")

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_FILE_NOT_OPEN`	The file is not open.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_INVALID_ID`	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using `israw_ExitFile()`.
`IS_AVI_ERR_NOT_SUPPORTED`	The file was opened in "read-only" mode (see `israw_InitFile()`).
`IS_AVI_ERR_SEEK_FAILED`	The image could not be found.

Related functions

- `israw_GetFrame()`

4.5.10 israw_SetImageInfo

USB 2.0 USB 3.0 GigE	-

Syntax

```
INT israw_SetImageInfo(UINT unFileID, UINT unWidth, UINT unHeight, UINT unBitsPerPixel)
```


© 2015 Thorlabs Scientific Imaging

Description

`israw_SetImageInfo()` sets the width, height, and bit depth of the data that is saved in the RAW file (see `is_raw_AddFrame()`).

Input parameters

`unFileID`	Instance ID set by the `israw_InitFile()` function
`Width`	Image width
`Height`	Image height
`BitsPerPixel`	Bit depth (bits per pixel)

Return values

`IS_AVI_NO_ERR`	No error, function executed successfully.
`IS_AVI_ERR_EXCEPTION`	An error has occurred.
`IS_AVI_ERR_GENERIC`	Generic error
`IS_AVI_ERR_INVALID_ID`	The indicated instance could not be found. Either the ID is invalid or the instance has already been deleted using `israw_ExitFile()`.
`IS_AVI_ERR_INVALID_VALUE`	No valid file name was selected.
`IS_AVI_ERR_NOT_SUPPORTED`	The file was opened in "read-only" mode (see `israw_InitFile()`).

Related functions

- `israw_AddFrame()`
- `israw_GetImageInfo()`

4.6 Obsolete functions

We are continuously extending and enhancing the uc480 API. The resulting product upgrades sometimes require replacing obsolete functions with new ones. We recommend against using the obsolete functions. They will continue to be supported for reasons of backward compatibility, but they will not be documented any longer.

Note on older functions

If it is necessary to continue working with the older functions, it is possible to add the `uc480_deprecated.h` header file additionally to the `uc480.h` header file. The `uc480_deprecated.h` header file contains all obsolete function definitions and constants which are no longer part of the `uc480.h` header file.

The following table lists the obsolete functions and indicates the recommended alternatives. See also History of API functions.

Obsolete function	Recommended alternative	No longer documented since version	New function valid from version
`is_ConvertImage()`	`is_Convert()`	4.50	4.00
`is_DisableDDOverlay()`	`is_DirectRenderer()`	4.50	3.40
`is_EnableDDOverlay()`		4.50	
`is_GetCameraLUT`	`is_LUT()`	4.50	4.40
`is_GetCameraType()`	`is_GetCameraInfo()`	4.50	2.00

is_GetCaptureErrorInfo()	is_CaptureStatus()	4.50	3.90
is_GetDC()	is_DirectRenderer()	4.50	3.40
is_GetDDOvlSurface()		4.50	
is_GetEthDeviceInfo	is_DeviceInfo()	4.50	4.00
is_GetExposureRange	is_Exposure()	4.50	3.90
is_GetGlobalFlashDelays()	is_IO()	4.50	3.90
is_GetLastMemorySequence()	The uEye memory board is not supported any longer (see below).	3.30	
is_GetMemorySequenceWindow()			
is_GetNumberOfMemoryImages()			
is_GetOsVersion	-	4.50	
is_GetPixelClockRange()	is_PixelClock()	4.50	4.00
is_GetRevisionInfo()	is_GetCameraInfo()	3.20	2.00
is_GetHWGain()	is_SetAutoParameter()	3.31	2.20
is_HideDDOverlay()	is_DirectRenderer()	4.50	3.40
is_IsMemoryBoardConnected()	The uEye memory board is not supported any longer (see below).	3.30	
is_LoadBadPixelCorrectionTable()	is_HotPixel()	4.50	3.80
is_LoadImage()	is_ImageFile()	4.50	4.00
is_LoadImageMem()		4.50	
is_LoadParameters()	is_ParameterSet()	4.50	4.00
is_LockDDMem()	is_DirectRenderer()	4.50	3.40
is_LockDDOverlayMem()		4.50	
is_MemoryFreezeVideo()	The uEye memory board is not supported any longer (see below).	3.30	
is_ReleaseDC()	is_DirectRenderer()	4.50	3.40
is_ResetCaptureErrorInfo()	is_CaptureStatus()	4.50	3.90
is_ResetMemory()	The uEye memory board is not supported any longer (see below).	3.30	
is_SaveBadPixelCorrectionTable()	is_HotPixel()	4.50	3.80
is_SaveImage()	is_ImageFile()	4.50	4.00
is_SaveImageEx()		4.50	
is_SaveImageMem()		4.50	
is_SaveImageMemEx()		4.50	
is_SaveParameters()	is_ParameterSet()	4.50	4.00
is_SetAOI()	is_AOI()	4.50	3.80
is_SetAutoCfgIpSetup()	is_IpConfig()	4.50	3.82
is_SetBadPixelCorrection()	is_HotPixel()	4.50	3.80
is_SetBadPixelCorrectionTable		4.50	

© 2015 Thorlabs Scientific Imaging

()			
is_SetBayerConversion()	is_SetColorConverter()	4.50	3.30
is_SetBlCompensation()	is_Blacklevel()	4.50	4.01
is_SetBrightness()	is_Exposure() is_SetHardwareGain()	3.40	3.90 2.00
is_SetCameraLUT	is_LUT()	4.50	4.40
is_SetContrast()	is_Gamma() is_SetHardwareGamma() is_Blacklevel()	3.40	4.40 2.00 4.01
is_SetConvertParam()	is_Convert()	4.50	4.00
is_SetDDUpdateTime()	is_DirectRenderer()	4.50	3.40
is_SetEdgeEnhancement()	is_EdgeEnhancement()	4.50	4.00
is_SetExposureTime()	is_Exposure()	4.50	3.90
is_SetFlashDelay()	is_IO()	4.50	3.90
is_SetFlashStrobe()		4.50	
is_SetGamma()	is_Gamma()	4.50	4.40
is_SetHwnd()	is_DirectRenderer()	4.50	3.40
is_SetImageAOI()	is_AOI()	4.50	3.80
is_SetImageSize()		4.50	
is_SetImagePos()		4.50	
is_SetIO()	is_IO()	4.50	3.90
is_SetIOMask()		4.50	
is_SetKeyColor()	is_DirectRenderer()	4.50	3.40
is_SetLED()	is_IO()	4.50	3.90
is_SetMemoryMode()	The uEye memory board is not supported any longer (see below).	3.30	
is_SetPixelClock()	is_PixelClock()		4.00
is_SetWhiteBalance()	is_SetAutoParameter()	3.31	2.20
is_SetWhiteBalanceMultipliers ()			
is_SetPersistentIpCfg()	is_IpConfig()	4.50	3.82
is_ShowDDOverlay()	is_DirectRenderer()	4.50	3.40
is_StealVideo()		4.50	
is_TransferImage()	The uEye memory board is not supported any longer (see below).	3.30	
is_TransferMemorySequence()			
is_UnlockDDMem()	is_DirectRenderer()	4.50	3.40
is_UnlockDDOverlayMem()		4.50	
is_UpdateDisplay()		4.50	

The `is_SetWhiteBalance()` and `is_SetWhiteBalanceMultipliers()` functions have been completely replaced by the `is_SetAutoParameter()` function and are no longer supported by the uc480 API.

4.7 Programming Notes

Note

Parameter Validity

Functions that refer to an initialized camera have the camera handle `HIDS hCam` as the first parameter. All parameters that are set using these functions remain valid for as long as the handle is valid, that is, until you close the corresponding camera or exit the program. The next time you open the camera, it is initialized with the defaults again.

Attention

All input parameters of a function have to be initialized with valid values before the function is called; this also applies to parameters that are not used. Variables can be preset with '0', for example. For unused parameters, the NULL pointer has to be passed.

The uc480.h header file

The `uc480.h` header file contains all the definitions and constants needed for the uc480 API. After the installation of the uc480 drivers you will find this file in the directory:

- Windows: `C:\Program Files\Thorlabs\DCx Cameras\Develop\include`
- Linux: `/usr/include`

Note

Note on older functions

If it is necessary to continue working with the older functions, it is possible to add the `uc480_deprecated.h` header file additionally to the `uc480.h` header file. The `uc480_deprecated.h` header file contains all obsolete function definitions and constants which are no longer part of the `uc480.h` header file.

See also:

- Programming in C / C++
- Programming in C#
- Programming in VB.NET
- Programming in Delphi
- Programming with ActiveX
- Thread programming

4.7.1 Programming in C/C++

For programming with the uc480 API, we suggest to use the C/C++ programming language. This programming language offers efficient access to all functions of the uc480 API. Enabling access to image memory contents through pointers, C/C++ is especially suitable for image processing applications.

Most of the uc480 sample programs were created in Microsoft Visual Studio using the C++ programming language.

Hint

We suggest that you keep the function libraries (DLL, AX and OCX files) in the default directory. After the installation, these files reside e.g. under Window (32 bit) in `C:\Windows\System32\`.

 © 2015 Thorlabs Scientific Imaging

Copying these files to other locations may result in version conflicts.

Required Files

In order to access the uc480 API, make sure to include the following files in your project:

- Header file: `uc480.h`
- Lib file: uc480.lib
- Function library (DLL): `uc480.dll`

In order to access the DCx Camera AVI functions, make sure to include the following files in your project:

- Header file: uc480_tools.h
- Lib file: uc480_tools.lib
- Function library (DLL): `uc480_tools.dll`

In order to access the uc480 DirectShow functions, make sure to include the following files in your project:

- Header file: uc480CaptureInterface.h
- DirectShow interface: uc480capture.ax

Note

Programming under Linux

In order to access the uc480 API, make sure to include the following files in your project:

- Header file: uc480.h
- Library: `libuc480.so`

4.7.2 Programming in C#

We suggest to use the C# programming language for the creation of visualization applications. While it is possible to access image memory contents, doing so is more tedious than in C/C++ due to the 'managed code'. To access image memory contents in C#, you can use 'unsafe code' or the 'Marshall class'. Some system-level functions, such as Windows event handling, can be integrated using the Windows API.

The uc480 SDK includes sample programs for programming with Microsoft Visual Studio in the C# programming language.

Required Files

In order to access the uc480 API in C#, make sure to include the following files in your project:

- Header file: `uc480.cs`
- Function library (DLL): `uc480.dll`

In order to access the uc480 AVI functions in C#, make sure to include the following files in your project:

- Header file: `uc480_tools.cs`
- Function library (DLL): `uc480_tools.dll`

Hint

We suggest that you keep the function libraries (DLL, AX and OCX files) in the default directory. After the installation, these files reside e.g. under Window (32 bit) in `C:\Windows\System32\`.

Copying these files to other locations may result in version conflicts.

4.7.3 Programming in VB.NET

We suggest to use the Visual Basic programming language for the creation of applications which are exclusively used for visualization purposes. The access to image memory contents is extremely tedious due to the missing pointer arithmetics.

We suggest to use the uc480 ActiveX component when programming in VB.Net. The uc480 SDK includes a sample program for programming with Microsoft Visual Studio in the VB.NET programming language using the uc480 ActiveX component.

The constants can be looked up in the `uc480.h` file.

4.7.4 Programming in Delphi

The uc480 SDK does not provide direct integration of the uc480 API for the Delphi programming language. In order to use the uc480 API in Delphi, you need to create separate header files. We suggest to use the uc480 ActiveX component (see also Programming with ActiveX) when programming in Delphi.

Attention

To use the uc480_`api.dll` in Delphi, the `cdecl` calling convention has to be used.

Hint

We suggest that you keep the function libraries (DLL, AX and OCX files) in the default directory. After the installation, these files reside e.g. under Window (32 bit) in `C:\Windows\System32\`.

Copying these files to other locations may result in version conflicts.

4.7.5 Programming with ActiveX

The uc480 SDK comes with an ActiveX component that allows you to use almost all functions of the DCx Camera. Programming the uc480 ActiveX component is described in the uc480 ActiveX Manual. After the installation, you will find this manual in the `C:\Programs\uc480\Help directory`.

Note

ActiveX is only available on Windows systems.

Required Files

In order to access the uc480 ActiveX component, make sure to include the following file in your project:

- ActiveX control: `uc480Cam.ocx`

Hint

We suggest that you keep the function libraries (DLL, AX and OCX files) in the default directory. After the installation, these files reside e.g. under Window (32 bit) in `C:\Windows\System32\`.

Copying these files to other locations may result in version conflicts.

4.7.6 Thread Programming

In general, the uc480 API is thread-safe. This means that the uc480 API can be accessed by multiple threads simultaneously. Simultaneous attempts to call the same function are recognized and prevented by the driver.

Note

 © 2015 Thorlabs Scientific Imaging

Multi-threading

We recommend that you call the following functions from one thread per camera in order to avoid unpredictable behavior of the application.

- `is_InitCamera()`
- `is_SetDisplayMode()`
- `is_ExitCamera()`

Attention

Using USB cameras under Windows

The following events require a Windows message loop. This message loop has to be executed by the thread that loads the uc480 API. The message loop is usually provided by the application window. In some cases, the message loop might not be created automatically (e.g. in console applications). In this case you will need to implement the message loop yourself.

This applies to the following uc480 events:

- `IS_SET_EVENT_REMOVED`
- `IS_SET_EVENT_REMOVAL`
- `IS_SET_EVENT_DEVICE_RECONNECTED`
- `IS_SET_EVENT_NEW_DEVICE`

If no message loop exists, a USB camera will not be automatically detected after reconnecting.

4.8 Lists

- Complete list of all return values
- Error codes of AVI functions
- Linux: not supported functions

4.8.1 Complete List of All Return Values

No	Error	Description
-1	IS_NO_SUCCESS	General error message
0	IS_SUCCESS	Function executed successfully
1	IS_INVALID_CAMERA_HANDLE	Invalid camera handle Most of the uc480 SDK functions expect the camera handle as the first parameter.
2	IS_IO_REQUEST_FAILED	An IO request from the uc480 driver failed. Possibly the versions of the uc480.dll (API) and the driver file (uc480_usb.sys) do not match.
3	IS_CANT_OPEN_DEVICE	An attempt to initialize or select the camera failed (no camera connected or initialization error).
11	IS_CANT_OPEN_REGISTRY	Error opening a Windows registry key
12	IS_CANT_READ_REGISTRY	Error reading settings from the Windows registry
15	IS_NO_IMAGE_MEM_ALLOCATED	The driver could not allocate memory.
16	IS_CANT_CLEANUP_MEMORY	The driver could not release the allocated memory.
17	IS_CANT_COMMUNICATE_WITH_DRIVER	Communication with the driver failed because no driver has been loaded.
18	IS_FUNCTION_NOT_SUPPORTED_YET	The function is not supported yet.
32	IS_INVALID_CAPTURE_MODE	The function can not be executed in the current camera operating mode (free run, trigger or standby).
49	IS_INVALID_MEMORY_POINTER	Invalid pointer or invalid memory ID
50	IS_FILE_WRITE_OPEN_ERROR	File cannot be opened for writing or reading.
51	IS_FILEREAD_OPEN_ERROR	The file cannot be opened.
52	IS_FILE_READ_INVALID_BMP_ID	The specified file is not a valid bitmap file.
53	IS_FILE_READ_INVALID_BMP_SIZE	The bitmap size is not correct (bitmap too large).
108	IS_NO_ACTIVE_IMG_MEM	No active image memory available. You must set the memory to active using the is_SetImageMem() function or create a sequence using the is_AddToSequence() function.
112	IS_SEQUENCE_LIST_EMPTY	The sequence list is empty and cannot be deleted.

© 2015 Thorlabs Scientific Imaging

No	Error	Description
113	IS_CANT_ADD_TO_SEQUENCE	The image memory is already included in the sequence and cannot be added again.
117	IS_SEQUENCE_BUF_ALREADY_LOCKED	The memory could not be locked. The pointer to the buffer is invalid.
118	IS_INVALID_DEVICE_ID	The device ID is invalid. Valid IDs start from 1 for USB cameras, and from 1001 for GigE cameras.
119	IS_INVALID_BOARD_ID	The board ID is invalid. Valid IDs range from 1 through 255.
120	IS_ALL_DEVICES_BUSY	All cameras are in use.
122	IS_TIMED_OUT	A timeout occurred. An image capturing process could not be terminated within the allowable period.
123	IS_NULL_POINTER	Invalid array
125	IS_INVALID_PARAMETER	One of the submitted parameters is outside the valid range or is not supported for this sensor or is not available in this mode.
127	IS_OUT_OF_MEMORY	No memory could be allocated.
129	IS_ACCESS_VIOLATION	An internal error has occured.
139	IS_NO_USB20	The camera is connected to a port which does not support the USB 2.0 high-speed standard. Cameras without a memory board cannot be operated on a USB 1.1 port.
140	IS_CAPTURE_RUNNING	A capturing operation is in progress and must be terminated before you can start another one.
145	IS_IMAGE_NOT_PRESENT	The requested image is not available in the camera memory or is no longer valid.
148	IS_TRIGGER_ACTIVATED	The function cannot be used because the camera is waiting for a trigger signal.
151	IS_CRC_ERROR	A CRC error-correction problem occurred while reading the settings.
152	IS_NOT_YET_RELEASED	This function has not been enabled yet in this version.
153	IS_NOT_CALIBRATED	The camera does not contain any calibration data.
154	IS_WAITING_FOR_KERNEL	The system is waiting for the kernel driver to respond.
155	IS_NOT_SUPPORTED	The camera model used here does not support this function or setting.
156	IS_TRIGGER_NOT_ACTIVATED	The function is not possible as trigger is disabled.
157	IS_OPERATION_ABORTED	The dialog was canceled without a selection so that no file could be saved.

No	Error	Description
158	IS_BAD_STRUCTURE_SIZE	An internal structure has an incorrect size.
159	IS_INVALID_BUFFER_SIZE	The image memory has an inappropriate size to store the image in the desired format.
160	IS_INVALID_PIXEL_CLOCK	This setting is not available for the currently set pixel clock frequency.
161	IS_INVALID_EXPOSURE_TIME	This setting is not available for the currently set exposure time.
162	IS_AUTO_EXPOSURE_RUNNING	This setting cannot be changed while automatic exposure time control is enabled.
163	IS_CANNOT_CREATE_BB_SURF	The BackBuffer surface cannot be created.
164	IS_CANNOT_CREATE_BB_MIX	The BackBuffer mix surface cannot be created.
165	IS_BB_OVLMEM_NULL	The BackBuffer overlay memory cannot be locked.
166	IS_CANNOT_CREATE_BB_OVL	The BackBuffer overlay memory cannot be created.
167	IS_NOT_SUPP_IN_OVL_SURF_MODE	Not supported in BackBuffer Overlay mode.
168	IS_INVALID_SURFACE	Back buffer surface invalid.
169	IS_SURFACE_LOST	Back buffer surface not found.
170	IS_RELEASE_BB_OVL_DC	Error releasing the overlay device context.
171	IS_BB_TIMER_NOT_CREATED	The back buffer timer could not be created.
172	IS_BB_OVL_NOT_EN	The back buffer overlay was not enabled.
173	IS_ONLY_IN_BB_MODE	Only possible in BackBuffer mode.
174	IS_INVALID_COLOR_FORMAT	Invalid color format
175	IS_INVALID_WB_BINNING_MODE	Mono binning/mono sub-sampling do not support automatic white balance.
176	IS_INVALID_I2C_DEVICE_ADDRESS	Invalid I²C device address
177	IS_COULD_NOT_CONVERT	The current image could not be processed.
178	IS_TRANSFER_ERROR	Transfer error. Frequent transfer errors can mostly be avoided by reducing the pixel rate.
179	IS_PARAMETER_SET_NOT_PRESENT	Parameter set is not present.
180	IS_INVALID_CAMERA_TYPE	The camera type defined in the .ini file does not match the current camera model.
181	IS_INVALID_HOST_IP_HIBYTE	Invalid HIBYTE of host address
182	IS_CM_NOT_SUPP_IN_CURR_DISPLAYMODE	The color mode is not supported in the current display mode.
183	IS_NO_IR_FILTER	No IR filter available
184	IS_STARTER_FW_UPLOAD_NEEDED	The camera's starter firmware is not compatible with the driver and needs to be updated.
185	IS_DR_LIBRARY_NOT_FOUND	The DirectRenderer library could not be found.
186	IS_DR_DEVICE_OUT_OF_MEMORY	Not enough graphics memory available.

© 2015 Thorlabs Scientific Imaging

No	Error	Description
187	IS_DR_CANNOT_CREATE_SURFACE	The image surface or overlay surface could not be created.
188	IS_DR_CANNOT_CREATE_VERTEX_BUFFER	The vertex buffer could not be created.
189	IS_DR_CANNOT_CREATE_TEXTURE	The texture could not be created.
190	IS_DR_CANNOT_LOCK_OVERLAY_SURFACE	The overlay surface could not be locked.
191	IS_DR_CANNOT_UNLOCK_OVERLAY_SURFACE	The overlay surface could not be unlocked.
192	IS_DR_CANNOT_GET_OVERLAY_DC	Could not get the device context handle for the overlay.
193	IS_DR_CANNOT_RELEASE_OVERLAY_DC	Could not release the device context handle for the overlay.
194	IS_DR_DEVICE_CAPS_INSUFFICIENT	Function is not supported by the graphics hardware.
195	IS_INCOMPATIBLE_SETTING	Because of other incompatible settings the function is not possible.
196	IS_DR_NOT_ALLOWED_WHILE_DC_IS_ACTIVE	A device context handle is still open in the application.
197	IS_DEVICE_ALREADY_PAIRED	The device is already paired.
198	IS_SUBNETMASK_MISMATCH	The subnet mask of the camera and PC network card are different.
199	IS_SUBNET_MISMATCH	The subnet of the camera and PC network card are different.
200	IS_INVALID_IP_CONFIGURATION	The configuration of the IP address is invalid.
201	IS_DEVICE_NOT_COMPATIBLE	The device is not compatible to the drivers.
202	IS_NETWORK_FRAME_SIZE_INCOMPATIBLE	The settings for the image size of the camera are not compatible to the PC network card.
203	IS_NETWORK_CONFIGURATION_INVALID	The configuration of the network card is invalid.
204	IS_ERROR_CPU_IDLE_STATES_CONFIGURATION	The configuration of the CPU idle has failed.
205	IS_DEVICE_BUSY	The camera is busy ad cannot transfer the requested image.

4.8.2 Error Codes of AVI Functions

No.	Error	Description
300	IS_AVI_NO_ERR	Function executed successfully.
301	IS_AVI_ERR_INVALID_FILE	The file has no valid AVI format.
302	IS_AVI_ERR_NEW_FAILED	No memory could be allocated for the AVI file.
303	IS_AVI_ERR_CREATESTREAM	No AVI stream could be created.
304	IS_AVI_ERR_PARAMETER	One of the submitted parameters is outside the valid range.
305	IS_AVI_ERR_NO_CODEC_AVAIL	The maximum number of instances allowed in this system has been reached. It is not possible to create another instance.
306	IS_AVI_ERR_INVALID_ID	The indicated AVI instance could not be found. Either the AVI ID is invalid or the instance has already been deleted using isavi_ExitAVI().
307	IS_AVI_ERR_COMPRESS	The last image compression failed.
309	IS_AVI_ERR_CAPTURE_RUNNING	Another capturing operation is in progress or an AVI file is still open.
310	IS_AVI_ERR_CAPTURE_NOT_RUNNING	No capturing operation is running or no AVI file is opened.
311	IS_AVI_ERR_PLAY_RUNNING	A playback is already running.
312	IS_AVI_ERR_PLAY_NOT_RUNNING	No playback is running.
313	IS_AVI_ERR_WRITE_INFO	The AVI file could not be modified.
314	IS_AVI_ERR_INVALID_VALUE	The q parameter is outside the range of 1…100.
315	IS_AVI_ERR_ALLOC_MEMORY	No memory could be allocated.
316	IS_AVI_ERR_INVALID_CM	The submitted color mode is not supported for AVI capturing.
317	IS_AVI_ERR_COMPRESSION_RUN	The current image could not be processed since compression is still in progress.
318	IS_AVI_ERR_INVALID_SIZE	The submitted size is invalid.
319	IS_AVI_ERR_INVALID_POSITION	The submitted position is invalid.
320	IS_AVI_ERR_INVALID_UEYE	No DCx camera was found.
321	IS_AVI_ERR_EVENT_FAILED	The event could not be initialized. The Windows SetEvent function failed.

4.8.3 Linux: Not Supported Functions

The following uc480 API functions are not supported by the Linux driver version 4.60 and older:

is_DirectRenderer()
is_GetColorDepth()
is_GetDC()
is_ReleaseDC()

© 2015 Thorlabs Scientific Imaging

is_RenderBitmap()
is_SetDisplayMode()
is_SetDisplayPos()
is_SetHwnd()
is_SetOptimalCameraTiming()
is_UpdateDisplay()

5 Specifications

This chapter lists the specifications of the available DCx camera models.

- Model comparison
- Camera and sensor data
- Mechanical specifications
- Electrical specifications
- Accessories

© 2015 Thorlabs Scientific Imaging

5.1 Model Comparison

The following table outlines the key features of DCx camera series for direct comparison (see also the DCx Camera Family chapter).

	DCC1545M DCC1645C	DCU223x DCU224x	DC1240x	DC3240x
Sensor Type	CMOS	CCD	CMOS	CMOS
Objective Mount	CS (C*)	C	C	C
EMC compliance	CE B FCC B	CE A FCC A	CE B FCC A	CE A FCC A
WxHxD [mm] (CCD size)	44 x 44 x 25.6	32 x 34 x 30.2 (37.2)	32 x 34 x 30.2	29 x 29 x 29
Mounting holes bottom top side	 1 0 0	 2 3 3	 2 3 3	 2 3 3
Thread diameter	1/4" (#8-32, M4)*)	#8-32, M4	#8-32, M4	1/4" (#8-32, M4)*)
Adjustable flange back distance	+	-	-	-
IP protection class	30	30	30	30
Interface	USB 2.0	USB 2.0	USB 2.0	USB 3.0 USB 2.0
Power supply	USB	USB	USB	USB
Lockable connector	-	+	+	+
I/O connector	10-pin connector	9-pin micro D-SUB	9-pin micro D-SUB	8-pin HR25
Optocoupler for I/O	-/-	1/1	1/1	1/1
Optocoupler speed	-	100 µs	100 µs	High (trigger)
Max. cable length (m)	5	5	5	5
Dig. I/O/GPIO	0/0/0	1/1/0	1/1/0	1/1/2
PWM	-	-	-	+
RS-232	-	-	-	+ (GPIO)
I^2C bus	-	-	-	-
Hot pixel correction	Software	Software	Software	Hardware
Color calculation	Software	Software	Software	Hardware
Hardware: Dig. Subsampling/Binning	-	-	-	+
Bit depth: Internal/ transferred	8/8	8/8	8/8	16/16
LUT: Internal/ transferred	-	-	-	12/12

	DCC1545M DCC1645C	DCU223x DCU224x	DC1240x	DC3240x
Max. pixel clock (MHz) at full resolution	43	43	43	480
Max. data rate (MBytes/s)	38	38	38	300

*) with supplied adapter

© 2015 Thorlabs Scientific Imaging

5.2 Model Naming Conventions

In the table below, you will find an overview of all the models in the DCx camera series.

			DC	X	##	#	X
Short for "Digital Camera							
C	CMOS						
U	CCD						
Sensor and shutter type							
12 15/16 22 32	CMOS Global Shutter CMOS Rolling Shutter CCD Progressive Scan CMOS Global Shutter, USB3						
Resolution							
3 4, 40, 45	XGA SXGA	1024 x 768 (0.78 MPixel) 1280 x 1024 (1.30 MPixel)					
	Color / Mono						
M C N	Monochrome sensor Color sensor Near Infrared sensor						

5.3 Camera and Sensor Data

In this chapter the technical properties of the sensors are listed. You can look up e.g. binning factors. Also you will find the parameters for the different interfaces.

Cameras with CMOS sensors

- DCC1240x / DCC3240x
- DCC1545M
- DCC1645C

Cameras with CCD sensors

- DCU223x
- DCU224x

Note:

The diagrams shown in the sensor specifications section indicate the **relative** sensitivities of the DCx Cameras in the spectral range. Therefore, the characteristic curves cannot be compared to each other.

5.3.1 DCC1240x / DCC3240x

Sensor specification			
Sensor type	CMOS		
Shutter system	Electronic global and rolling shutter		
Characteristic	Linear		
Readout mode	Progressive scan		
Resolution class	SXGA		
Resolution	1280 x 1024 pixels (1.3 Megapixel)		
Aspect ratio	5:4		
Bit depth	10 bits *8		
Optical sensor class	1/1.8 inch		
Exact sensitive area	6.784 mm x 5.427 mm		
Exact optical sensor diagonal	8.69 mm (1/1.84 inch)		
Pixel size	5.30 µm, square		
Micro lens shift	12°		
Sensor name, monochrome	e2v EV76C560ABT		
Sensor name, color	e2v EV76C560ACT		
Sensor name, NIR *7	e2v EV76C661ABT		
Special features	• Automatic hotpixel correction in the sensor, see `is_HotPixel()` • Multi AOI with 2 or 4 AOI, see Camera basics: AOI • Sequence AOI • Sensor internal image scaler, downscaling by factor 1...4, see `is_SetSensorScaler()` and uc480 Viewer: Size) • Allows to switch between global and rolling shutter readout, see `is_DeviceFeature()`		
Gain			
Monochrome model (master gain)	4x		
Color model (master / RGB)	4x / 3.96x		
Gain boost	2x		
Camera timing		DCC1240x	DCC3240x
Pixel clock range (allowed/ recommended)	MHz	7 to 35 / 35*1	5 to 85 / 85*1
Max. pixel clock with subsampling/binning	MHz	85*1	85*1
Frame rate (freerun mode)	fps	25.8*2	60.0*2
Frame rate (trigger mode, 1 ms exposure)	fps	24.7*2	56.9*2
Exposure time in freerun mode	ms	0.009*2 to 2000*3	0.009*2 to 2000*3
Exposure time in trigger mode	ms	0.009*2 to 2000*3	0.009*2 to 2000*3
AOI			
Mode		Horizontal + Vertical*4	

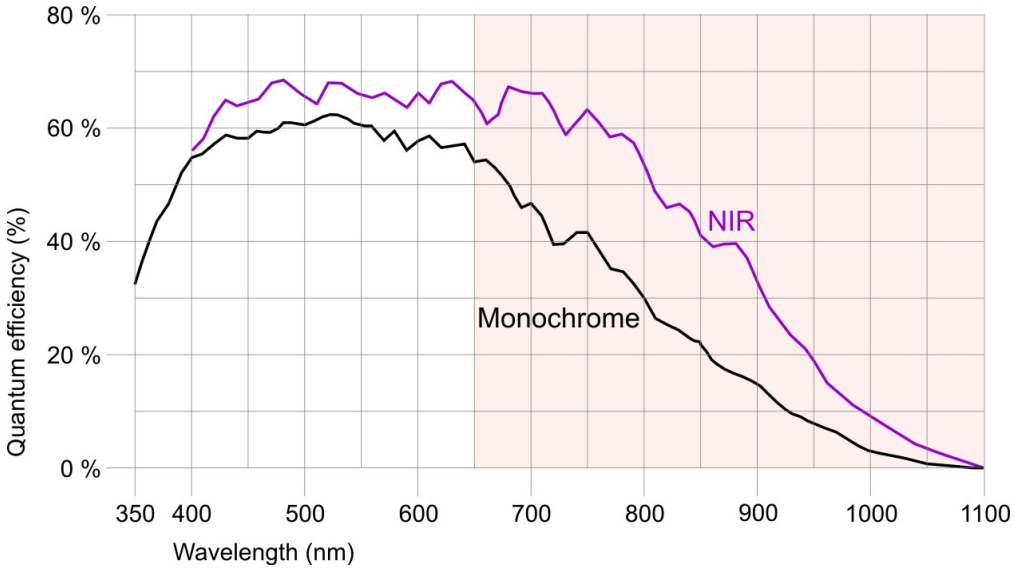
© 2015 Thorlabs Scientific Imaging

AOI image width, step width	Pixels	16 to 1280, 4	16 to 1280, 4
AOI image height, step width	Pixels	4 to 1024, 2	4 to 1024, 2
AOI position grid horizontal, vertical	Pixels	2, 2	2, 2
AOI frame rate, 640 x 480 pixels (VGA)	fps	52.0	123.0
AOI frame rate, 320 x 240 pixels (CIF)	fps	98.0	229.0
Binning			
Mode		Horizontal + Vertical*4	
Method		H + V combined, mono/color binning, H: additive. V: averaging	
Factors		2x	
Frame rate with 2x binning, 640 x 480 pixels (VGA)	fps	60.0	64.0
Subsampling			
Mode		Scaler	
Hardware trigger			
Mode		Asynchronous	Asynchronous
Trigger delay with rising edge	µs	20 ±0.25	3 ±0.25*6
Trigger delay with falling edge	µs	33 ±0.25	21 ±0.25*6
Additive trigger delay (optional)	µs	15 µs...4 s	15 µs...4 s
Power consumption[*5]			
	W	0.3 to 0.7	1.3*5

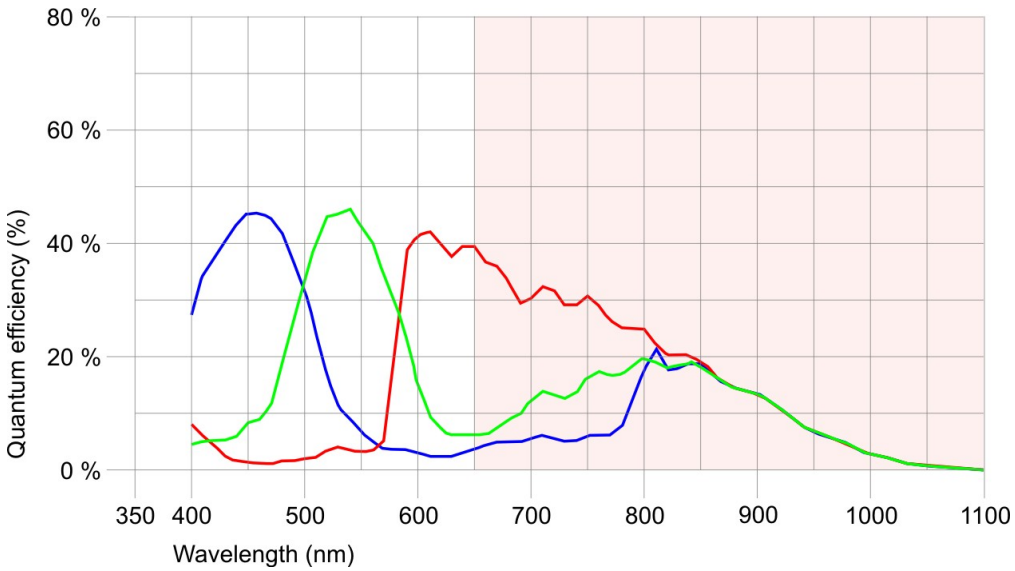
** Not yet defined.
*1 The maximum possible pixel clock frequency depends on the PC hardware used.
*2 Requires maximum pixel clock frequency.
*3 Requires minimum pixel clock frequency.
*4 Use of this function increases the frame rate.
*5 The power consumption depends on the sensor model and the pixel clock setting.
*6 Not yet confirmed
*7 DCC3240N only
*8 Only for USB3.0 transmission (DCC3240x), with USB2.0 - bit depth is 8 bit.

Please see also the DCC1240x / DCC3240x Application Notes chapter.

Relative sensor sensitivity



Sensor sensitivity of the DCC1240M (monochrome) / DCC3240x (monochrome and NIR)



Sensor sensitivity of the DCC1240C / DCC3240C

Note

The colored part of above diagrams just indicates the IR wavelength range in order to tell it from the visible.

© 2015 Thorlabs Scientific Imaging

5.3.2 DCC1545M

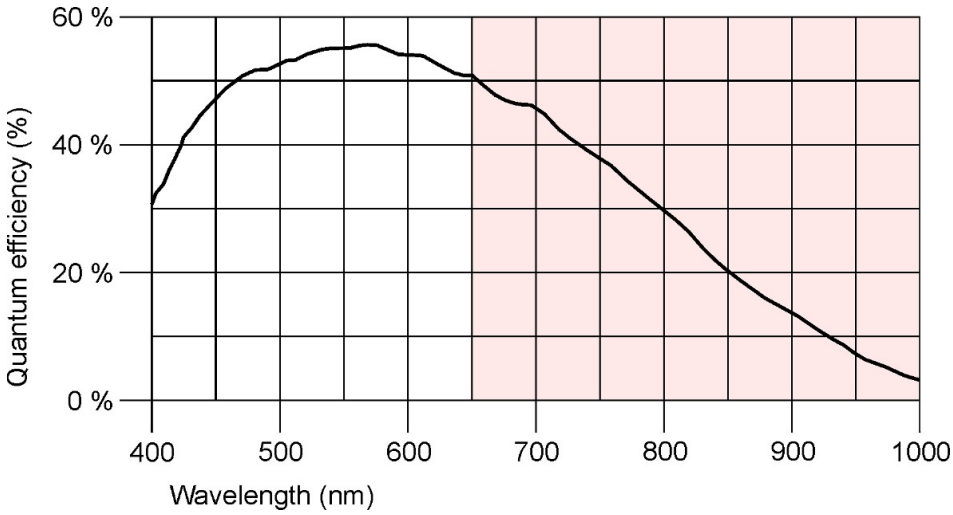
Sensor specification		
Sensor type	CMOS	
Shutter system	Electronic rolling shutter	
Readout mode	Progressive scan	
Resolution class	SXGA	
Resolution	1280 x 1024 pixels (1.31 Megapixel)	
Aspect ratio	5:4	
Bit depth	10 bits	
Optical sensor class	1/2 inch	
Exact sensitive area	6.656 x 5.325 mm	
Exact optical sensor diagonal	8.52 mm (1/1.88 inch)	
Pixel size	5.20 µm, square	
Sensor name	Aptina MT9M001 (monochrome)	
Gain		
Monochrome model (master gain)	13x	
Analog gain boost	1.5x	
Camera timing		
Pixel clock range	MHz	5 to 43[*1]
Max. pixel clock with subsampling/ binning	MHz	48[*1]
Frame rate (freerun mode)	fps	25.0[*2]
Frame rate (trigger mode, 1 ms exposure)	fps	25.0[*2]
Exposure time in freerun mode	ms	0.037[*2] to 983[*2]
Exposure time in trigger mode	ms	0.037[*2] to 983[*2]
AOI		
Mode		Horizontal[*4] + Vertical[*4]
AOI image width, step width	Pixels	32 to 1280, 4
AOI image height, step width	Pixels	4 to 1024, 2
AOI position grid horizontal, vertical	Pixels	4, 2
AOI frame rate, 640 x 480 pixels (VGA)	fps	84
Binning		
Mode		none
Subsampling		
Mode		Horizontal[*4] + Vertical[*4]
Method		H + V: Color subsampling
Factor		2x, 4x, 8x
Frame rate w/ 2x subsampling,	fps	94

640 x 480 pixels		
Frame rate w/ 4x subsampling, 320 x 240 pixels	fps	258
Hardware trigger		
Mode		Asynchronous
Trigger delay with rising edge	µs	22.0 ±0.25
Trigger delay with falling edge	µs	40.3 ±0.25
Additive trigger delay (optional)	µs	15 µs...4 s
Power consumption[*5]		
	W	0.5 to 1.0

*[1] The maximum possible pixel clock frequency depends on the PC hardware used.
*[2] Requires maximum pixel clock frequency.
*[3] Requires minimum pixel clock frequency.
*[4] Use of this function increases the frame rate.
*[5] The power consumption depends on the sensor model and the pixel clock setting.

Please see also the DCC1545M Application Notes chapter.

Relative sensor sensitivity



Sensor sensitivity of the DCC1545M (monochrome)

Note

The colored part of above diagram just indicates the IR wavelength range in order to tell it from the visible.

© 2015 Thorlabs Scientific Imaging

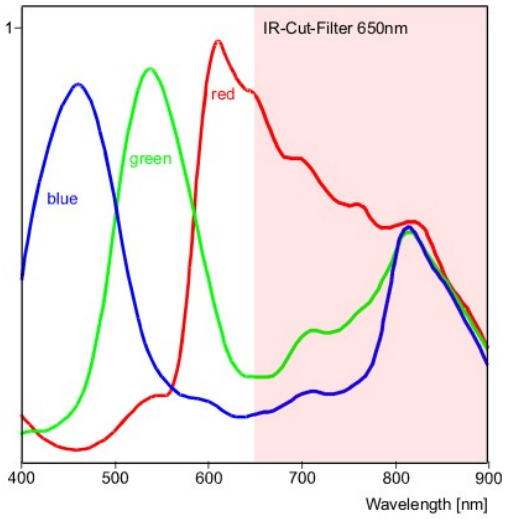
5.3.3 DCC1645C

Sensor specification		
Sensor type	CMOS	
Shutter system	Electronic rolling shutter	
Characteristic	Linear	
Readout mode	Progressive scan	
Resolution class	SXGA	
Resolution	1280 x 1024 pixels (1.31 Megapixel)	
Aspect ratio	5:4	
Bit depth	10 bits	
Optical sensor class	1/3 inch	
Exact sensitive area	4.608 x 3.686 mm	
Exact optical sensor diagonal	5.90 mm (1/2.71 inch)	
Pixel size	3.60 µm, square	
Micro lens shift	25°	
Sensor name	Aptina MT9M131 (color)	
Gain		
Color model (master/RGB)	4.27x / 3.1x	
Analog gain boost	2.0x	
Camera timing		
Pixel clock range	MHz	5 to 40[*1]
Max. pixel clock with subsampling/ binning	MHz	40[*1]
Frame rate (freerun mode)	fps	25.0[*2]
Frame rate (trigger mode, 1 ms exposure)	fps	24.9[*2]
Exposure time in freerun mode	ms	0.037[*2] to 10122[*3]
Exposure time in trigger mode	ms	0.037[*2] to 10122[*3]
AOI		
Mode		Horizontal[*4] + Vertical[*4]
AOI image width, step width	Pixels	32 to 1280, 4
AOI image height, step width	Pixels	4 to 1024, 2
AOI position grid horizontal, vertical	Pixels	4, 2
AOI frame rate, 1280 x 720 pixels (HD 720)	fps	34
AOI frame rate, 800 x 600 pixels (SVGA)	fps	61
Binning		
Mode		none
Subsampling		
Mode		Horizontal[*4] + Vertical[*4]

Method		H + V: Color subsampling
Factor		2x, 4x
Frame rate w/ 2x subsampling, 640 x 480 pixels	fps	89
Frame rate w/ 4x subsampling, 320 x 240 pixels	fps	263
Hardware trigger		
Mode		Asynchronous
Trigger delay with rising edge	µs	180.9 ±0.25
Trigger delay with falling edge	µs	199.3 ±0.25
Additive trigger delay (optional)	µs	15 µs...4 s
Power consumption[*5]		
	W	0.3 to 0.8

*[1] The maximum possible pixel clock frequency depends on the PC hardware used.
*[2] Requires maximum pixel clock frequency.
*[3] Requires minimum pixel clock frequency.
*[4] Use of this function increases the frame rate.
*[5] The power consumption depends on the sensor model and the pixel clock setting.
Please see also the DCC1645C Application Notes chapter.

Relative sensor sensitivity



Sensor sensitivity of the DCC1645C

Note

The colored part of above diagram just indicates the IR wavelength range in order to tell it from the visible.

© 2015 Thorlabs Scientific Imaging

5.3.4 DCU223x

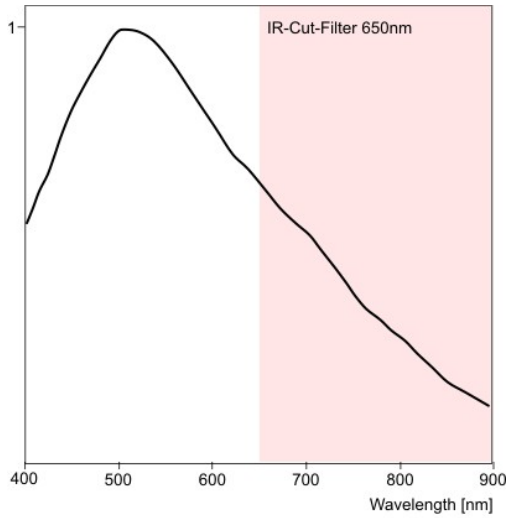
Sensor specification		
Sensor type	CCD	
Shutter system	Electronic global shutter	
Characteristic	Linear	
Readout mode	Progressive scan	
Resolution class	XGA	
Resolution	1024 x 768 pixels (0.79 Megapixel)	
Aspect ratio	4:3	
Bit depth	12 bits	
Optical sensor class	1/3 inch	
Exact sensitive area	4.762 x 3.571 mm	
Exact optical sensor diagonal	5.95 mm (1/2.69 inch)	
Pixel size	4.65 µm, square	
Sensor name, monochrome	Sony ICX204AL	
Sensor name, color	Sony ICX204AK	
Gain		
Monochrome model (master gain)	10.47x	
Color model (master/RGB)	7.59x/4.0x	
Analog gain boost	2.0x (monochrome model only)	
Camera timing		
Pixel clock range (allowed/ recommended)	MHz	5 to 30 / 10 to 20[*1]
Pixel clock for optimal image quality	MHz	15[*1]
Max. pixel clock with subsampling/binning	MHz	30[*1]
Frame rate (freerun mode)	fps	30.0[*2]
Frame rate (trigger mode, 1 ms exposure)	fps	28.7[*2]
Exposure time in freerun mode	ms	0.030[*2] to 773[*3]
Exposure time in trigger mode	ms	0.030[*2] to 10 min[*3]
AOI		
Mode		Horizontal + Vertical[*4]
AOI image width, step width	Pixels	16 to 1024, 4
Mono: AOI image height, step width	Pixels	120 to 768, 1
Color: AOI image height, step width	Pixels	120 to 768, 2
Mono: AOI position grid horizontal, vertical	Pixels	1, 1

Color: AOI position grid horizontal, vertical	Pixels	2, 2
AOI frame rate, 800 x 600 pixels (SVGA)	fps	37
AOI frame rate, 640 x 480 pixels (VGA)	fps	45
AOI frame rate, 320 x 240 pixels (CIF)	fps	78
Binning		
Mode		Vertical*4
Method		V: Monochrome binning, additive
Factor		2x, 3x, 4x
Frame rate with 2x binning, 1024 x 384 pixels	fps	53
Frame rate with 3x binning, 1024 x 256 pixels	fps	71
Frame rate with 4x binning, 1024 x 192 pixels	fps	85
Subsampling		
Mode		-
Hardware trigger		
Mode		Asynchronous
Trigger delay with rising edge	µs	39.5 ±2.6
Trigger delay with falling edge	µs	57.9 ±2.6
Additive trigger delay (optional)	µs	15 µs...4 s
Power consumption[*5]		
	W	1.0 to 1.7

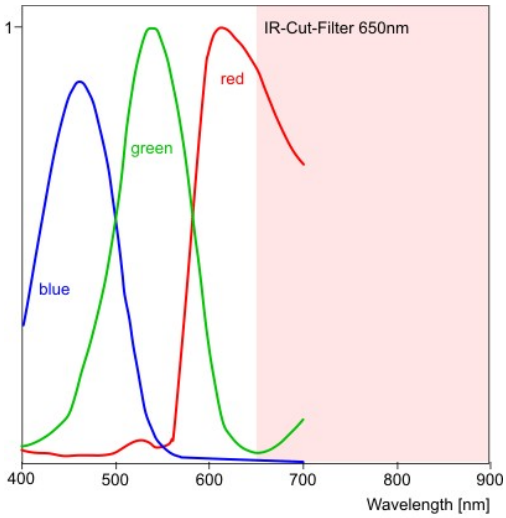
*1 The maximum possible pixel clock frequency depends on the PC hardware used.
*2 Requires maximum pixel clock frequency.
*3 Requires minimum pixel clock frequency.
*4 Use of this function increases the frame rate.
*5 The power consumption depends on the sensor model and the pixel clock setting.
Please see also the DCU223x Application Notes chapter.

Relative sensor sensitivity

 © 2015 Thorlabs Scientific Imaging



Sensor sensitivity of the DCU223M (monochrome)



Sensor sensitivity of the DCU223C (Color)

Note

The colored part of above diagrams just indicates the IR wavelength range in order to tell it from the visible.

5.3.5 DCU224x

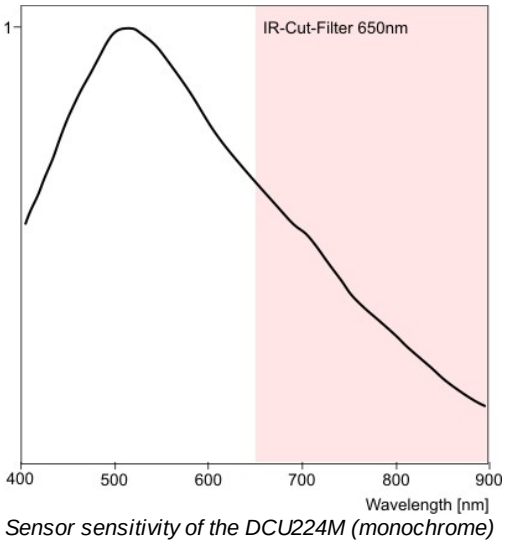
Sensor specification		
Sensor type	CCD	
Shutter system	Electronic global shutter	
Characteristic	Linear	
Readout mode	Progressive scan	
Resolution class	SXGA	
Resolution	1280 x 1024 pixels (1.31 Megapixel)	
Aspect ratio	5:4	
Bit depth	12 bits	
Optical sensor class	1/2 inch	
Exact sensitive area	5.952 x 4.762 mm	
Exact optical sensor diagonal	7.62 mm (1/2.1 inch)	
Pixel size	4.65 µm, square	
Sensor name, monochrome	Sony ICX205AL	
Sensor name, color	Sony ICX205AK	
Gain		
Monochrome model (master gain)	13.66x	
Color model (master/RGB)	8.9/4.0x	
Analog gain boost	2.0x (monochrome model only)	
Camera timing		
Pixel clock range (allowed/ recommended)	MHz	5 to 30 / 10 to 20[*1]
Pixel clock for optimal image quality	MHz	14[*1]
Max. pixel clock with subsampling/ binning	MHz	30[*1]
Frame rate (freerun mode)	fps	15.0[*2]
Frame rate (trigger mode, 1 ms exposure)	fps	17.0[*2]
Exposure time in freerun mode	ms	0.066[*2] to 1360[*3]
Exposure time in trigger mode	ms	0.066[*2] to 10 min[*3]
AOI		
Mode		Horizontal + Vertical[*4]
AOI image width, step width	Pixels	16 to 1280, 4
Mono: AOI image height, step width	Pixels	120 to 1024, 1
Color: AOI image height, step width	Pixels	120 to 1024, 2
Mono: AOI position grid horizontal, vertical	Pixels	1, 1
Color: AOI position grid horizontal,	Pixels	2, 2

© 2015 Thorlabs Scientific Imaging

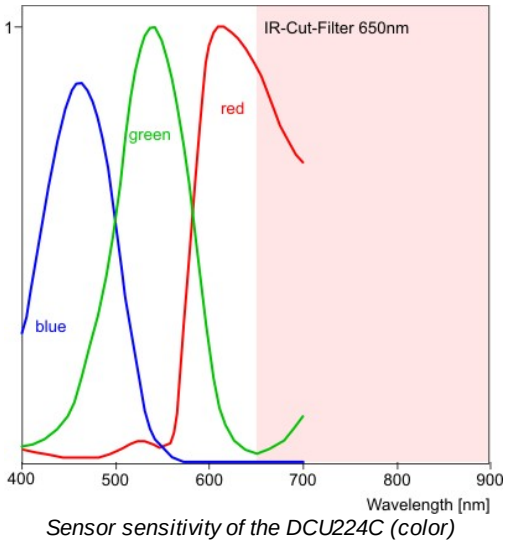
vertical		
AOI frame rate, 1024 x 768 pixels (XGA)	fps	18
AOI frame rate, 640 x 480 pixels (VGA)	fps	28
AOI frame rate, 320 x 240 pixels (CIF)	fps	38
Binning		
Mode		Vertical*4
Method		V: Monochrome binning, additive
Factor		2x, 3x, 4x
Frame rate with 2x binning, 1280 x 512 pixels	fps	23
Frame rate with 3x binning, 1280 x 340 pixels	fps	28
Frame rate with 4x binning, 1280 x 256 pixels	fps	31
Subsampling		
Mode		Vertical*4
Method		V: Color subsampling
Factor		4x
Frame rate w/ 4x subsampling, 1280 x 256 pixels	fps	31
Hardware trigger		
Mode		Asynchronous
Trigger delay with rising edge	µs	39.9 ±2.5
Trigger delay with falling edge	µs	57.7 ±2.5
Additive trigger delay (optional)	µs	15 µs...4 s
Power consumption*5		
	W	1.1 to 2.1

*1 The maximum possible pixel clock frequency depends on the PC hardware used.
*2 Requires maximum pixel clock frequency.
*3 Requires minimum pixel clock frequency.
*4 Use of this function increases the frame rate.
*5 The power consumption depends on the sensor model and the pixel clock setting.
Please see also the DCU224x Application Notes chapter.

Relative sensor sensitivity



Sensor sensitivity of the DCU224M (monochrome)



Sensor sensitivity of the DCU224C (color)

Note

The colored part of above diagrams just indicates the IR wavelength range in order to tell it from the visible.

© 2015 Thorlabs Scientific Imaging

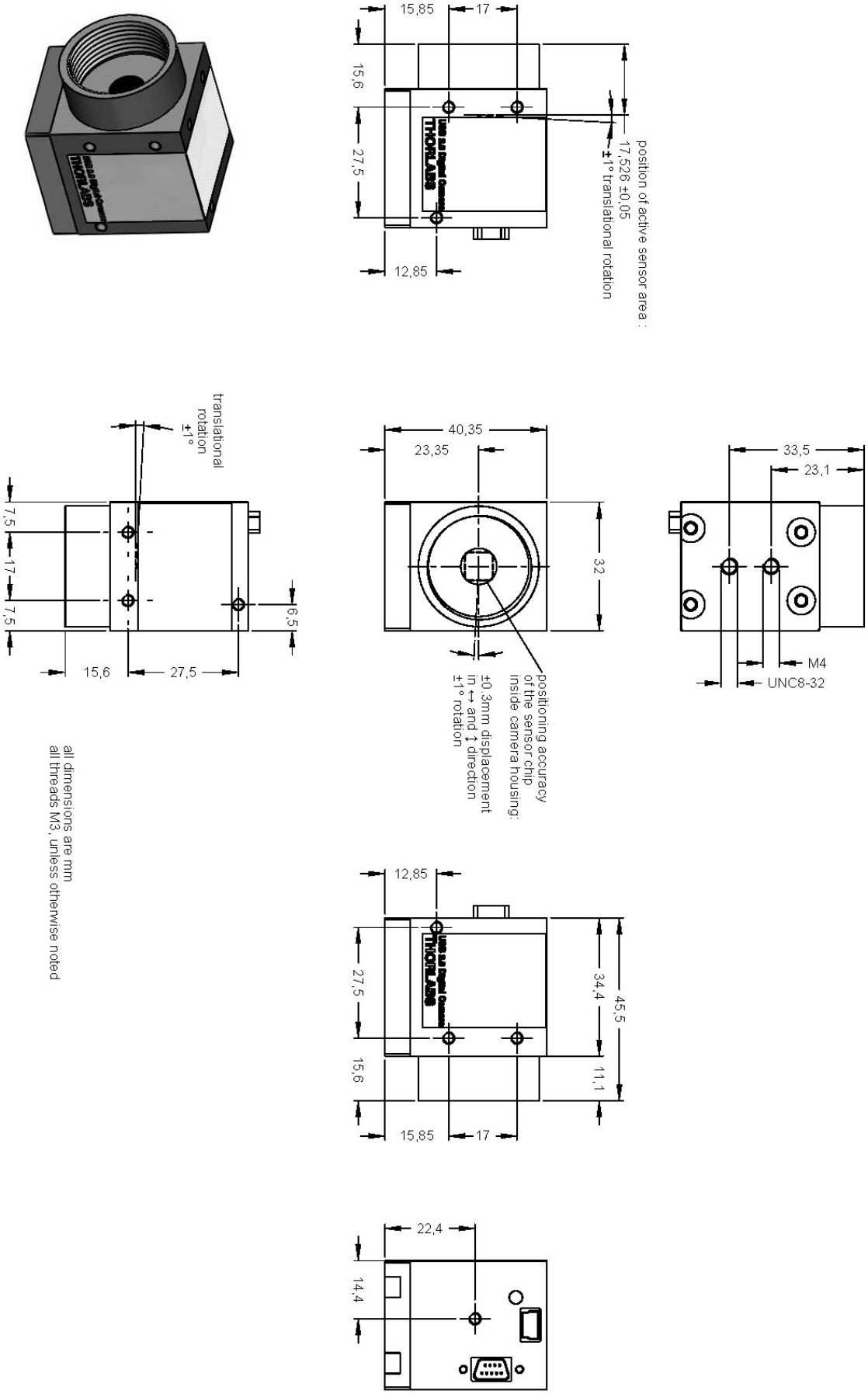
5.4 Mechanical Specifications

- DCU223x, DCU224x
- DCC1545M, DCC1645C
- DCC1240x, DCC3240x

General

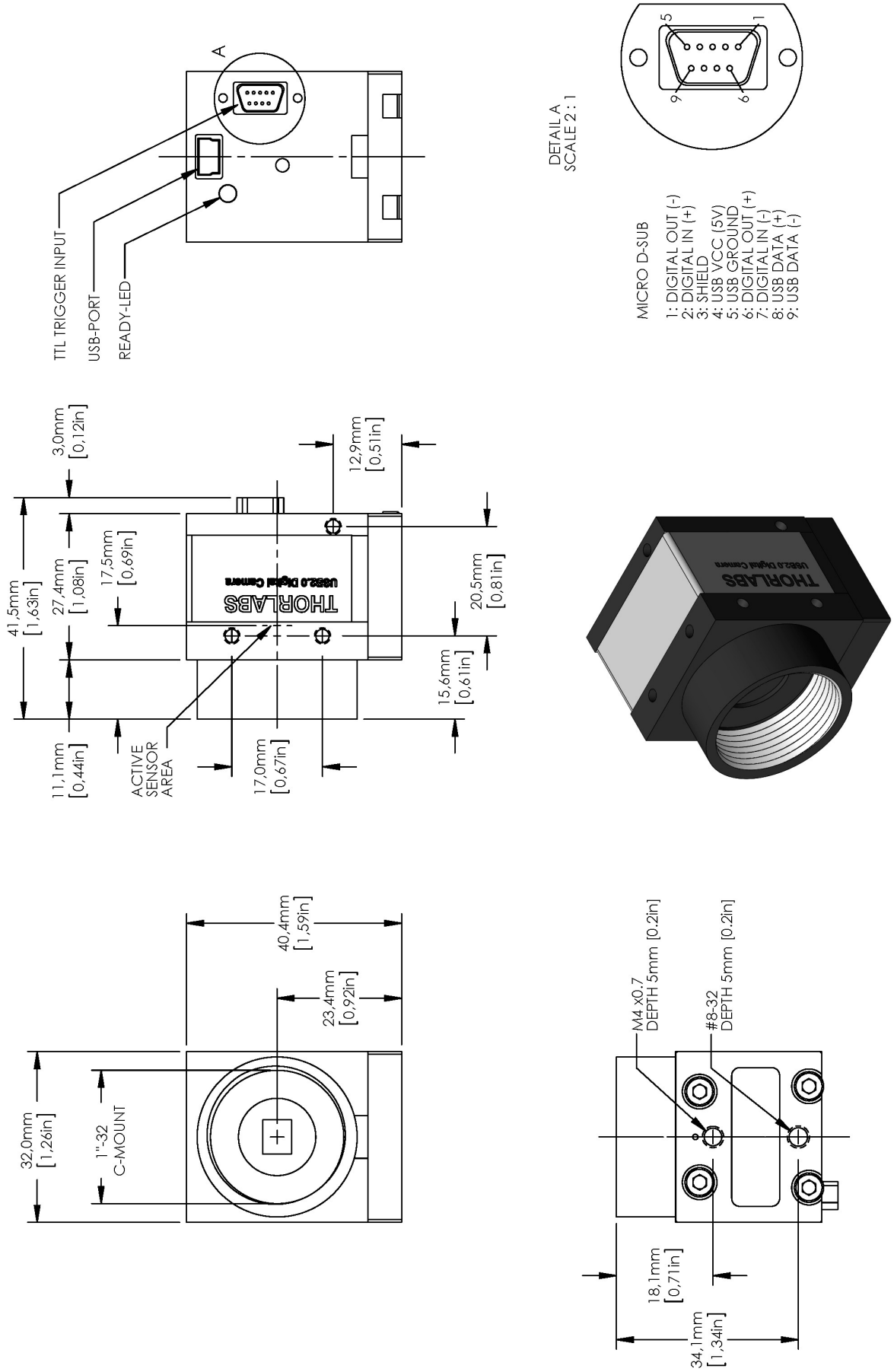
- Flange back distance
- Position accuracy of the sensor
- Filter glasses
- Ambient conditions

5.4.1 DCU223x, DCU224x

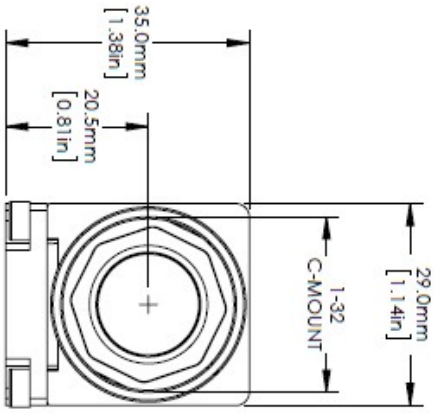


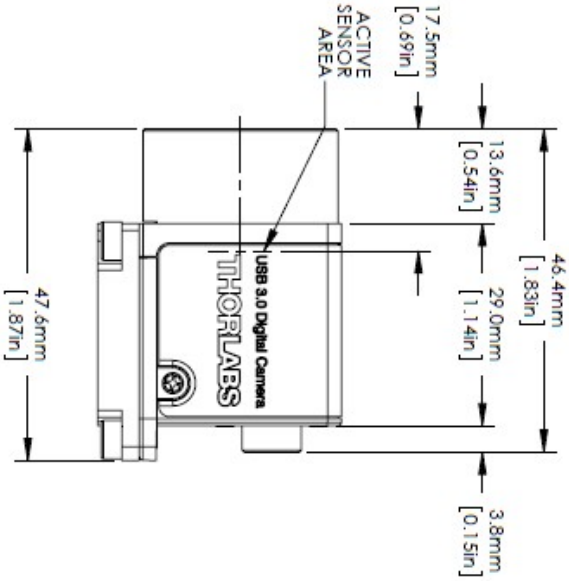
© 2015 Thorlabs Scientific Imaging

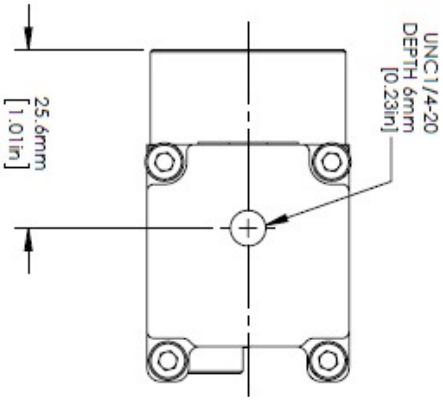
5.4.2 DCC1240x

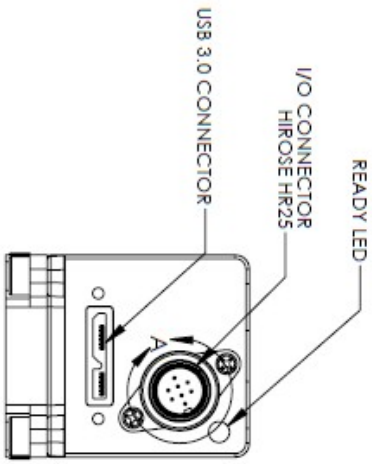


5.4.3 DCC3240x

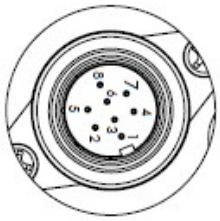








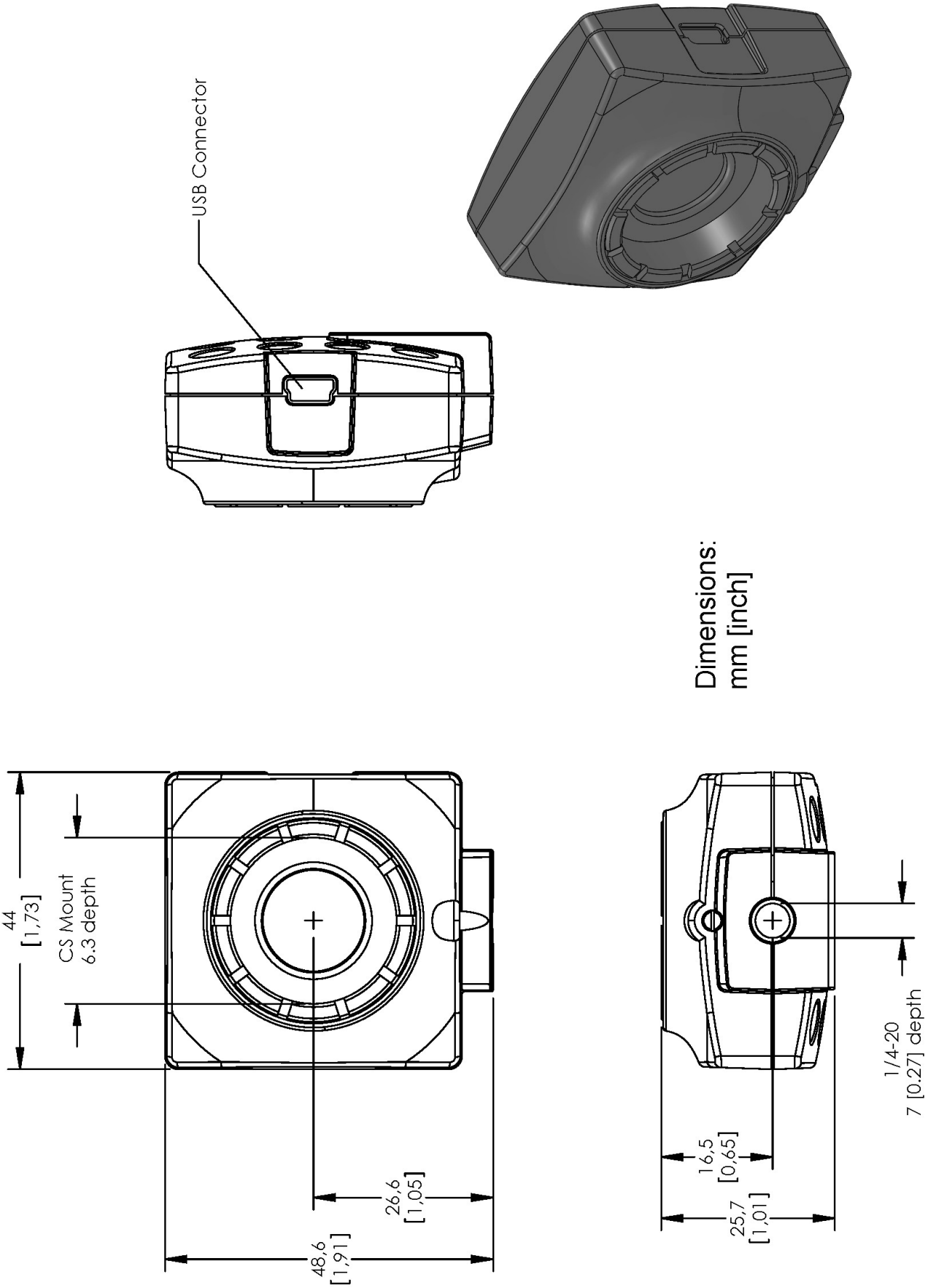
PIN	DESCRIPTION
1	GROUND
2	FLASH OUTPUT, OPTO-DECOUPLED (-)
3	GPIO 1, 3.3V LVCMOS
4	TRIGGER INPUT,OPTO-DECOUPLED (-)
5	FLASH OUTPUT, OPTO-DECOUPLED (+)
6	GPIO 2, 3.3V LVCMOS
7	TRIGGER INPUT,OPTO-DECOUPLED (+)
8	OUTPUT SUPPLY VOLTAGE, 5V (100mA)



DETAIL A
SCALE 2 : 1
REAR VIEW

© 2015 Thorlabs Scientific Imaging

5.4.4 DCC1545M, DCC1645C



5.4.5 Flange Back Distance

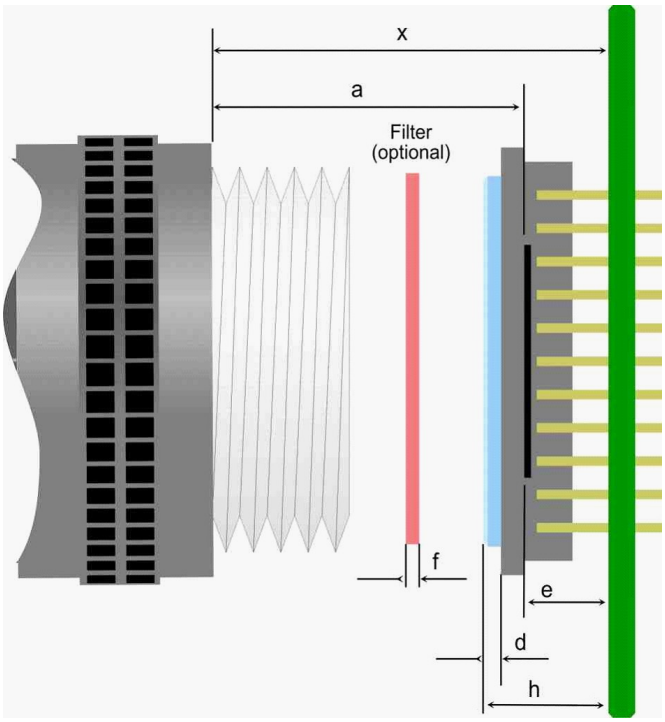
- Calculating the flange back distance
- Maximum immersion depth for lenses

5.4.5.1 Calculating the Flange Back Distance

To correctly determine the flange back distance of a DCx camera, you need to consider the distance between the lens flange and the active area of the sensor and, additionally, the type and thickness of any materials inserted into the optical path.

The "distance in air" between the threaded flange and the active area is 17.526 mm with C-mount lenses and 12.526 mm with CS-mount lenses.

This "mechanical distance" can change due to the material-specific refractive index of the inserted materials. The glass cover of the sensor and all filters inserted into the optical path (see Filter types table) must be taken into account in the calculation.



Calculating the flange back distance (schematic illustration)

Designation	Description
a	Distance from threaded flange to active sensor area (flange back distance) 17.526 mm*1 for C-mount 12.526 mm*1 for CS-mount
x	Distance from threaded flange to PCB
e	Distance from active sensor area to PCB
d	Thickness of the glass cover of the sensor
f	Filter thickness (optional)
n	Refractive index
h	Maximum sensor height above the PCB

*1 This distance describes the equivalent in air (see introduction above)

 © 2015 Thorlabs Scientific Imaging

Hint

You can use the following formula to calculate the mechanical flange back distance:

$$x = a + \frac{d \times (n_{glass} - 1)}{n_{glass}} + \frac{f \times (n_{filter} - 1)}{n_{filter}} + e$$

The tolerances for the position accuracy of DCx camera sensors are given in the Position accuracy chapter.

Calculating the flange back distance for DCx Cameras with C-mount

Camera model	Thickness sensor glass [mm]	Active sensor area to PCB [mm]	Flange to active sensor area without filter glass[mm]	Flange to active sensor area with filter glass [mm]	Sensor height above the PCB [mm]
Designation	d	e	x'	x	h
CMOS					
DCC1240x / DCC3240x	0.550	1.130	18.840	19.190	2.530
CCD					
DCU223x	0.750	2.810	20.590	20.930	4.780
DCU224x	0.750	2.810	20.590	20.930	4.830

Calculation example: UI-154x-xx with IR-cut filter

(a = 17.526 mm, d = 0.525 mm, nGlass = 1.50, f = 1mm, nFilter = 1.53; see Filter types table)

$$x = 17.526\,\text{mm} + \frac{0.525\,\text{mm} \times (1.50 - 1)}{1.50} + \frac{1.00\,\text{mm} \times (1.53 - 1)}{1.53} + 1.27\,\text{mm} = 19.32\,\text{mm}$$

Calculating the flange back distance for DCC1545M and DCC1645C cameras with CS-mount

Note

For these cameras with CS-mount, the flange back distance is only 12.526 mm.

Camera model	Thickness sensor glass [mm]	Active sensor area to PCB [mm]	Threaded flange to active sensor area	Flange to active sensor area without filter glass[mm]	Flange to active sensor area with filter glass [mm]	Sensor height above the PCB [mm]
Designation	d	e	a	x	x'	h
DCC1545M	0.525	1.270	13.23	15.95	17.50	2.480
DCC1645C	0.550	1.400	13.10	16.08	17.63	2.500

5.4.5.2 Maximum Immersion Depth for Lenses

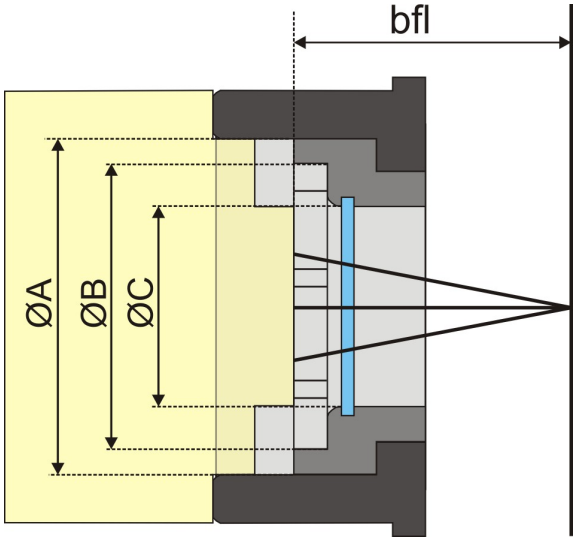
Some C-mount lenses reach deep into the camera flange. This may cause the lens to push against the back of the filter glass inside the camera or even make it impossible to screw in the lens.

The table below indicates the maximum possible immersion depth for each DCx camera model. The actual immersion depth of a lens is given in the relevant data sheet. As lens parts with a small diameter are allowed to reach deeper into the camera flange, the immersion depths are specified based on the diameter.

Beside the immersion depth also the back focal length has to be considered, that means the distance between the last lens and the sensor (named "bfl" in the image below). The back focal length can be calculated for C-mount with the following formula:

$$bfl = 17{,}526 - x$$

x stands for the maximum immersion depth (see table below).



Camera front (sectional view)

Note

Front Panel of the DCU223x / DCU224x housing

Models introduced before 2008 have a different front panel. On these models, the filter glass is held in the C-mount lens connector by two screws. This front panel version is indicated by (V2) in the table below.

Camera	Type	Thread depth	for diameter at lens end	resulting max. immersion depth (mm)		min. required back focal length (mm)	
			(Ø A, B, C in mm)	CMOS	CCD	CMOS	CCD
DCU223X DCU224X DCC1240 X	C-mount	5 mm	24.0	6.9	6.4	6.4	6.9
			20.0	9.4	8.9		
			14.1	11.1	10.6		
DCU223X (V2)	C-mount	5 mm	24.0	-	8.4	-	7.8

 © 2015 Thorlabs Scientific Imaging

DCU224X (V2)			17.1	-	9.2		
			14.1	-	9.7		
DCC1545 M DCC1645 C	CS-mount	4 mm	24.0	6.1	-	4.9	-
			14.6	7.6	-		
	C-mount with extension ring	4 mm*1	22.0*1	11.1	-		
			14.6*1	12.6	-		
DCC3240 x	C-mount	5 mm	24.0	7.2	-	6.8	-
			20.4	9.7	-		
			14.6	10.7	-		

*1 May vary depending on the inside diameter of the extension ring used

*2 Without IR cut filter

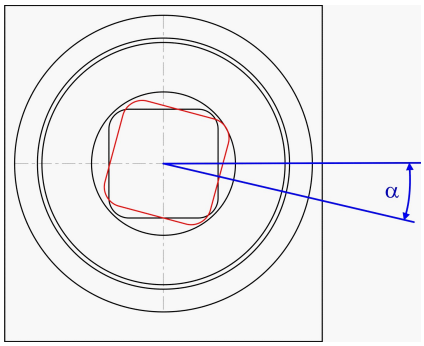
Note

The data given in the table include the following tolerances as a safety clearance:

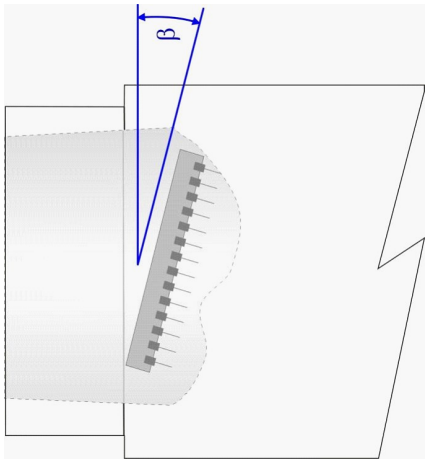
- Immersion depth: 0.2 mm
- Diameter: 0.2 mm

5.4.6 Position Accuracy of the Sensor

The following illustration shows the tolerance margins of the sensor position relative to the camera housing. A maximum error in all directions (rotation, translation) cannot occur simultaneously.



Position accuracy of the sensor (1)



Position accuracy of the sensor (2)

Position accuracy inside the camera housing, in each direction	±0.3	mm
Horizontal/vertical rotation (α)	±1.0	
Translational rotation (β)	±1.0	
Flange back distance	±0.05	mm

Note

C-mount lenses can also be subject to inaccuracies of the flange back distance. The tolerance usually is ±0.05 mm. In some cases, however, the inaccuracies of camera and lens might add up,

resulting in a total error > 0.05 mm.

5.4.7 Filter Glasses

- Filter types
- Mounting the filter
- Cleaning the filter glasses

5.4.7.1 Filter Types

Each DCx camera has a filter glass in the front flange to prevent the entry of dust particles. Color cameras by default use an IR cut filter, which is required to ensure correct color rendering. The default filter glass in monochrome cameras has no filter effect. Every camera model is available with different filter variants such as daylight cut filters (type DL).

The following table shows an overview of the different optical filters used in the DCx Cameras:

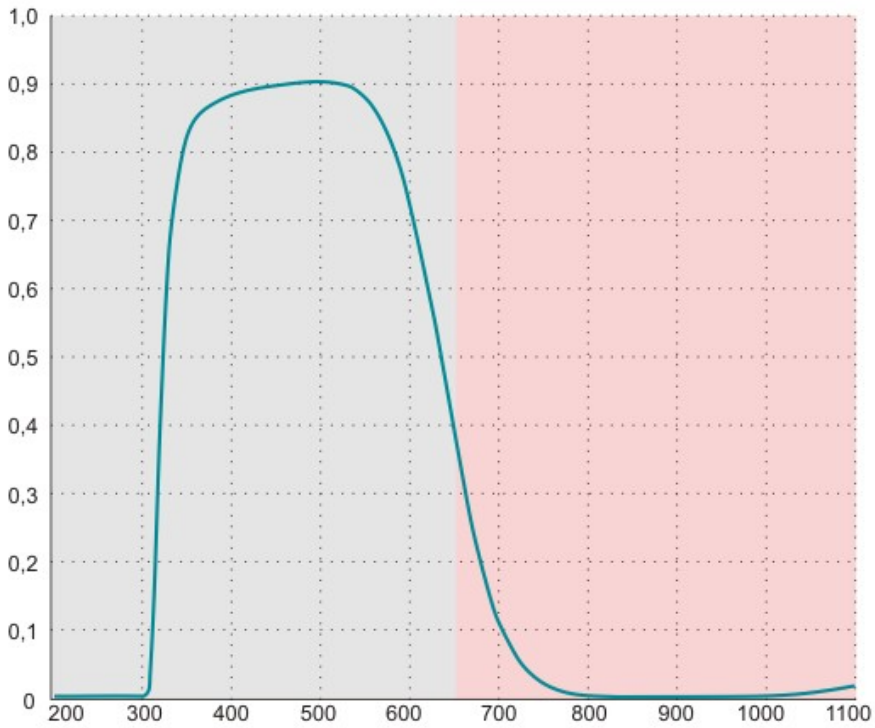
Filter type	Name	Refractive index (n_{Filter})	Glass type	Thickness (f)	Cut-off wavelength	Non-reflective
IR cut filter (old)	BG	1.53	BG40	1 mm	650 nm	-
IRcut filter (new)	HQ	1.53	D263	1 mm	650 nm	On one side
Daylight cut filter	DL	1.53	RG665	1 mm	695 nm	-
Glass	GL	1.53	D263	1 mm	380 nm	On both sides

 © 2015 Thorlabs Scientific Imaging

Notes

- All sensors have a D263 type cover glass. This glass is opaque to wavelengths below 330 nm.
- You can tell the filter type from visually by its coloration:
  - Reddish glass: HQ filter
  - Bluish glass: BG filter
  - Opaque glass: DL filter
  - Plain glass: GL filter
- New DCx color cameras use an IR cut filter of the type HQ by default. This filter offers an improved accuracy of the infrared content. HQ filters achieve a higher image brightness and better color rendering compared with the BG filter.
- uc480 drivers of version V3.24 and higher determine automatically which the IR filter is used in a camera. The corresponding color correction is selected automatically.

Infrared cut filter (type BG)

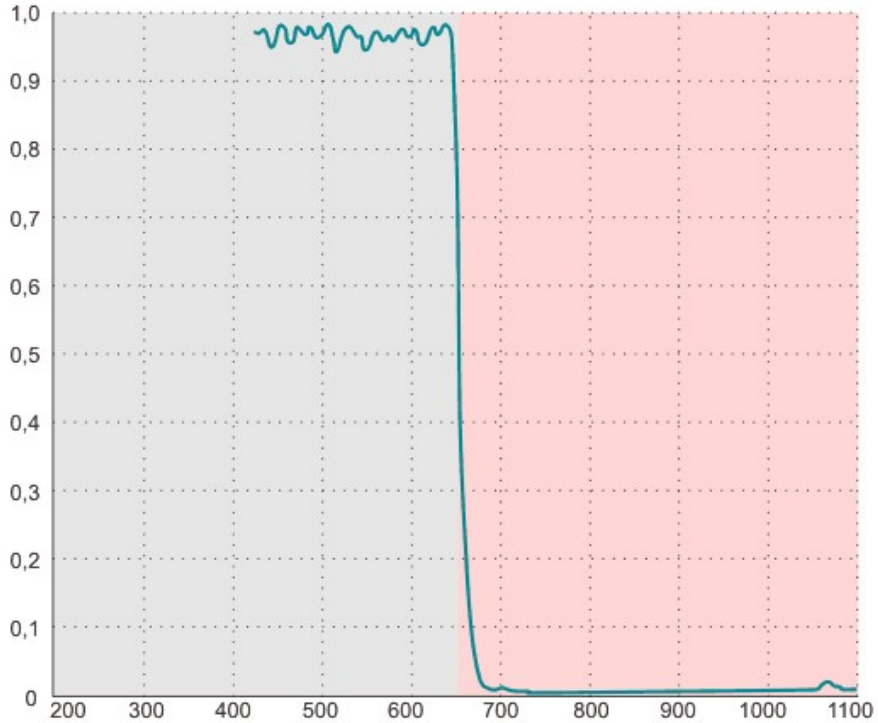


BG filter

Note

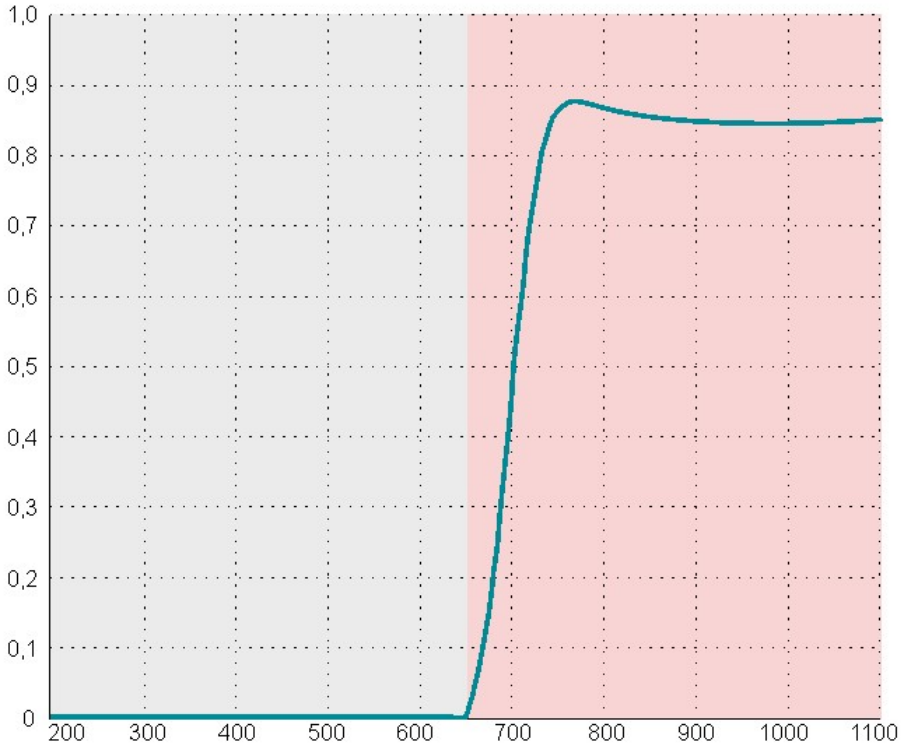
The colored part of above diagram just indicates the IR wavelength range in order to tell it from the visible.

Infrared cut filter (type HQ)



HQ filter

Daylight cut filter (type DL)



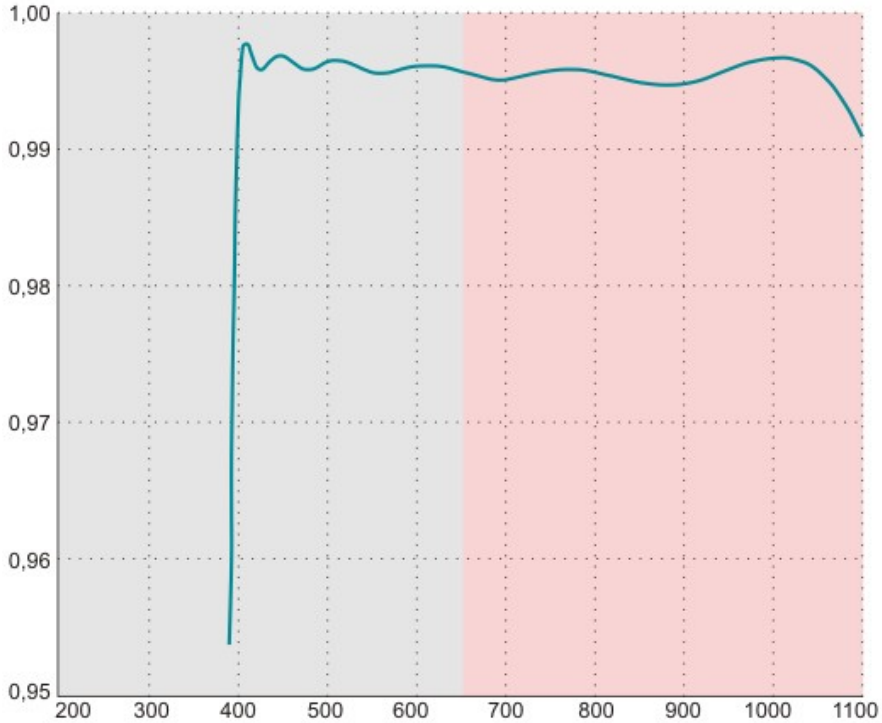
DL cut filter

Note

The colored part of above diagrams just indicates the IR wavelength range in order to tell it from the visible.

© 2015 Thorlabs Scientific Imaging

Plain glass filter (type GL)



Glass filter

Notes

- The colored part of above diagram just indicates the IR wavelength range in order to tell it from the visible.
- A different scale is used for the Y-axis of the glass filter curve, to show the curves between 400 nm and 1100 nm better.

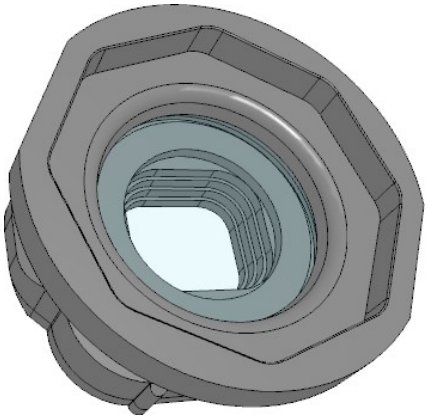
5.4.7.2 Mounting the Filter

Attention

It is recommended to have the filter changed under clean room conditions, otherwise dust might enter the sensor area and become visible in images.

When completely removing the adjustment ring and the filter glass, the rubber gasket should also be removed. Due to expansion of the rubber gasket during installation, it's difficult to reinstall then the filter glass.

The threaded ring presses the filter glass on a rubber gasket. A properly mounted threaded ring will seal off the sensor. The threaded ring is screwed into the adjusting ring from the front with a torque of 0.2 Nm.



Threaded ring

Notes

- A special tool is required for adjusting the threaded ring!
- Some DCx camera models have a different design with a separate filter glass that is secured by two screws.

© 2015 Thorlabs Scientific Imaging

5.4.7.3 Cleaning the Filter Glasses

When using the DCx camera with it's lens removed, the filter glass may become soiled from outside. This might be visible in captured images. In such case, the filter glass needs cleaning.

Note

It is strongly recommended to return the cameras to manufacturer for professional cleaning.The manufacturer is not liable for any damage resulting from cleaning the filter glasses. This even applies if the following instructions have been observed.

Instructions for cleaning DCx camera filter glasses

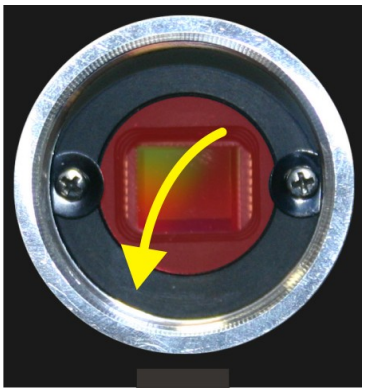
- The filter glasses can be cleaned only from outside. If remove the filter glasses, the sensor might become soiled. Thorlabs is not liable for any damage to the sensor resulting from removal of the filter glasses.
- First, remove dirt particles on the glass using compressed air. Do not use compressed air from compressors or spray cans since it often contains oil droplets or droplets of other liquids. For best results, use purified nitrogen from nitrogen bottles.
- Only use lint-free wipes or cotton-free swabs for cleaning. Never touch the filter glasses with bare fingers - it's mostly difficult to remove fingerprints completely!
- We recommend to use pure alcohol for cleaning. 100% isopropyl alcohol evaporates without leaving any residues. Only add small quantities of alcohol to the wipe. Never pour alcohol directly onto the camera.

Attention

Never use cleaning agents containing acetone for cleaning the filter glasses! Acetone may damage the filter glass coating and may deteriorate the optical quality of the glasses.

Cameras with fixed filter glass

Use a wipe to remove dust particles in a single sweep over the edge of the filter glass (see figure below).



Cleaning fixed DCx filter glasses

Cameras with replaceable filter glass

Use a wipe to remove dust particles in a circular sweep (see figure below).

Cleaning interchangeable DCx filter glasses

© 2015 Thorlabs Scientific Imaging

5.4.8 Ambient Conditions

Attention

- Avoid high air humidity levels and rapid temperature changes when using DCx Cameras.
- Temperatures below +4 °C (39 °F) combined with excessive relative air humidity levels can cause icing.
- At ambient temperatures above 45 °C (113 °F), the image quality could be reduced due to increased thermal noise. It is recommended to mount the camera to a heat-dissipating unit when high ambient temperatures prevail.

Note

The temperature values given above refer to the ambient temperature. The internal camera temperature is usually higher than the ambient temperature and must not exceed than 70 °C (158 ° F).

	Min.	Max.	
Ambient temperature	-5 23	50 122	°C °F
Storage temperature	-20 -4	60 140	°C °F
Relative humidity*1	20	80	%

*1 Non-condensing

Note

Non-condensing means that the relative air humidity must be below 100 %. Otherwise, moisture will form on the camera surface. If, for example, air has a relative humidity of 40 % at 35 °C (95 °F), the relative humidity will increase to over 100 % if the air cools down to 19.5 °C (67 °F); condensation begins to form.

Vibration and shock resistance

Vibration and shock resistance of the USB DCx Cameras were tested as specified in DIN EN 60068-2-6(1996-05), DIN EN 60068-2-27(1995-03) and DIN EN 60068-2-29(1995-03). The mechanical shock was at 80 g; the vibration testing was performed with sinusoidal vibration at a frequency between 30 Hz-500 Hz and an amplitude of 10 g.

5.5 Camera Interface

This section of the manual contains information on connecting the cameras and wiring IOs.

For information on a **camera's power consumption**, please refer to the Camera and sensor data section. This section contains information on all camera models sorted by sensor type.

- DCU223x, DCU224x, DCC1240x
- DCC3240x
- EEPROM Specification

5.5.1 DCU223x, DCU224x, DCC1240x

In this section the additional digital input / output of these cameras is described in detail:

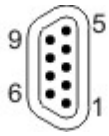
- I/O Connector - Pin Assignment
- Digital Input (Trigger) Circuit
- Digital Output (Flash) Circuit

5.5.1.1 I/O Connector - Pin Assignment

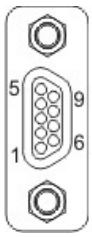
Attention

USB cables with non-standard connectors must be connected to the camera first and then to the PC. Otherwise the camera might not be recognized correctly.

9-pin micro D-Sub socket

Pin	Description	
1	Digital output (-)	 DCU22xX / DCC1240X Micro D-Sub socket male, camera rear view
2	Digital input (+)	
3	Shielding	
4	USB power supply (VCC) 5 V	
5	USB ground (GND)	
6	Digital output (+)	
7	Digital input (-)	
8	USB data (+)	
9	USB data (-)	

Pin assignment of the CAB-DCU-Tx cable for USB 2.0, trigger and flash

Pin	Description	Cable color	
1	Digital output (-)	green	 *DCU22xX / DCC1240X*

 © 2015 Thorlabs Scientific Imaging

2	Digital input (+)	white	*Micro D-Sub connector female, connecting side view*
6	Digital output (+)	yellow	
7	Digital input (-)	brown	

For a comprehensive list of all cables and connectors available for DCU22xX / DCC1240X cameras, please refer to the DCU22xX / DCC1240X Accessories section.

5.5.1.2 Digital Input (Trigger) Circuit

Digital input specifications

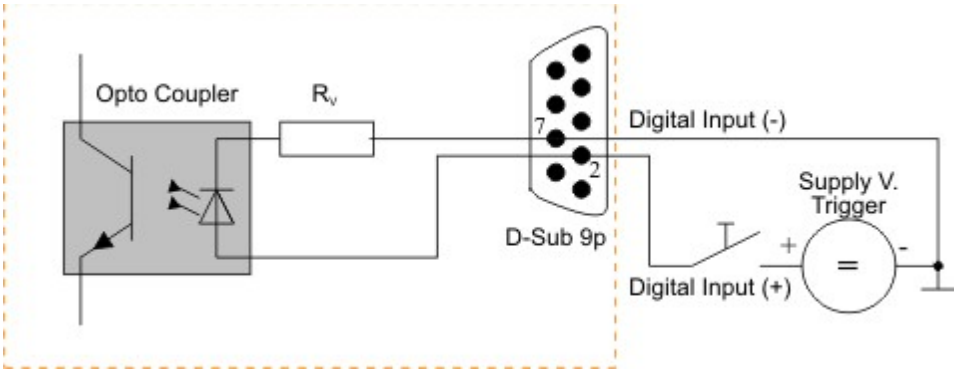
USB board revision *)	1.2		2.0 or higher		
	Min.	Max.	Min.	Max.	
Level low	0	2	0	2	V
Level high	9	24	5	24	V
Voltage range	0	30	0	30	V
Trigger pulse width (edge)	100	-	100	-	µs
Trigger edge steepness	35		35		V/ms
Breakdown voltage		50		50	V
Input current	10	-	10	-	mA

Note

*) For information on how to determine the USB board revision, please refer to the DCx Driver Compatibility chapter.

For interpreting the trigger signal, either the negative or the falling edge can be used. The digital input is galvanically isolated using an opto coupler to protect the camera and the PC against surges. Only DC voltages may be applied to the digital input.

Digital input wiring



Wiring of the trigger connector

© 2015 Thorlabs Scientific Imaging

5.5.1.3 Digital Output (Flash) Circuit

Digital output specifications

USB board revision *)	1.2	2.0 or higher	
	Max.	Max.	
Output current (short-time)	50	500	mA
Output current (permanent)	15	150	mA
Output voltage	30	30	V
Breakdown voltage	50	50	V
Collector power dissipation	100	125	mW

NOTE

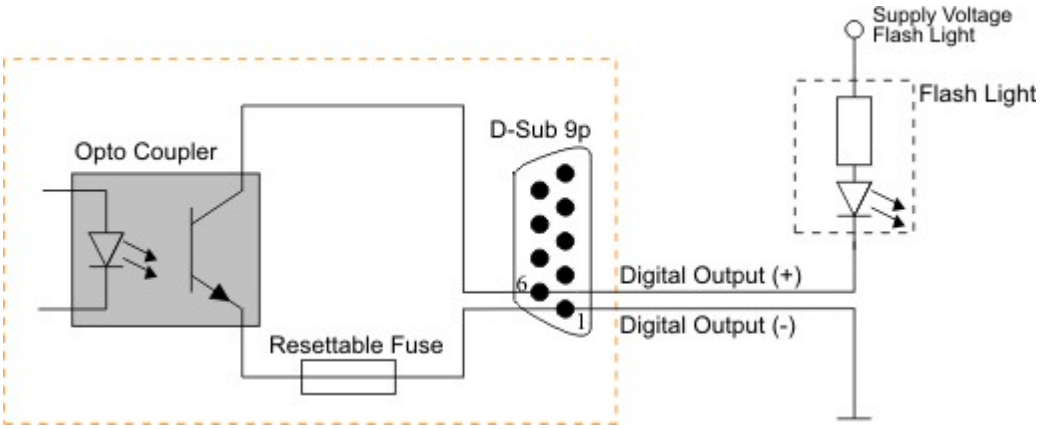
*) For information on how to determine the USB board revision, please refer to the DCx Driver Compatibility chapter.

The digital output is galvanically isolated using an opto coupler to protect the camera and the PC against surges. Only DC voltages may be applied to the digital input.

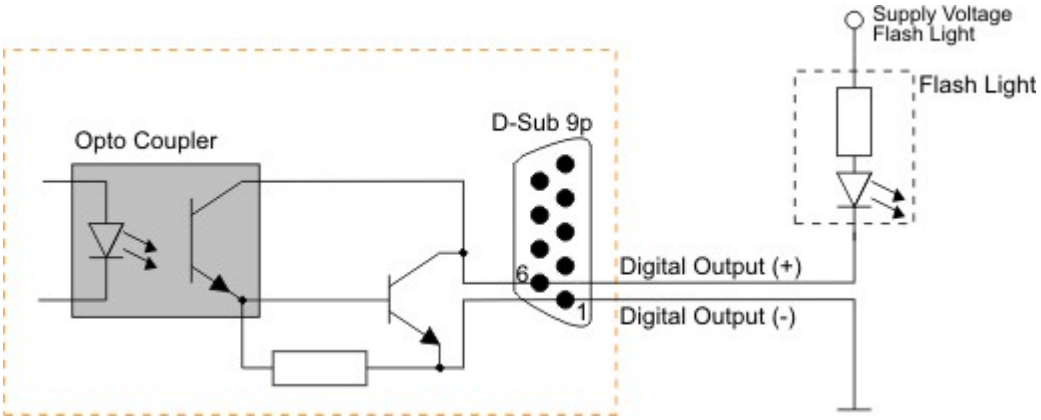
The output of the opto coupler can be used as an open collector or open emitter output. This means that the output signal can be connected to ground or to the supply voltage. The output signal is active if the collector-emitter switch is closed (software setting: Flash high active, see also the Camera Properties: Input/Output section).

Digital output wiring

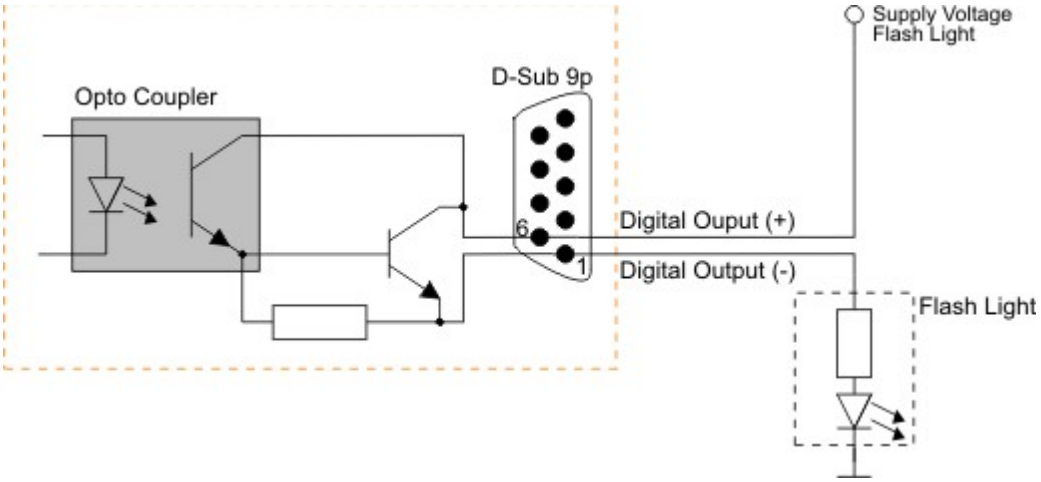
The following figures show examples of how the digital output is wired.



Wiring of the digital output as an open collector output (rev. 1.2)



Wiring of the digital output as an open collector output (rev. 2.0)



Wiring of the digital output as an open emitter output (rev. 2.0)

© 2015 Thorlabs Scientific Imaging

5.5.2 DCC3240x

- I/O connector Pin Assignment
- GPIO Interface
- Digital Input (Trigger) Circuit
- Digital Output (Flash) Circuit
- RS-232 Serial Interface

5.5.2.1 I/O Connector Pin Assignment

Attention

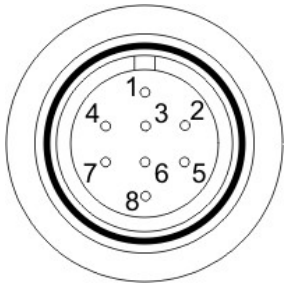
The General Purpose IO are not potential-free and have no protective circuit. Faulty wiring (overvoltage, undervoltage or inverting the wiring when used as serial interface) can result in a damage in the electronics.

During operation as serial interface only LVCMOS levels are allowed to the connector pins. To get a serial RS-232 compliant interface, an external level shifter (LVCMOS/RS-232) is required.

Applying RS-323 levels directly to the pins as well as mixing up the signals RxD and TxD can destroy the camera electronics!

Pin assignment of the 8-pin Hirose connector HR25 for trigger, flash and GPIO

Pin	Description	Cable color
1	Ground (GND)	gray
2	Flash output, opto-decoupled (-)	green
3	GPIO 1, 3.3 V LVCMOS	blue
4	Trigger input, opto-decoupled (-)	brown
5	Flash output, opto-decoupled (+)	yellow
6	GPIO 2, 3.3 V LVCMOS	red
7	Trigger input, opto-decoupled (+)	white
8	Output supply voltage, 5 V (100 mA)	pink



Hirose connector male, camera rear view

For a comprehensive list of all cables and connectors available for DCC3240X cameras, please refer to the DCC3240X Accessories section.

5.5.2.2 GPIO Interface

GPIO specifications

The two GPIOs (General Purpose I/O) can be used as inputs or outputs. This selection is made by software using the corresponding SDK API functions. Please observe the following criteria:

- Input: 3.3 V LVCMOS, max. input voltage 4.0 V
- Output: 3.3 V LVCMOS, max. 10 mA

Attention

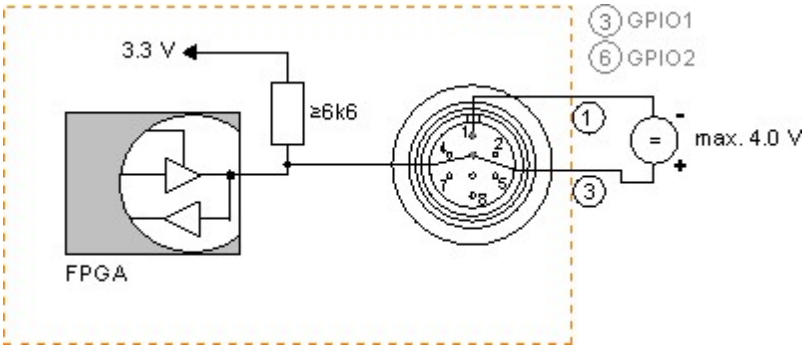
The General Purpose IO are not potential-free and have no protective circuit. Faulty wiring (overvoltage, undervoltage or inverting the wiring when used as serial interface) can result in a damage in the electronics.

During operation as serial interface only LVCMOS levels are allowed to the connector pins. To get a serial RS-232 compliant interface, an external level shifter (LVCMOS/RS-232) is required.

Applying RS-323 levels directly to the pins as well as mixing up the signals RxD and TxD can destroy the camera electronics!

GPIO wiring as input

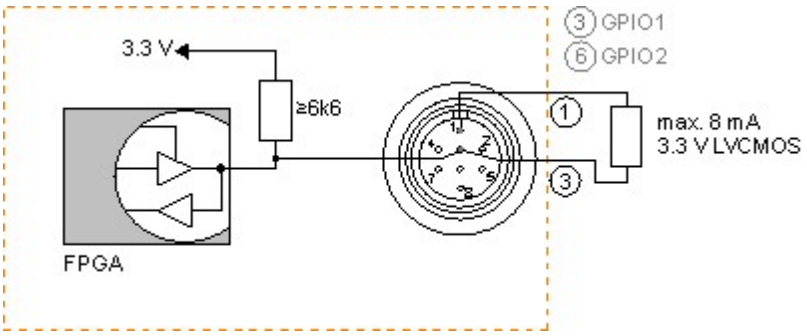
The following figures illustrate GPIO wiring examples.



GPIO input

	Min.	Max.	
Signal level "Low"	0	0.8	V
Signal level "High"	2.0	4.0	V

GPIO wiring as output



GPIO output

	Min.	Max.	
Signal level "Low"	0	0.8	V

© 2015 Thorlabs Scientific Imaging

Signal level "High"	2.4	3.3	V
Output current	0	8.0	mA

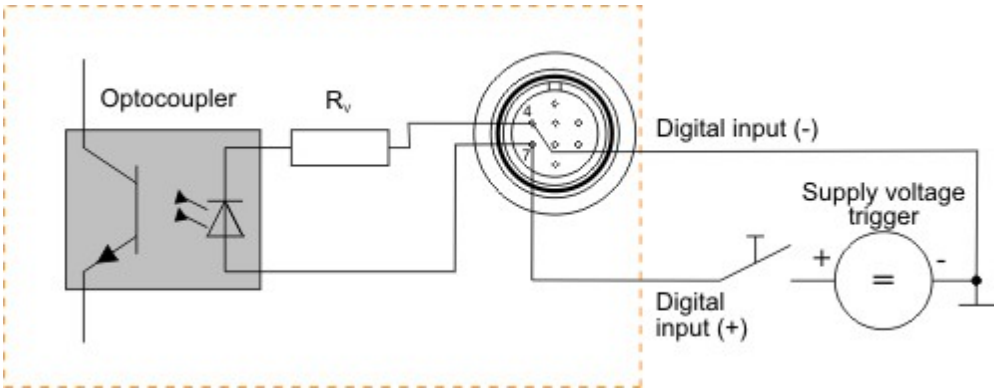
5.5.2.3 Digital Input (Trigger) Circuit

Digital input specifications

	Min.	Max.	
Level low	0	1	V
Level high	5	24	V
Voltage range	0	-	V
Trigger pulse width (edge)	10	-	µs
Trigger edge steepness	35	-	V/ms
Breakdown voltage	-	50	V
Input current	10	-	mA

For interpreting the trigger signal, either the negative or the falling edge can be used. The digital input is galvanically isolated using an optocoupler to protect the camera and the PC against surges. Only DC voltages may be applied to the digital input.

Digital input wiring



Wiring of the trigger connector

© 2015 Thorlabs Scientific Imaging

5.5.2.4 Digital Output (Flash) Circuit

Digital output specifications

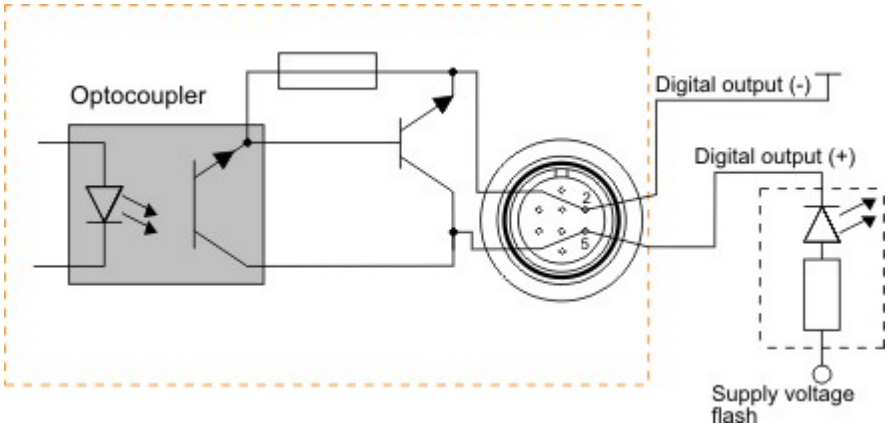
	Max.	
Output current (short-time)	500	mA
Output current (permanent)	150	mA
Output voltage	30	V
Breakdown voltage	50	V
Collector power dissipation	125	mW

The digital output is galvanically isolated using an optocoupler to protect the camera and the PC against surges. Only DC voltages may be applied to the digital output.

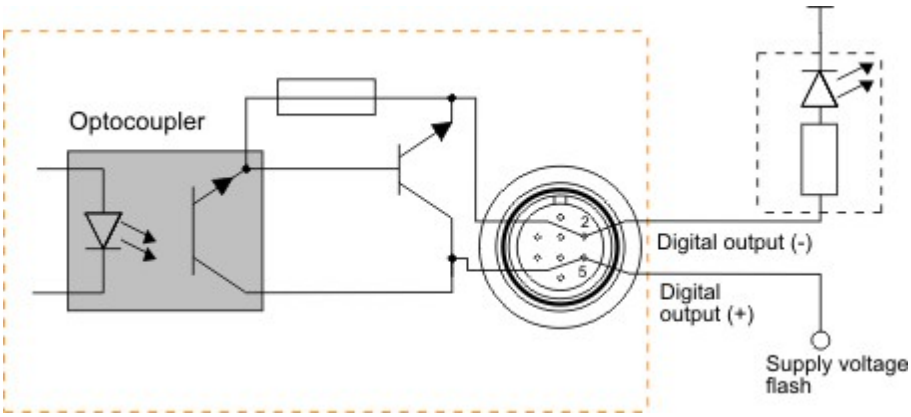
The output of the optocoupler can be used as an open collector or open emitter output. This means that the output signal can be connected to ground or to the supply voltage. The output signal is active if the collector-emitter switch is closed (software setting: Flash high active, see also the Camera properties: Input/output section).

Digital output wiring

The following figures show examples of how the digital output is wired.



Wiring of the digital output as an open collector output



Wiring of the digital output as an open emitter output

5.5.2.5 RS-232 Serial Interface

Attention

The General Purpose IO are not potential-free and have no protective circuit. Faulty wiring (overvoltage, undervoltage or inverting the wiring when used as serial interface) can result in a damage in the electronics.

During operation as serial interface only LVCMOS levels are allowed to the connector pins. To get a serial RS-232 compliant interface, an external level shifter (LVCMOS/RS-232) is required.

Applying RS-323 levels directly to the pins as well as mixing up the signals RxD and TxD can destroy the camera electronics!

Serial interface specification

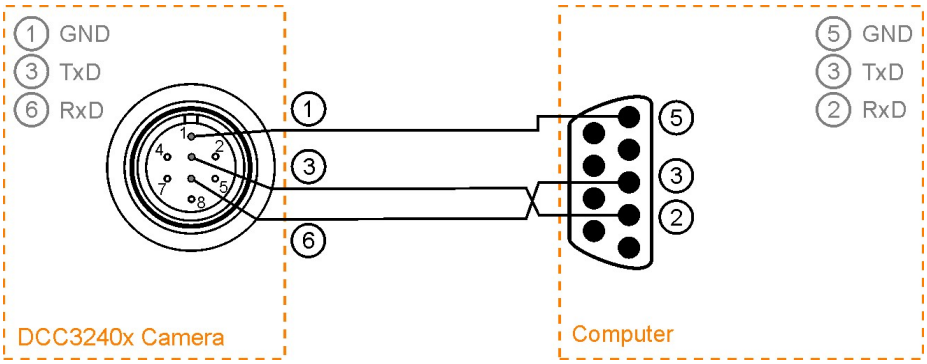
		Min.	Max.	
Minimum output voltage				
	Signal level "Low"	0	0.8	V
	Signal level "High"	2.0	4.0	V
Maximum input voltage		Min.	Max.	
	Signal level "Low"	0	0.8	V
	Signal level "High"	2.0	4.0	V
Supported Baud rates	9.600 19.200 38.400 57.600 115.200			baud
Transmission mode	Full duplex, 8N1			
Data bits	8			
Stop bits	1			
Parity	None			

Note

With the 8N1 mode, the maximum payload data rate achievable is 80% of the selected baud rate.

Serial interface wiring (UART)

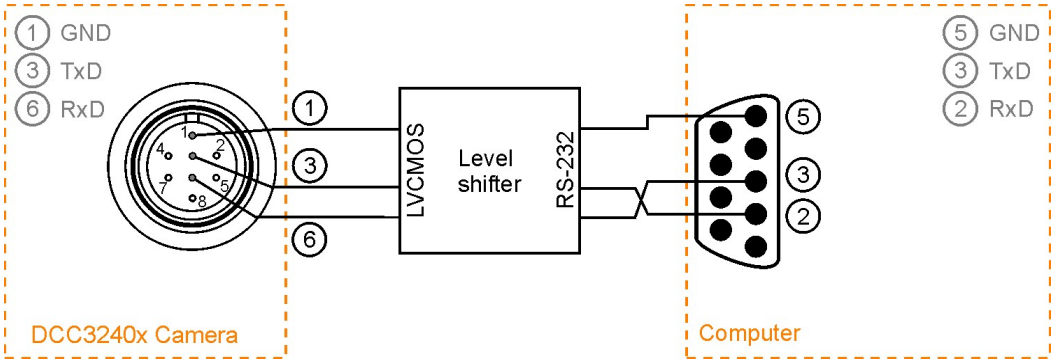
The following figure shows the wiring of the serial interface with GPIO 1 as camera-side output (TxD) and GPIO 2 as camera-side input (RxD). The GPIO must be configured accordingly (see `is_IO()`).



Serial interface connector (UART)

© 2015 Thorlabs Scientific Imaging

Serial interface wiring (RS-232)



Serial interface connector (RS-232)

5.5.3 Camera EEPROM Specification

DCx Cameras have an EEPROM memory where the camera manufacturer, type, and serial number are stored. A 64 byte memory space can be used freely by the user.

EEPROM Specifications	
Data retention	10 years
Read/write cycles	100,000
Size of user data space	64 bytes

5.6 Accessories for DCx cameras

Lenses

Thorlabs also supplies a wide variety of lenses and objectives from leading manufacturers. Please contact Thorlabs for technical support and a detailed quote tailored to your needs.

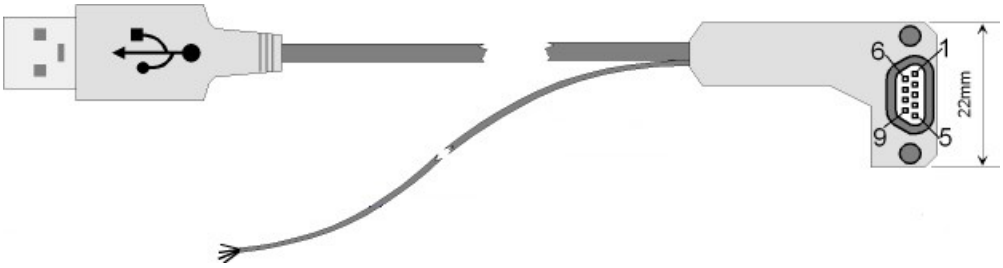
USB cables

All Thorlabs DCx cameras are shipped with a 1.5m USB2.0, A to Mini B, cable.

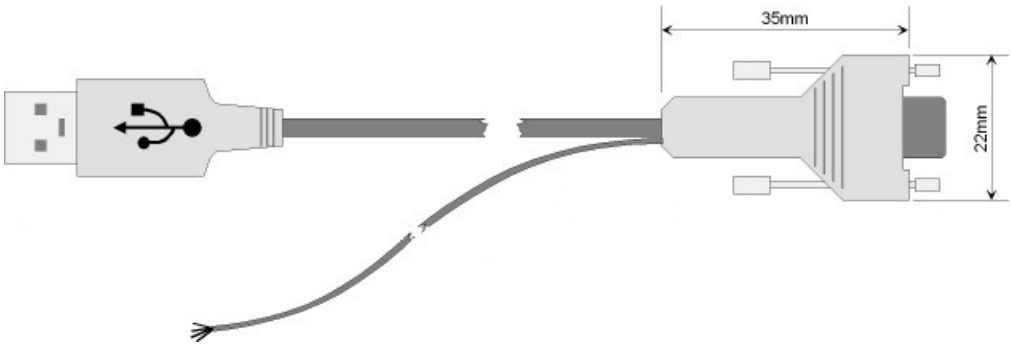
5.6.1 Accessories for DCU22xX / DCC1240X

For information on the pin assignment of the cables and connectors see chapter Pin Assignment I/O Connector.

USB Cables with Cables for digital I/Os



CAB-DCU-T1 (angled Micro D SUB camera connector; trigger in / out)



CAB-DCU-T2 (straight Micro D SUB camera connector, trigger in only)

Type	Length	Cable Type	Function	Connector Camera Side	Connector PC Side
CAB-DCU-T1	3 m	USB cable, AWG 28, single shielded, additional cable for digital I/Os, 4-wire, open wires	USB & Trigger in / out	Micro D-Sub for screw-mounting, angled	USB 2.0 Type A
CAB-DCU-T2	3 m		USB & Trigger in	Micro D-Sub for screw-mounting, straight	

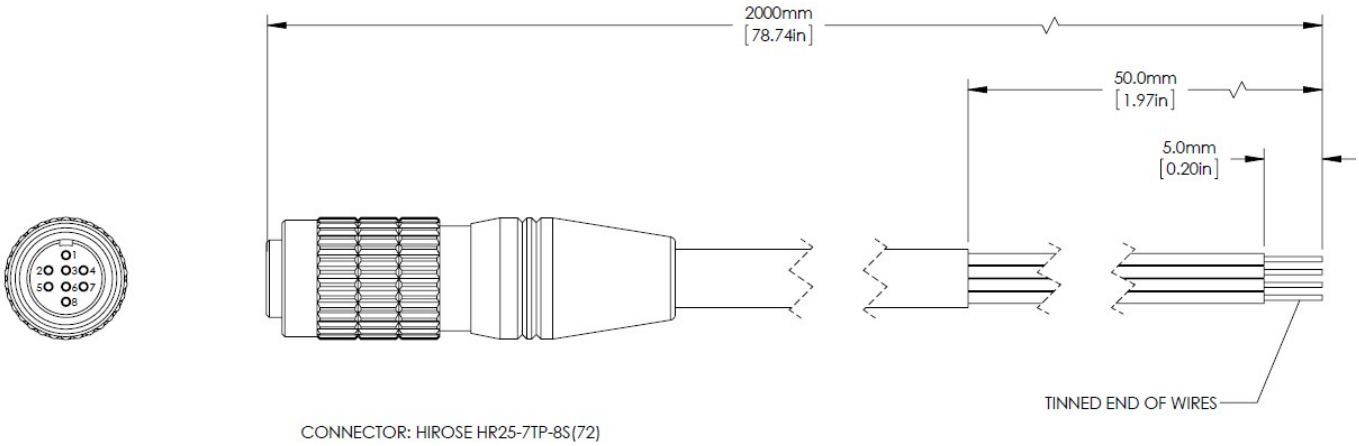
 © 2015 Thorlabs Scientific Imaging

5.6.2 Accessories for DCC1x45X

For USB cables and accessories see also Accessories for all DCx cameras.

DCC1545M and DCC1645C CS mount cameras are delivered with both CS / C objective mount adapter and CS / SM1 1" optics adapters.

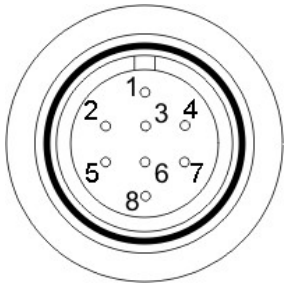
5.6.3 Accessories for DCC3240x



Type	Length	Cable Type	Function	Connector Camera Side
CAB-DCU-T3	2 m	Shielded high-flexible control cable 8 x0.1mm, Ø 4.9mm	GPIO digital in (trigger) digital out (flash)	Hirose 8 pin (HR25)

Pin assignment of the 8-pin Hirose connector HR25 for trigger, flash and GPIO

Pin	Description	Cable color
1	Ground (GND)	gray
2	Flash output, opto-decoupled (-)	green
3	GPIO 1, 3.3 V LVCMOS	blue
4	Trigger input, opto-decoupled (-)	brown
5	Flash output, opto-decoupled (+)	yellow
6	GPIO 2, 3.3 V LVCMOS	red
7	Trigger input, opto-decoupled (+)	white
8	Output supply voltage, 5 V (100 mA)	pink



Hirose connector female, cable tip view

6 Appendix

- Troubleshooting/FAQ
- Status LED on USB DCx Cameras
- Color and memory formats
- uc480 parameter file (ini file)
- History of API functions
- Certifications and Compliances
- Thorlabs 'End of Life' Policy (WEEE)
- Exclusion of Liability and Copyright
- Thorlabs Worldwide Contacts

6.1 Troubleshooting/FAQ

Installation and connection

- Installation of the Thorlabs camera software fails.

You need administrator privileges to install the software. Operating the cameras, however, does not require administrator privileges.

- The camera is connected to the PC, but cannot be opened in Thorcam

Check the LED on the camera:

- LED is **red:** Camera detection failed. Check whether the TSI driver software has been installed. Disconnect and reconnect the camera to the USB cable. The camera should then be correctly recognized.
  If the camera is still not listed in the uc480 Camera Manager, open the Windows Device Manager to check whether the camera has been correctly recognized. If recognition was successful, you will find an entry in the format “uc480.....cameras” under “Universal Serial Bus Controllers.” A question mark or exclamation mark before the entry indicates that camera was not correctly recognized. You can remove the entry using the shortcut menu (right-click). Disconnect and reconnect the camera. The Found New Hardware Wizard will detect it as a new device and install the appropriate drivers. The camera should then be correctly recognized.
- LED is **green:** The camera is fully operational. Check whether the camera has been opened in a different application.
- LED is **off:** No power supply to the camera. Check the cable, the connectors and, if applicable, the power supply to the hubs. In case of a DCU22xX or DC1240X camera, check whether any pins of the micro D-sub connector have been bent.
- LED **flashes:** A fault has occurred in the camera hardware. Please contact the Thorlabs.

USB DCx camera operation

- The camera can be opened in the software, but captures images sporadically or not at all.

Check for dropped frames in the settings window of the ThorCam software. If significant percentage of dropped frames are reported, the camera speed settings are too high for the system you are using. Check the following:

- Use only USB 2.0 (USB 3 in case of DCC3240x) certified cables and hubs.
- Do not use any passive extension cables.
- Do not connect the camera to the USB ports on the front of the PC, but to the ones directly on the main board. You will find those USB ports at the back of the PC.

In addition, check the following camera settings in the software:

 © 2015 Thorlabs Scientific Imaging

- Pixel clock frequency: Reduce the pixel clock if data transfer errors occur. When you are operating more than one USB camera on one port, the pixel clock of all the cameras added together should not exceed about 40 MHz.

Is it possible to operate an older USB camera with Windows 7?

Look for the serial number to see if your camera can be operated with Windows 7:

- The support for Windows 7 was introduced with driver version 3.50. This driver can be used with cameras from serial number 400 26 27000 on.

6.1.1 PCs with Energy Saving CPU Technology

This application note is related to all DCx USB cameras connected to PC systems using current CPU models that implement modern energy saving technologies.

Symptoms:

- Low USB bandwidth provided by the PC system

- TransferFailed errors occurring even at moderate pixel clock settings

- Camera operates at low speed only

Summary:

Current CPUs with modern energy saving features can cause bandwidth limitations on USB. The only available approach to this issue is to disable CPU sleep states. Unfortunately this is not possible for all systems.

Detailed explanation:

Modern CPUs like Intel i5 & i7 and others make use of advanced energy saving technologies ensuring a low power consumption and long battery life for mobile

devices. Additionally those CPU implement features for increasing the performance of single cores if there is enough thermal headroom available when other cores have little load.

A basic idea to achieve this is to put a CPU core to sleep while there is nothing to do for it. Various different activity states of CPU cores are available in modern CPUs. These CPU states are referred to as “C-states”. C0 is the working state of a core.

Increasing numbers refer to less activity and longer wake up times. Current CPU fall down to variations of the C3 state which are referred to as “Sleep”, “Deep Sleep” and similar.

Unfortunately negative effects of the sleep states have shown up. It is observed that the available bandwidth of PC busses drops significantly when part of the CPU enters these states.

The operation of DCx USB cameras is affected by the sleep states because they reduce the speed of the USB system. The available bandwidth on the USB may drop down to around 30% of the maximum bandwidth when the CPU, or one of its cores, enters sleeping states.

One would expect that a CPU core will not fall into a sleep state while it is obviously needed for the operation of the USB. But obviously USB data transfers do not prevent the CPU from falling to sleep. If the code execution load of a CPU core is low enough it will fall asleep and immediately reduce the USB bus speed.

For operation at high frame rates DCx cameras require an adequate USB bandwidth which might not be available when CPU cores are in sleep states.

Advice:

If you seem to be running into this low bandwidth issue please check and try the following. These first hints are general recommendations for issues with the USB

data transfer. You can check the USB performance with the “Optimum” pixel clock settings checkbox in uc480 Demo software. A good USB system should be able to reach a pixel clock

setting near the maximum value.

- Please remove other USB devices from the system (USB keyboard and mouse are fine). Run tests with only one camera connected at once.
- Make sure using a USB port directly on the mainboard. Front panel or other ports are connected to the mainboard with poor cabling quality frequently.
- Make sure to use USB2.0 certified cables to connect the camera.
- If you are using USB hubs or extensions: Run a test without these devices, connect the camera directly to the PC.
- Disable other equipment that is connected via USB. For example WLAN and Bluetooth adapters might use USB to connect.
- If you are using a mobile PC: run it on mains power, not battery.
- Check your energy saving options in the operating system. Disable energy saving features and set the available features to “full performance” or similarly named options.

If you checked the above and still observe low USB performance you might be experiencing the issue with CPU sleep states.

6.2 Status LED on USB DCx Cameras

DCU223x, DCU224x and DCC1240x

The LED on the rear side of the USB DCx camera indicates whether

- the DCx camera is powered on – LED lights up red (only USB board rev. 2.0 or higher).
- the uc480 driver has been loaded and the camera is operational – LED lights up green
- an error has occurred – green LED flashes:
- 2x flash: unknown sensor, please contact our Thorlabs team.

If the LED does not light up green, please check the following:

- Has the camera been connected correctly?
- Have the driver and the camera been installed properly in the uc480_Camera_Manager on the host PC?
- Does the host PC meet all system requirements?

Revision 1.2 (green Status LED)

Revision 2.0 (red/green LED)

 © 2015 Thorlabs Scientific Imaging

DCC3240x

The LED on the DCC3240x flashes 2x green if the camera is connected to a USB 2.0 port. If the camera is connected to a USB 3.0 port the LED flashes 3x green.

6.3 Color and Memory Formats

Attention

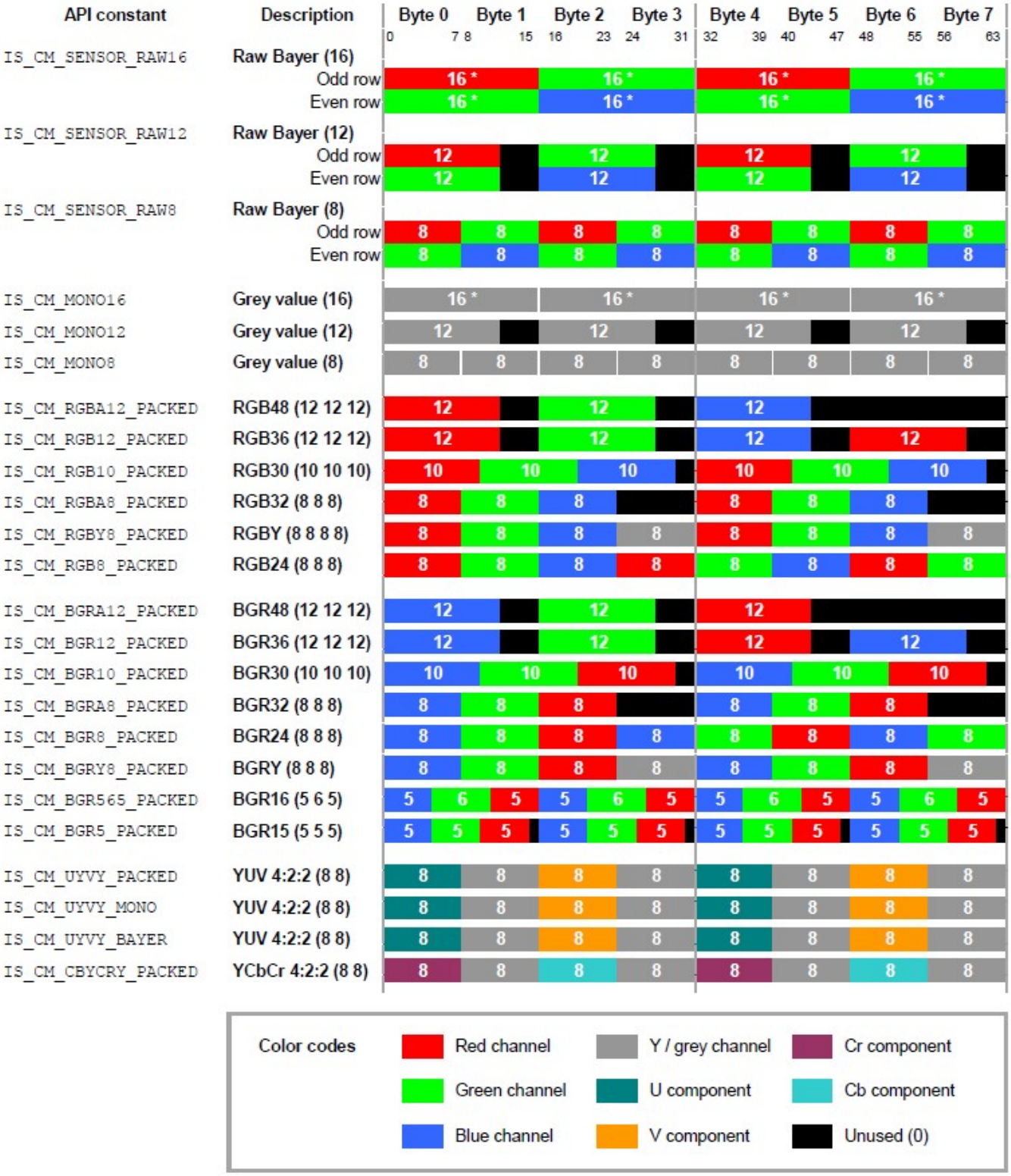
Obsolete parameters

The following parameters for color formats are obsolete and should no longer be used (see also `is_SetColorMode()`):

- `IS_SET_CM_RGB32`
- `IS_SET_CM_RGB24`
- `IS_SET_CM_RGB16`
- `IS_SET_CM_RGB15`
- `IS_CM_UYVY_MONO_PACKED`
- `IS_CM_UYVY_BAYER_PACKED`
- `IS_CM_BAYER_RG8 (identical to IS_CM_SENSOR_RAW8)`
- `IS_CM_BAYER_RG12 (identical to IS_CM_SENSOR_RAW12)`
- `IS_CM_BAYER_RG16 (identical to IS_CM_SENSOR_RAW16)`
- `IS_CM_BGR555_PACKED (has been renamed to IS_CM_BGR5_PACKED)`

Each color format supported by the DCx camera defines a different memory format. The following table shows the byte arrangement in memory:

 © 2015 Thorlabs Scientific Imaging



Colour and memory formats

Note

An asterisk (*) identifies formats which are filled starting with the most significant bit (MSB) but which may have less than the indicated number of payload bits, depending on the camera model.

For the RGB16 and RGB15 data formats, the MSBs of the internal 8-bit R, G and B colors are used.

The first pixel in the first line with the index (0,0) is always a red pixel at color cameras.

The list above does not contain the `IS_CM_RGB8_PLANAR` color format. In planar RGB the image is saved as 8 bit RGB. The channels red, green, and blue are stored separately, i.e. first all red information, then all green information and at last all blue information are saved.

© 2015 Thorlabs Scientific Imaging

6.4 uc480 Parameter File (ini file)

Using the `is_ParameterSet()` function, you can save the currently set DCx camera parameters to a file in the ini format (*.ini) or load an ini file.

Attention

Only camera-specific ini files can be loaded.

The ini file you want to load has to match the paired camera model.

When loading an ini file, make sure that the image size (AOI) and color depth parameters in the ini file match those in the allocated memory. Otherwise, display errors may occur.

uc480 parameter files can also be created and edited manually. The following table shows the structure of the parameter file. The entries in square brackets [] indicate sections. If a section does not exist in the ini file, the corresponding camera parameters will not be modified when you load the file.

Hint

You can use wildcards in the ini file:

If you specify * as first character, then the highlighted characters will be interpreted in the camera name only, e.g. ***DCU**224**C**. Thus this string applies for DCU223C and DCU224C.

Structure of a uc480 parameter file

Parameter	Description	Value range	Example
[Versions]			
`uc480.dll`	File version of the uc480 API	-	`4.00.0000`
`uc480_eth.sys`	(Not applicable to DCx Cameras)	-	`4.00.0000`
`uc480_usb.sys`	File version of the USB uc480 driver	-	`4.00.0000`
`uc480_boot.sys`	File version of the USB uc480 boot loader	-	`4.00.0000`
[Sensor]			
`Sensor`	Full name of the camera model	-	`DCC3240C`
[Image size]	Image size settings		
`Start X`	Start point (X coordinate) in AOI mode	0...(max. width[*1]-`Width`)	`100`
`Start Y`	Start point (Y coordinate) in AOI mode	0...(max. height[*1]-`Height`)	`100`
`Start X absolute`	Activate absolute AOI positioning in the memory (see `is_AOI()`)	0, 1	`1`
`Start Y absolute`	Activate absolute AOI positioning in the memory (see `is_AOI()`)	0, 1	`1`
`Width`	Width of the AOI	Sensor-dependent[*1]	`2460`

Parameter	Description	Value range	Example
`Height`	Height of the AOI	Sensor-dependent*1	`1820`
`Binning`	Activate binning mode and select factor	Sensor-dependent*2	`0`
`Subsampling`	Activate subsampling mode and select factor	Sensor-dependent*2	`0`
[Scaler]	The internal image scaling is only supported by sensors of the DCC1240x / DCC3240x camera series.		
`Mode`	Enable/disable scaling	0 = Scaling off 1 = Scaling on	`0`
`Factor`	Scaling factor		
[Multi AOI]			
`Enabled`	Activate/deactivate multi AOI	0 = Multi AOI off 1 = Multi AOI on	`0`
`Mode`	Mode of multi AOI Currently only `IS_AOI_MULTI_MODE_AXES` is supported		
`x1…x4, y1…y4`	Axis for multi AOI mode		
[Shutter]			
`Mode`	Shutter mode	Sensor-dependent (only supported by DCC1240x / DCC3240x models) 1 = Rolling shutter 2 = Global shutter 4 = Fast linescan 64 = Rolling shutter with global start 128 = Global shutter (alternative timing)	`1`
`Linscan number`	Line which is used in the linescan mode. The maximum possible line number depends on the height of the selected AOI.	Sensor-dependent (only supported by DCC1240x / DCC3240x models)	`512`
[Log Mode]			
`Mode`	Log mode (only supported by DCC1240x / DCC3240x models)	0 = Factory-default with anti-blooming 1 = Off (no anti-blooming) 2 = Manual Log mode	`0`
`Manual value`	Log mode value	Only in combination with manual Log mode	
`Manual gain`	Log mode gain		
[Timing]	Timing parameter settings		
`Pixelclock`	Current pixel clock of the camera	Sensor-dependent*1	`103`
`Framerate`	Current frame rate	Depends on `Pixelclock`	`15.104458`

© 2015 Thorlabs Scientific Imaging

Parameter	Description	Value range	Example
		and image geometry	
Exposure	Current exposure time	Depends on Framerate	0.334059
Long exposure	Activates long exposure If the long exposure is active, then the range of exposure changes.	Not supported by DCx cameras	0
[Selected Converter]	Sets the type of Bayer conversion for the specified color format when using color cameras (see is_SetColorConverter()). For a description of all color formats, see the Color and memory formats section.		
IS_SET_CM_RGB32	Color format	0, 1, 2, 4	2
IS_SET_CM_RGB24	Color format	0, 1, 2, 4	2
IS_SET_CM_RGB16	Color format	0, 1, 2, 4	2
IS_SET_CM_RGB15	Color format	0, 1, 2, 4	2
IS_SET_CM_Y8	Color format	0, 1, 2, 4	2
IS_SET_CM_RGB8	Color format	0, 1, 2, 4	2
IS_SET_CM_BAYER	Color format	0, 1, 2, 4	8
IS_SET_CM_UYVY	Color format	0, 1, 2, 4	2
IS_SET_CM_UYVY_MONO	Color format	0, 1, 2, 4	2
IS_SET_CM_UYVY_BAYER	Color format	0, 1, 2, 4	2
IS_CM_CBYCRY_PACKED	Color format	0, 1, 2, 4	8
IS_SET_CM_RGBY	Color format	0, 1, 2, 4	8
IS_SET_CM_RGB30	Color format	0, 1, 2, 4	8
IS_SET_CM_Y12	Color format	0, 1, 2, 4	8
IS_SET_CM_BAYER12	Color format	0, 1, 2, 4	8
IS_SET_CM_Y16	Color format	0, 1, 2, 4	8
IS_SET_CM_BAYER16	Color format	0, 1, 2, 4	8
IS_CM_RGBA8_PACKED	Color format	0, 1, 2, 4	2
IS_CM_RGB8_PACKED	Color format	0, 1, 2, 4	2
IS_CM_RGBY8_PACKED	Color format	0, 1, 2, 4	8
IS_CM_RGB10V2_PACKED	Color format	0, 1, 2, 4	8
[Parameters]	Additional image parameter settings		
Colormode	Sets the current color mode	see Color and memory formats	11
Brightness	Software correction of image brightness*3	0...255	100
Contrast	Software correction of image contrast*3	0...511	215
Gamma	Software correction of the gamma value	0.01...10.0	1.000000
Hardware Gamma	Sensor-based hardware correction of the gamma	0, 1	0

Parameter	Description	Value range	Example
	value		
Blacklevel Mode	Mode for black level correction of the sensor	0, 1, 32*2	1
Blacklevel Offset	Manual offset for black level correction of the sensor	0...255	0
Hotpixel Mode	Mode for hot pixel correction	0, 1, 2, 4*2	2
Hotpixel Threshold	Not used	-	0
Sensor Hotpixel	Activates the sensor-internal hot pixel correction	Sensor-dependent*1 1 = on 0 = off	0
GlobalShutter	Enables the Global Start shutter of the sensor	0, 1 Not supported by DCx models, they return "7" (not supported).	0
[Gain]	Sets the sensor gain control for image brightness		
Master	Master gain	0...100	0
Red	Red gain	0...100	6
Green	Green gain	0...100	0
Blue	Blue gain	0...100	6
GainBoost	Activate gain boost	0, 1	0
[Processing]	Parameters for image pre-processing in the driver		
EdgeEnhancementFactor	Enable edge enhancement	0...2	0
RopEffect	Image geometry change (Rop = raster operation), e.g. mirroring	0, 8, 16, 32, 64*2	0
Whitebalance	Enable software white balance	0, 1, 2, 4*2	0
Whitebalance Red	Red factor for software white balance	double value	1.000000
Whitebalance Green	Green factor for software white balance	double value	1.000000
Whitebalance Blue	Blue factor for software white balance	double value	1.000000
Color correction	Enable color correction	0, 1, 2, 4, 80*2	1
Color_correction_factor	Set the color correction factor	0.0...1.0	1.000000
Color_correction_satU	Saturation-U (see also is_SetSaturation())	0…200 100 = Saturation 1.0 200 = Saturation 2.0	0
Color_correction_satV	Saturation-V (see also is_SetSaturation())	0…200 100 = Saturation 1.0 200 = Saturation 2.0	0

© 2015 Thorlabs Scientific Imaging

Parameter	Description	Value range	Example
Bayer Conversion	Sets the size of the Bayer conversion mask for the current color format when using color cameras	1, 2*2	1
[Auto features]	Sets the parameters for automatic image control		
Auto Framerate control	Enable frame rate control	0, 1	0
Brightness exposure control	Enable exposure time control	0, 1	0
Brightness gain control	Enable sensor gain control	0, 1	0
Auto Framerate Sensor control	Enable the senor-internal control for frame rates (see also is_SetAutoParameter())	0 = off 1 = on	0
Brightness exposure Sensor control	Enable the sensor-internal brightness control	0 = off 1 = on	0
Brightness gain Sensor control	Enable the sensor-internal gain control	0 = off 1 = on	0
Brightness exposure Sensor control photometry	Not supported by DCx cameras		
Brightness gain Sensor control photometry	Not supported by DCx cameras		
Brightness control once		0, 1	0
Brightness reference	Reference value for brightness control	0...255	128
Brightness speed	Brightness control speed	0...100	50
Brightness max gain	Maximum gain for brightness control	0...100	100
Brightness max exposure	Maximum exposure time for brightness control	Depends on Pixelclock and image geometry	66.082816
Brightness Aoi Left	X start point of reference AOI for brightness control	0...(max. width*1-Aoi Width)	0
Brightness Aoi Top	Y start point of reference AOI for brightness control	0...(max. height*1-Aoi Height)	0
Brightness Aoi Width	Width of reference AOI for brightness control	Sensor-dependent*1	2560
Brightness Aoi Height	Height of reference AOI for brightness control	Sensor-dependent*1	1920
Brightness Hysteresis	Hysteresis value for auto exposure and gain (see IS_SET_AUTO_HYSTERESIS)	0…10	2
Brightness Skip Frames	Number of images that will be not analyzed for the control (see IS_SET_AUTO_SKIPFRAMES)	0…1000	4

Parameter	Description	Value range	Example
Auto WB control	Enable white balance control	0, 1	0
Auto WB type	White balance mode		
Auto WB RGB color mode	Color space of white balance (only active, if the mode "Auto (Kelvin)" is set)		
Auto WB offsetR	Red offset for white balance control	0...100	0
Auto WB offsetB	Blue offset for white balance control	0...100	0
Auto WB gainMin	Minimum gain for white balance control	0 <= gainMin <= gainMax <= 100	0
Auto WB gainMax	Maximum gain for white balance control	0 <= gainMin <= gainMax <= 100	100
Auto WB speed	White balance control speed	0...100	50
Auto WB Aoi Left	X start point of reference AOI for white balance control	0...(max. width[*1]-Aoi Width)	0
Auto WB Aoi Top	Y start point of reference AOI for white balance control	0...(max. height[*1]-Aoi Height)	0
Auto WB Aoi Width	Width of reference AOI for white balance control	Sensor-dependent[*1]	2560
Auto WB Aoi Height	Height of reference AOI for white balance control	Sensor-dependent[*1]	1920
Auto WB Once	Carry out white balance control once	0, 1	0
Auto WB Hysteresis	Hysteresis value for auto white balance (see IS_SET_AUTO_WB_HYSTERESIS)	0…10	2
Auto WB Skip Frames	Number of images that will not be analyzed for control (see IS_SET_AUTO_WB_SKIPFRAMES)	0…1000	4
[Trigger and Flash]	Sets the digital inputs/outputs		
Trigger mode	Trigger mode		
Trigger timeout	Timeout value for triggered image capture in 10 ms steps		
Trigger delay	Delay of triggered image capture in µs	Sensor-dependent[*1]	15
Trigger debounce mode	Not supported by DCx		

© 2015 Thorlabs Scientific Imaging

Parameter	Description	Value range	Example
	cameras		
Trigger debounce delay time	Not supported by DCx cameras		10
Trigger burst size	Not supported by DCx cameras		
Flash strobe	Activate flash output	0...6*2	0
Flash delay	Delay of the flash signal in µs	Depends on sensor setting, can be queried using is_IO()	0
Flash duration	Duration of the flash signal in µs	Depends on sensor setting, can be queried using is_IO()	200
PWM mode	PWM mode	see is_IO()	
PWM frequency	Frequency of the pulse-width modulation		
PWM duytcycle	Duty cycle of the pulse-width modulation	0.0…1.0 (1.0 corresponds to 100 %)	
GPIO state	State of the GPIO	see is_IO()	
GPIO direction	Direction of the GPIO	see is_IO()	
[Sequence AOI]	The sequence AOI is supported by DCC1240x / DCC3240x models only.		
NumberUsedAOI	Number of used AOIs	see is_AOI()	
StartX1…StartX3	X position of AOI 1-3		
StartY1…StartY3	Y position of AOI 1-3		
Gain1…Gain3	Gain of AOI 1-3		
Exposure1…Exposure3	Exposure of AOI 1-3		
ReadoutCycle1… ReadoutCycle3	Number of read-out cycles of AOI 1-3		
BinningMode1… BinningMode3	Binning mode of AOI 1-3		
SubsamplingMode1… SubsamplingMode3	Subsampling mode of AOI 1-3		
ScalerFactor1… ScalerFactor3	Scaling factor of AOI 1-3		
DetachImageParameter1… DetachImageParameter3	Changes of exposure time and gain are transferred to AOI 1-3	0 = Every change is transferred 1 = Changes are not transferred see also is_AOI()	
[Transfer]	Not supported by DCx cameras		
ImageDelay_us			
PacketInterval_us			

*1 This information is provided in Camera and sensor data chapter.

*2 For the parameters, please refer to the uc480.h header file provided in the \Develop\include

folder of the uc480 installation directory (see also Programming notes).

*3 Function obsolete, see chapter Obsolete functions.

6.5 Definition of IP Protection Classes

The housing of the DCx Cameras comply with **IP 30**:

- Protection against the ingression of small particles (diameter > 2.5 mm)
- No protection against water

6.6 History of API functions

New functions in software version 4.50

- `israw_AddFrame()`
- `israw_CloseFile()`
- `israw_ExitFile()`
- `israw_GetFrame()`
- `israw_GetImageInfo()`
- `israw_GetSize()`
- `israw_InitFile()`
- `israw_OpenFile()`
- `israw_SeekFrame()`
- `israw_SetImageInfo()`

New functions in software version 4.40

- `is_Gamma()`
- `is_LUT()`

New functions in software version 4.20

- `is_ImageBuffer()`
- `is_Measure()`

New functions in software version 4.01

- `is_Blacklevel()`

New functions in software version 4.00

- `is_AutoParameter()`
- `is_Convert()`
- `is_DeviceInfo()`
- `is_EdgeEnhancement()`
- `is_ImageFile()`
- `is_ParameterSet()`
- `is_PixelClock()`
- `is_Stream()`

New functions in software version 3.90

- `is_BootBoost()`

 © 2015 Thorlabs Scientific Imaging

- is_CaptureStatus()
- is_Exposure()
- is_IO()

New functions in software version 3.81

- is_Configuration()

New functions in software version 3.80

- is_AOI()
- is_DeviceFeature()
- is_HotPixel()

New functions in software version 3.70

- is_ColorTemperature()
- is_TriggerDebounce()

New functions in software version 3.61

- is_ScenePreset()
- is_Saturation()
- is_Sharpness()
- is_Zoom()

New functions in software version 3.52/3.60

- is_FaceDetection()
- is_Focus()
- is_ImageFormat()
- is_ImageStabilization()

New functions in software version 3.40

- is_DirectRenderer()
- is_GetImageInfo()
- is_GetDuration()
- is_GetSensorScalerInfo()
- is_SetSensorScaler()

New functions in software version 3.33

- is_Direct3D()
- is_GetTimeout()

New functions in software version 3.32

- is_GetTimeout()
- is_SetTimeout()
- is_SetTriggerCounter()

New functions in software version 3.30

- is_GetCameraLUT()
- is_GetCaptureErrorInfo()
- is_GetColorConverter()

- is_GetComportNumber()
- is_GetSupportedTestImages()
- is_GetTestImageValueRange()
- is_ResetCaptureErrorInfo()
- is_SetCameraLUT()
- is_SetColorConverter()
- is_SetSensorTestImage()

New function in software version 3.20

- is_SetOptimalCameraTiming()

New functions in software version 3.10

- is_EnableHdr()
- is_GetHdrKneepointInfo()
- is_GetHdrKneepoints()
- is_GetHdrMode()
- is_SetHdrKneepoints()

New functions in software version 3.00

- is_GetEthDeviceInfo()
- is_SetAutoCfgIpSetup()
- is_SetPacketFilter()
- is_SetPersistentIpCfg()
- is_SetStarterFirmware()

6.7 Certifications and Compliances

Compliance with the directives is demonstrated by meeting the following standards:

Product type	EMC immunity	EMC emission	UL certification
DCC1240x*1	EN 61000-6-2:2005	EN 61000-6-3:2001 + A11:2004	UL 60950-1, 2nd Edition, 2011-12-19 CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12
DCU223x *1 DCU224x *1	EN 61000-6-2:2001	EN 61000-6-4:2001	UL 60950-1, 2nd Edition, 2011-12-19 CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12
DCC1545M *1 DCC1645C *1	EN 61000-6-2:2005	EN 61000-6-3:2001 + A11:2004	UL 60950-1, 2nd Edition, 2011-12-19 CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12
DC3240x *1	EN 61000-6-2:2005	EN 61000-6-3:2007	UL 60950-1, 2nd Edition, 2011-12-19 CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12

*1 This equipment has been tested and found to comply with part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential

 © 2015 Thorlabs Scientific Imaging

area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.
Modifications not expressly approved by the manufacturer could void the user's authority to operated the equipment under FCC rules.

Certificates of Conformity available upon request.

6.8 Thorlabs 'End of Life' Policy (WEEE)

As required by the WEEE (Waste Electrical and Electronic Equipment Directive) of the European Community and the corresponding national laws, Thorlabs Scientific Imaging offers all end users in the EC the possibility to return “end of life” units without incurring disposal charges.

This offer is valid for Thorlabs Scientific Imaging electrical and electronic equipment

- sold after August 13th 2005
- marked correspondingly with the crossed out “wheelie bin” logo (see figure below)
- sold to a company or institute within the EC
- currently owned by a company or institute within the EC
- still complete, not disassembled and not contaminated

As the WEEE directive applies to self contained operational electrical and electronic products, this “end of life” take back service does not refer to other Thorlabs Scientific Imaging products, such as

- pure OEM products, that means assemblies to be built into a unit by the user (e. g. OEM laser driver cards)
- components
- mechanics and optics
- left over parts of units disassembled by the user (PCB’s, housings etc.).

Waste treatment on your own responsibility

If you do not return an “end of life” unit to Thorlabs Scientific Imaging, you must hand it to a company specialized in waste recovery. Do not dispose of the unit in a litter bin or at a public waste disposal site.

WEEE Number (Germany) : DE97581288

Ecological background

It is well known that waste treatment pollutes the environment by releasing toxic products during decomposition. The aim of the European RoHS Directive is to reduce the content of toxic substances in electronic products in the future.

The intent of the WEEE Directive is to enforce the recycling of WEEE. A controlled recycling of end-of-life products will thereby avoid negative impacts on the environment.

Crossed out "Wheelie Bin" symbol

 © 2015 Thorlabs Scientific Imaging

6.9 Exclusion of Liability and Copyright

Thorlabs Scientific Imaging has taken every possible care in preparing this Operation Manual. We however assume no liability for the content, completeness or quality of the information contained therein. The content of this manual is regularly updated and adapted to reflect the current status of the software. We furthermore do not guarantee that this product will function without errors, even if the stated specifications are adhered to.

Should you require further information about this product or encounter specific problems that are not discussed in sufficient detail in the User Manual, please contact your nearest Thorlabs office.

All rights reserved. This manual may not be reproduced, transmitted or translated to another language, either as a whole or in parts, without the prior written permission of *Thorlabs Scientific Imaging.*

Copyright © Thorlabs Scientific Imaging 2015. All rights reserved.

6.10 Thorlabs Worldwide Contacts

USA, Canada, and South America

Thorlabs, Inc.
56 Sparta Avenue
Newton, NJ 07860
USA
Tel: 973-579-7227
Fax: 973-300-3600
www.thorlabs.com
www.thorlabs.us (West Coast)
Email: sales@thorlabs.com
Support: techsupport@thorlabs.com

UK and Ireland

Thorlabs Ltd.
1 Saint Thomas Place, Ely
Cambridgeshire CB7 4EX
United Kingdom
Tel: +44-1353-654440
Fax: +44-1353-654444
www.thorlabs.com
Email: sales.uk@thorlabs.com
Support: techsupport.uk@thorlabs.com

Europe

Thorlabs GmbH
Hans-Böckler-Str. 6
85221 Dachau
Germany
Tel: +49-8131-5956-0
Fax: +49-8131-5956-99
www.thorlabs.de
Email: europe@thorlabs.com

Scandinavia

Thorlabs Sweden AB
Mölndalsvägen 3
412 63 Göteborg
Sweden
Tel: +46-31-733-30-00
Fax: +46-31-703-40-45
www.thorlabs.com
Email: scandinavia@thorlabs.com

France

Thorlabs SAS
109, rue des Côtes
78600 Maisons-Laffitte
France
Tel: +33-970 444 844
Fax: +33-811 38 17 48
www.thorlabs.com
Email: sales.fr@thorlabs.com

Brazil

Thorlabs Vendas de Fotônicos Ltda.
Rua Riachuelo, 171
São Carlos, SP 13560-110
Brazil
Tel: +55-16-3413 7062
Fax: +55-16-3413 7064
www.thorlabs.com
Email: brasil@thorlabs.com

Japan

Thorlabs Japan, Inc.
Higashi Ikebukuro
Q Building 2nd Floor 2-23-2
Toshima-ku, Tokyo 170-0013
Japan
Tel: +81-3-5979-8889
Fax: +81-3-5979-7285
www.thorlabs.jp
Email: sales@thorlabs.jp

China

Thorlabs China
Room A101, No. 100
Lane 2891, South Qilianshan Road
Putuo District
Shanghai 200331
China
Tel: +86-21-60561122
Fax: +86-21-32513480
www.thorlabs.hk
Email: chinasales@thorlabs.com

© 2015 Thorlabs Scientific Imaging

Index

A

ActiveX programming 412
Ambient conditions 453
AOI
  for automatic image control 119
AOI (Area of interest) 38
  AOI sequence 38
  multi AOI 38
AOI sequence 119
Application notes
  DCC1240x 67
  DCC1545M 70
  DCC1645C 71
  DCC3240x 67
  DCU223x 71
  DCU224x 71
Area of interest -> AOI 38
Area scan camera 37
Area scan sensor 37
Auto exposure shutter (AES) 51
Auto frame rate (AFR) 51
Auto gain (AGC) 51
Auto gain control (AGC) 51
Auto white balance (AWB) 51
Automatic image control
  hysteresis 51
  programming 107, 309
AVI functions
  errors 418
AVI recording 380

B

Bad pixel -> hot pixel 245
Bayer filter
  Bayer conversion 29
Binning 38, 43
Bit depth 47, 326
Black level correction 51
Blinking codes 470
Boot boost 76
Burst trigger mode 368

C

C# 411
C/C++ 410, 411, 412
Camera
  close 88
  EEPROM 76
  electrical data 454
  Information 87
  open 88
  parameters 80
Camera EEPROM 108
Camera ID 76
Camera Manager 72
  camera list 73
  Control center 74
Camera parameters 76
  load at start-up 147
Camera status
  get 108
Capture
  freerun synchronisation 92
  live mode (freerun) 18
  overlap trigger 20
  single snap mode 18
  trigger mode 20
Capture video 140
Characteristics
  gamma curve 44
  linear 44
  logarithmic 44
C-mount/CS-mount 442
Color depth 326
Color formats 472
COM port -> Serial interface 57
Contrast adjustment 47
CPU idle state 76
CRA correction 26

D

DCC3240x
  digital input 462
  digital output 463
  electrical data 459
  flsh 463
  GPIO 460
  I/O connector 459
  pin assignment 459

DCC3240x
  serial interface 464
  trigger 462
DCU223x / DCU224x
  electrical data 454
  I/O connector 454
  Pin assignment 454
DCx
  driver compatibility 65
DCx camera model 12
  DCC3240x 12
  DCU223x / DCC224x 13
DCx camera models 12, 420
  comparison table 421
DCx Model Naming Conventions 423
DCx naming convention 423
Delphi 412
DIB 22
Digital input
  trigger 54
Digital input/output 21, 54
  programming 336
Digital output
  flash 54
Direct3D 22, 165, 330
DirectShow 76
DirectX 64, 165
Display
  bitmap mode (DIB) 22
  color formats 472
  DIB mode 91
  Direct3D 22, 64, 91
  DirectDraw 407
  OpenGL 22
  overlay 91
  overlay display 165
Display overlay 91
Driver version 75

E

EEPROM of the camera 76, 465
Errors
  error messages 414
  transmission errors 50
  troubleshooting 468
Event handling 100
Exposure time 50
  setting 188

F

Fast line scan 67
Fill factor 26
Filter glasses 446
  clean 446
  DL filter 446
  IR cut filter 446
Firmware 53
Flange back distance 442
  calculate 442
Flash 20, 33, 271
Flash timing
  in freerun mode 56
  in trigger mode 54
Frame rate 50, 338
Freerun -> Capture: live mode 18
Freerun mode 140

G

Gain 343
  linearity 51
Gamma characteristic 44
Gamma-Kennlinie 201
Global shutter 33
GPIO 271
Graphics card 22, 64

H

Histogram 47
Hot pixel correction 31, 245
Hotpixel
  add 81
  edit 81

I

Image capture -> Capture 20
Image display -> Display 22
Image format
  set 254
Image memory
  allocate 88
INI file -> Parameter file 475
Interface
  serial 464
IP norm 482
IR cut filter 446

 © 2015 Thorlabs Scientific Imaging

is_AddToSequence 116
is_AllocImageMem 117
is_AOI 119
is_AutoParameter 129
is_Blacklevel 132
is_CameraStatus 135
is_CaptureStatus 137
is_CaptureVideo 140
is_ClearSequence 142
is_ColorTemperature 143
is_Configuration 147
is_Convert 154
is_CopyImageMem 156
is_CopyImageMemLines 157
is_DeviceFeature 158
is_DeviceInfo 163
is_DirectRenderer 165
is_DisableEvent 175
is_EdgeEnhancement 176
is_EnableAutoExit 179
is_EnableEvent 180
is_EnableMessage 183
is_ExitCamera 185
is_ExitEvent 186
is_ExitImageQueue 187
is_Exposure 188
is_ForceTrigger 195
is_FreeImageMem 197
is_FreezeVideo 199
is_Gamma 201
is_GetActiveImageMem 202
is_GetActSeqBuf 204
is_GetAutoInfo 206
is_GetBusSpeed 210
is_getCameraInfo 211
is_GetCameraList 213
is_GetCameraLUT 215
is_GetColorConverter 216
is_GetColorDepth 217
is_GetDLLVersion 218
is_GetError 219
is_GetFramesPerSecond 220
is_GetFrameTimeRange 221
is_GetImageHistogram 223
is_GetImageMem 230
is_GetImageMemPitch 231
is_GetNumberOfCameras 233
is_GetOsVersion 234
is_GetSensorInfo 235
is_GetSensorScalerInfo 237
is_GetSupportedTestImages 238
is_GetTestImageValueRange 240
is_GetTimeout 241
is_GetUsedBandwidth 242
is_GetVsyncCount 243
is_HasVideoStarted 244
is_HotPixel 245
is_ImageFile 250
is_ImageFormat 254
is_InitCamera 261
is_InitEvent 265
is_InitImageQueue 267
is_InquireImageMem 269
is_IO 271
is_IsVideoFinish 284
is_LockSeqBuf 286
is_LUT 287
is_LUT() 407
is_Measure 290
is_ParameterSet 294
is_PixelClock 297
is_ReadEEPROM 300
is_RenderBitmap 302
is_ResetToDefault 304
IS_SET_EVENT_TRANSFER_FAILED 180
is_SetAllocatedImageMem 306
is_SetAutoParameter 309
is_SetBinning 317
is_SetCameraID 320
is_SetColorConverter 322
is_SetColorCorrection 324
is_SetColorMode 326
is_SetDisplayMode 330
is_SetDisplayPos 334
is_SetErrorReport 335
is_SetExternalTrigger 336
is_SetFrameRate 338
is_SetGainBoost 340
is_SetGamma 341
is_SetHardwareGain 343
is_SetHWGainFactor 346
is_SetImageMem 348
is_SetOptimalCameraTiming 349

is_SetRopEffect 351
is_SetSaturation 353
is_SetSensorScaler 354
is_SetSensorTestImage 357
is_SetSubSampling 359
is_SetTimeout 363
is_SetTriggerCounter 365
is_SetTriggerDelay 366
is_StopLiveVideo 367
is_Trigger 368
is_TriggerDebounce 370
is_UnlockSeqBuf 374
is_WaitEvent 375
is_WaitForNextImage 376
is_WriteEEPROM 378
isavi_AddFrame 380
isavi_CloseAVI 381
isavi_DisableEvent 381
isavi_EnableEvent 382
isavi_ExitAVI 383
isavi_ExitEvent 384
isavi_GetAVIFileName 385
isavi_GetAVIFileNameW 386
isavi_GetAVISize 387
isavi_GetnCompressedFrames 388
isavi_GetnLostFrames 389
isavi_InitAVI 390
isavi_InitEvent 391
isavi_OpenAVI 392
isavi_OpenAVIW 393
isavi_ResetFrameCounters 394
isavi_SetFrameRate 395
isavi_SetImageQuality 396
isavi_SetImageSize 397
isavi_StartAVI 399
isavi_StopAVI 400
israw_AddFrame 401
israw_CloseFile 401
israw_ExitFile 402
israw_GetFrame 402
israw_GetImageInfo 403
israw_GetSize 404
israw_InitFile 404
israw_OpenFile 405
israw_SeekFrame 406
israw_SetImageInfo 406

L

LED 271
LED -> Status LED
  DCx Status LED 470
Lens
  immersion depth 444
Line scan
  fast line scan 67
Line scan mode 37, 158
Linux 64
  not supported functions 418
Live mode 140
Log mode 158
Lookup table (LUT) 44, 45

M

Memory board 407
Memory formats 472
Micro lenses 26
Multi AOI 38, 119
Multi camera systems
  system requirements 64

N

Network card 64

O

Obsolete functions 407
OpenGL 22, 330
Operating system 64
Overlap trigger -> Capture 20
Overlay -> Display 22
Overlay: Direct3D
  display (programming) 91
Overlay: display 165

P

Parameter
  new 53
Parameter file (ini file) 475
Pixel clock 50
Pixel pre-processing 45
Port 76
Ports 57
Processor

 © 2015 Thorlabs Scientific Imaging

Processor
  idle states 147
  support for multitasking 147
Programming languages 410, 411, 412
Programming notes 410
PWM 271

Q

Quick start
  connection 62
  image capture 62
  programming 85

R

Raw Bayer -> Bayer filter 29
RAW file 401
Reference AOI
  for automatic image control 119
Return values 414
RGB gain 51
ROI -> AOI 38, 119
Rolling shutter 33
Rolling shutter (global start) 33
RS-232 -> Serial interface 57

S

Sensor
  position accuracy 445
Sensor formats 25
Serial interface
  DCC3240x 57
Setting the area of interest -> AOI 119
Shock resistance 453
Shutter
  global 33
  rolling 33
Standby mode of the camera 21
Starter firmware 53, 261
Status LED 65
  DCx camera 470
Subsampling 38, 42
support 468
Support file
  create 76
System requirements 64

T

Temperature range 453
Thread programming 412
Trigger 271
Trigger mode 368
Trigger mode -> Image capture 20

U

UART 464
uc480 Camera Manager 72
  Control Center 74
uc480 Hotpixel Editor 81
uc480 programs 72
uc480.h 410, 411, 412
uc480_Api.dll 410, 411, 412
USB
  bandwidth 60
  cable 59
  connection 59
  hub 65
  standard 58
  topology 58
USB 3.0
  cable 60
  conncetion 60

V

VB .NET 412
Vibration resistance 453

W

Windows 64