THORLABS

CDD and CMOS Cameras

# DCU223x, DCU224x
# DCC1240x
# DCC1545M, DCC1645C
# DCC3240X

# Operation Manual and SDK

2013

Version: 4.20
Date: 25.03.2013

# Contents

**We aim to develop and produce the best solution for your application in the field of optical measurement technique. To help us to live up to your expectations and improve our products permanently we need your ideas and suggestions. Therefore, please let us know about possible criticism or ideas. We and our international partners are looking forward to hearing from you.**

***Thorlabs GmbH***

**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.**

**Note**

**This manual also contains "NOTES" and "HINTS" written in this form.**

**Please read these advices carefully!**

# 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 54
- First steps to DCx Camera programming 127
- The uc480 Viewer 75 application

**Further important information**

- What is new in this version? 16
- Contents 15 of this Manual
- The DCx camera family 14
- Specifications 458

**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.

All modules must only be operated with proper shielded connection cables.

Only with written consent from Thorlabs may changes to single components be carried out or components not supplied by Thorlabs be used.

This precision device is only serviceable if properly packed into the complete original packaging including the plastic foam sleeves. If necessary, ask for a replacement package.

## 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:

| | **Minimum**[*1] | **Recommended** |
|---|---|---|
| CPU speed | 600 MHz | 2 x 2.4 GHz |
| Memory (RAM) | 256 MB | 2048 MByte |
| USB host controller | USB 2.0 high speed (480 Mbps)<br>USB 3.0 (4000 Mbps) for DC3240x cameras | USB 2.0 high speed (480 Mbps)<br>USB 3.0 (4000 Mbps) for DC3240x cameras<br>Intel® or NVIDIA® nForce mainboard chipset |
| Graphics card | Onboard graphics chip | AGP/PCIe graphics card<br>Latest version of Microsoft DirectX Runtime 9.0c |
| Operating system | Windows 7 32 or 64 bit<br>Windows XP 32 bit (Service Pack 2)<br><br>Linux (Kernel 2.6) | Windows 7 32 or 64 bit<br>Windows Vista 32 or 64 bit (Service Pack 1) [*3]<br>Windows XP 32 bit (Service Pack 3)<br>Linux (Kernel 2.6) |

[*1] With the minimum system requirements the camera performance may be limited.

[*3] DCC3240x USB 3.0 cameras are not supported under Windows Vista.

**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 PCs with energy saving CPU technology 500 for hints and possible solutions.

**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**

For DCx 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.

**Software Requirements under Linux**

For operating the DCx camera under Linux the following components must be installed:

| Component | Version |
|---|---|
| Linux-Kernel | 2.6.9 up to 2.6.24 |
| The standard C library `libc/glibc` | 2.0 or higher |
| GNU Compiler Collection `gcc` | 3.4 or higher |
| POSIX Thread Library<br>(POSIX Threads Enabled `libc`) | - |
| `bash` or `sh` shell<br>(for running the installation script) | - |
| Qt (for compiling the demo program) | - |

# 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 30 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 459 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 (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 480 Mbps. 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 17, on sensor technology 23, important camera parameters 44, and the USB 51 interfaces. We recommend to read this chapter to become familiar with the general functionality of the DCx Cameras.

**Section B: Operation**

- Quick start 54 to using your DCxCamera
- Installing 56 and Using DCx Camera software 67
  These sections show how to connect cameras and start operation using the software tools uc480 Camera Manager and uc480 Viewer.
- Application notes by camera model 63
  This section explains special features and limitations of some camera models.

**Section C: Programming**

- First steps 127 to programming with your DCxCamera
- How to proceed 129
  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 155/Description of AVI functions 359
  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 379
  This chapter lists obsolete API functions and recommended alternatives.
- Lists and programming notes 454
  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 458
  All information on the camera's sensor and performance 460, mechanical 472 as well as electrical specifications 487 are contained in this section.
- Accessories 497
  Here you will find a list of accessories for DCx Cameras sorted by model.

**Appendix**

- Information on Troubleshooting 499
- Status LEDs on USB DCx 501 cameras
- Color and memory formats 502
- uc480 parameter file (ini file) 504
- Definition of IP protection classes 509

# 1.6 What's New in this Version?

Version 4.20 of the DCxCamera software package includes many new features and enhancements. The following table gives you an overview of the major new functions.

Please make sure to also read the file named `WhatsNew.txt` which you will find in the `C:\Program Files\Thorlabs\DCx Cameras\Help` directory when the installation is completed. This file contains late-breaking information on new functions and fixed issues.

**New in Version 4.20**

| Cameras & functions | Described in chapter |
|---|---|
| CCD models with hardware revision 3:<br>• Improved long time exposure function<br>• 12 bit per pixel now also available for USB 2.0 cameras | Camera and sensor data 460 |
| More functions for the USB 3 DCC3240 camera series:<br>• 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.<br>• 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()` 290 |
| New color formats are added:<br>• `IS_CM_RGB12_PACKED`<br>• `IS_CM_RGBA12_PACKED`<br>• `IS_CM_BGR12_PACKED`<br>• `IS_CM_BGRA12_PACKED`<br>• `IS_CM_RGB8_PLANAR`<br>The following formats are renamed and the previous formats are moved to the `uc480_deprecated.h` header file:<br>• `IS_CM_BGR10V2_PACKED` (new `IS_CM_BGR10_PACKED`)<br>• `IS_CM_RGB10V2_PACKED` (new: `IS_CM_RGB10_PACKED`)<br>• `IS_CM_BGR555_PACKED` (new: `IS_CM_BGR5_PACKED`)<br>The following formats are moved to the `uc480_deprecated.h` header file as they are identical to existing formats:<br>• `IS_CM_BAYER_RG8` (now: `IS_CM_SENSOR_RAW8`)<br>• `IS_CM_BAYER_RG12` (now: `IS_CM_SENSOR_RAW12`)<br>• `IS_CM_BAYER_RG16` (now: `IS_CM_SENSOR_RAW16`) | `is_SetColorMode()` 319<br>Color and memory formats 502 |
| New camera demos:<br>• Simultaneous opening of multiple cameras and sending a single software trigger (Multi-camera demo)<br>• 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()` 159 |

**Older versions**

See the History of uc480 Software Versions 509 and History of uc480 API functions 516 chapters.

# 2 Camera Basics

This chapter explains the basics of DCx Camera technology.



## 2.1 Operating Modes

DCx Cameras support the following operating modes:



### 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.

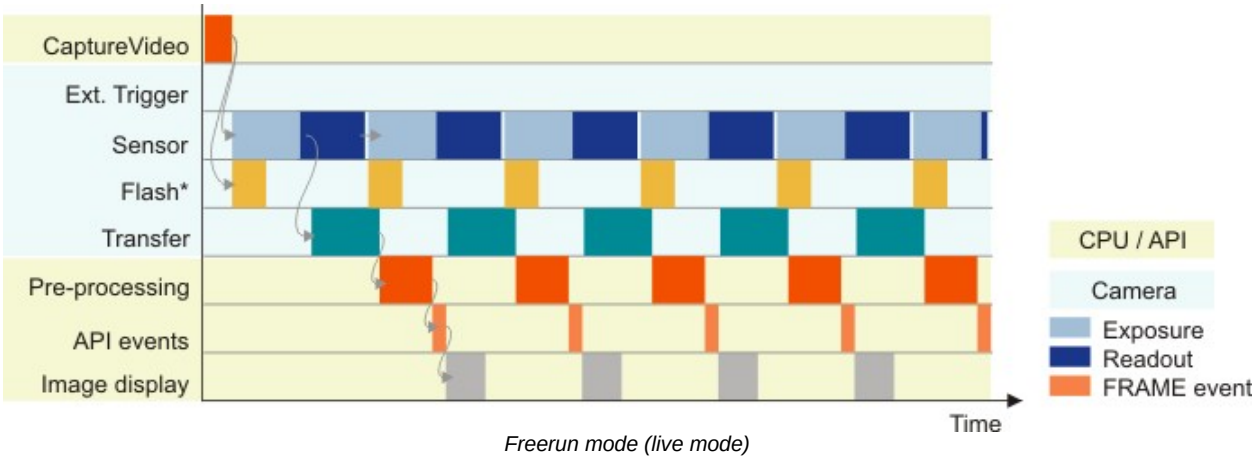
**Note**

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) 49 chapter.

**Continuous mode (live mode)**

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




*Freerun mode (live mode)*

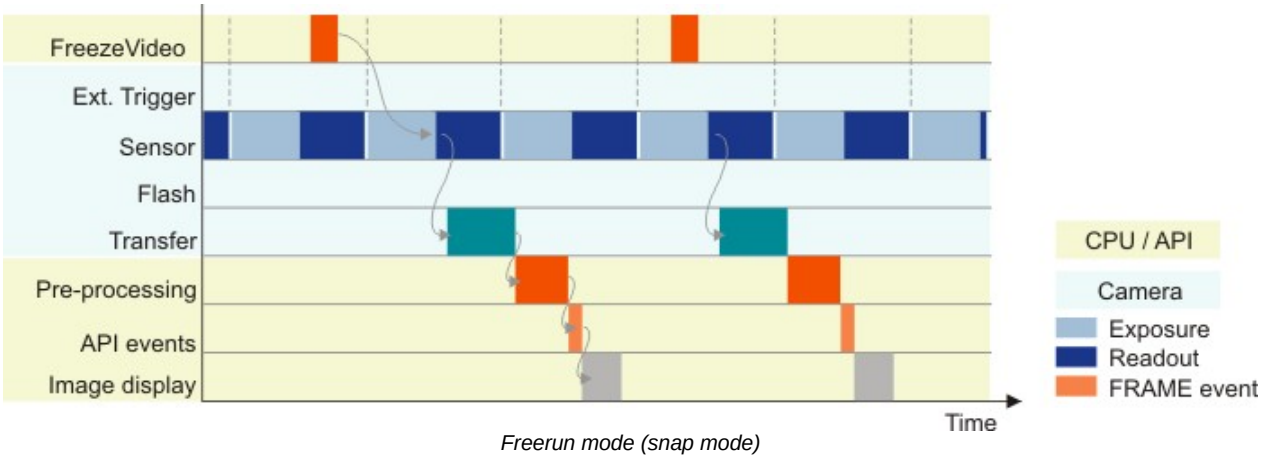
* Flash function optional. See also Digital in-/output (trigger/flash) 49.

**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 30
- Basics: Trigger mode 19
- Basics: Applying new parameters 46

**Programming:**

- Capture modes 134

## 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 camera properties 104 of the uc480 Viewer application or use the API command. 327

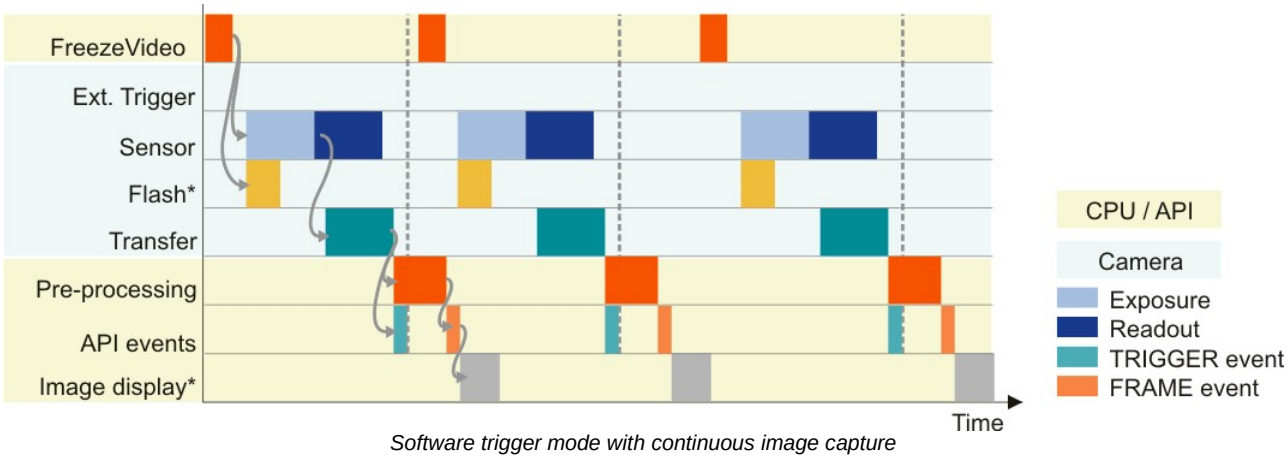
**Note**

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) 47 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) 47

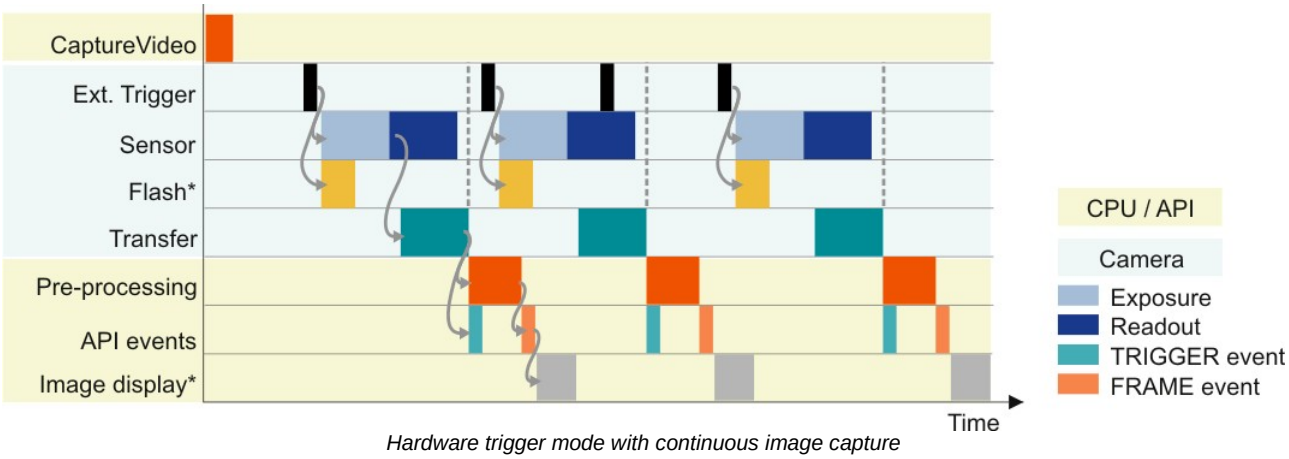
**Hardware trigger mode**

When this mode is enabled, calling the `is_FreezeVideo()` 223 (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()` 177 (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.




*Hardware trigger mode with continuous image capture*

* Optional flash function. See also Digital input/output (trigger/flash) 47

**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 17
- Basics: Digital input/output (trigger/flash) 47
- uc480 Viewer: Trigger 103

**Programming:**

- Image capture modes: Trigger 134

## 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 133.

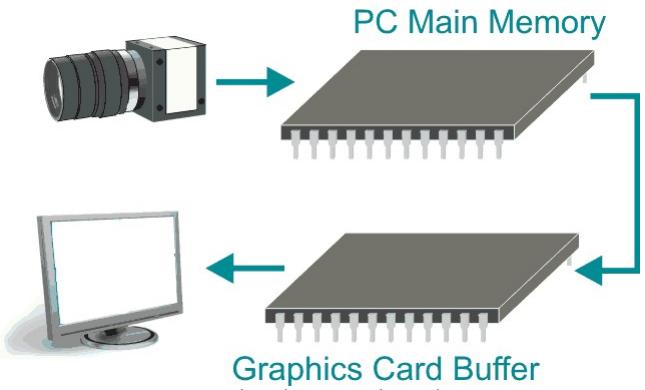
**Attention**

The "DirectDraw BackBuffer" and "DirectDraw Overlay Surface" display modes are obsolete. Please use the Direct3D functions instead (see also Obsolete functions 379).

**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()` 297 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.

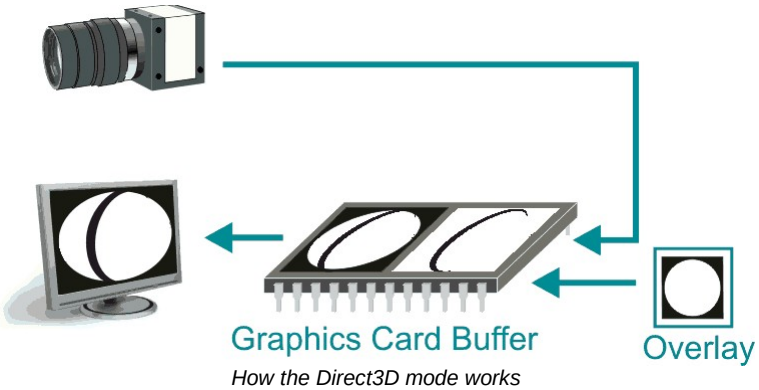



*How the Bitmap mode works*

**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 56). 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()` 198 API function.

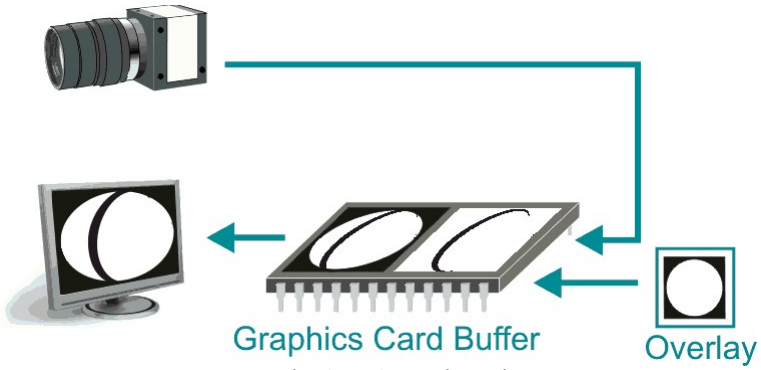



*How the Direct3D mode works*

**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 programmed by the user. | Integrated. Complex overlays can be displayed without flicker. | Integrated. |
| 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. |

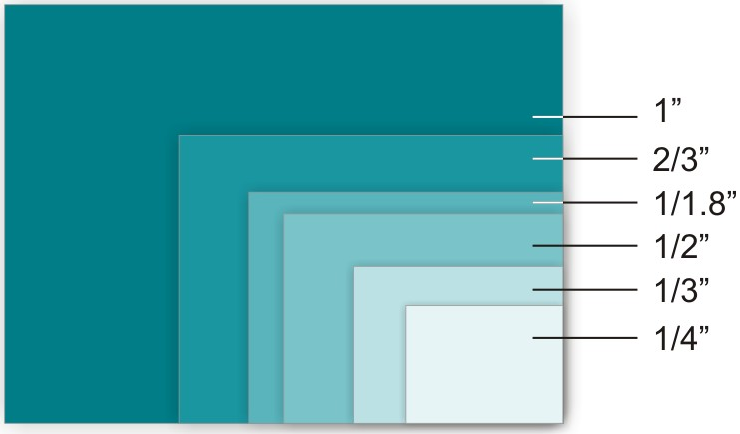
## 2.3 Sensor



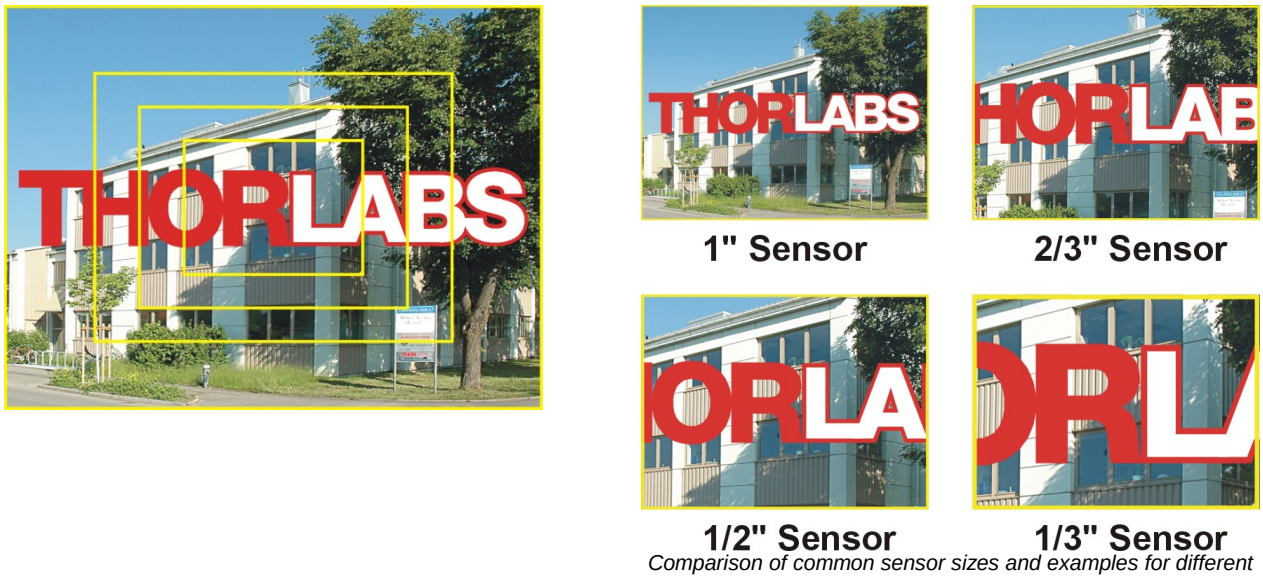
### 2.3.1 Sensor Sizes

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*

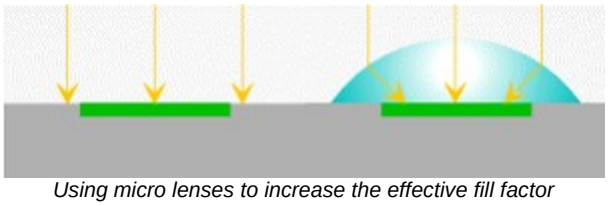
The size of each single sensor cell (pixel) depends on the size of the active sensor area and the resolution. In general, less pixels over the same sensor area (or a larger sensor area with the same resolution) will result in

greater photo sensitivity of the sensor.

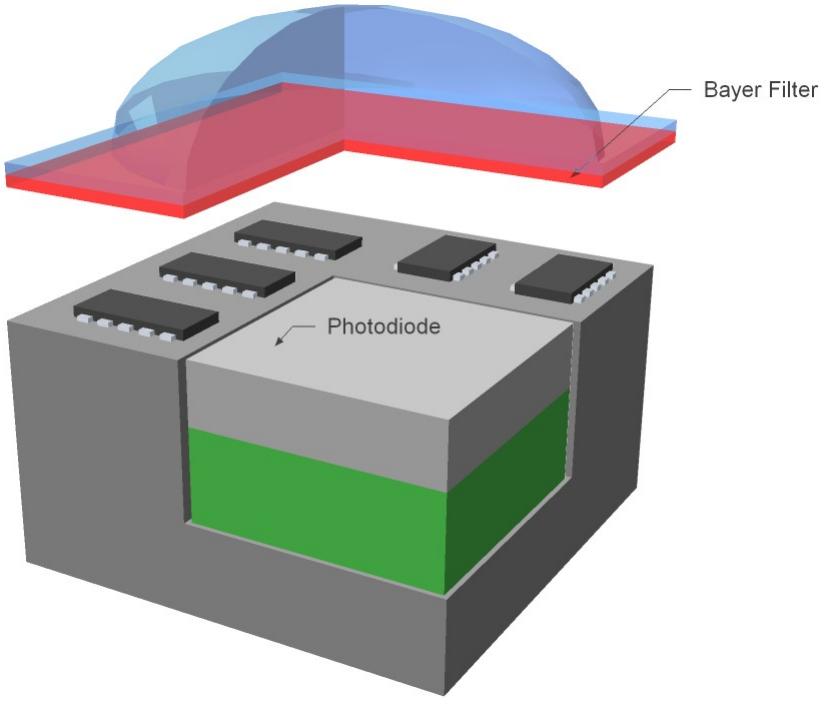
## 2.3.2 Micro Lenses

**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.



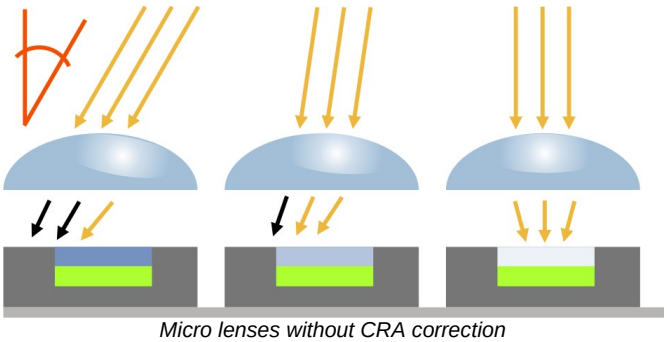
*Using micro lenses to increase the effective fill factor*




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

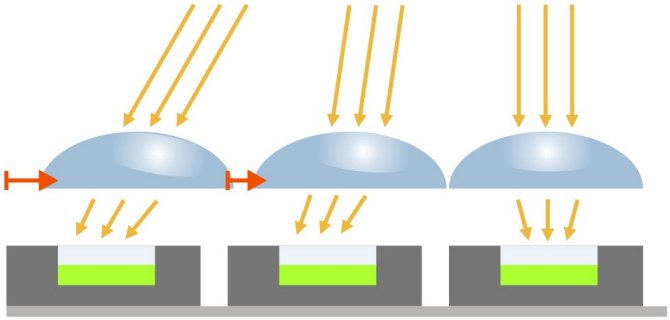
### Micro lenses with CRA correction

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*

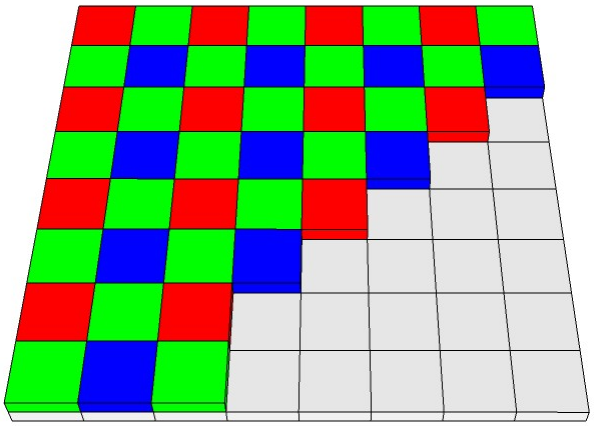
**Note**

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 461
- DCC1645C 466

### 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.

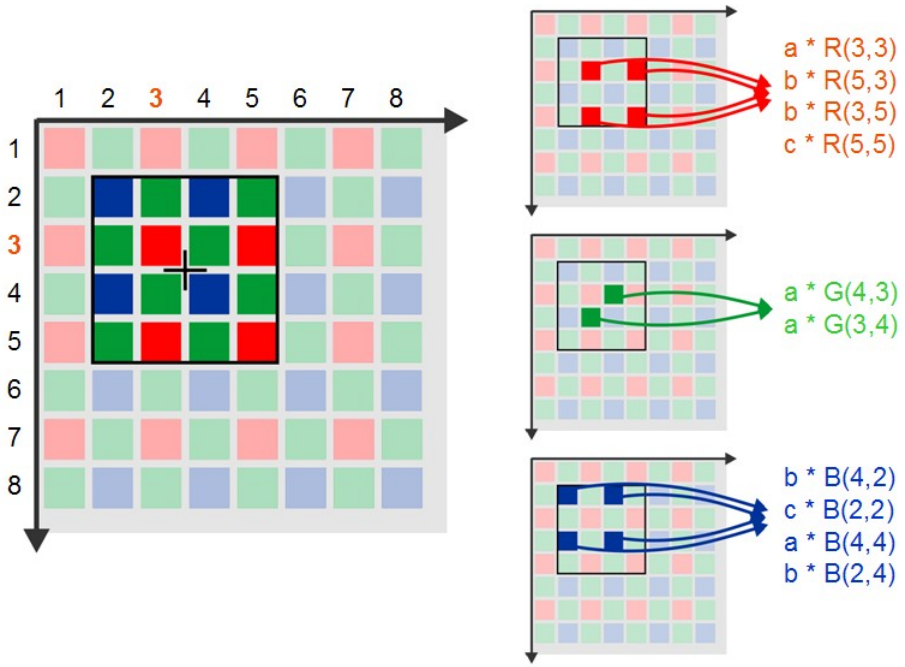
**Note**

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:**

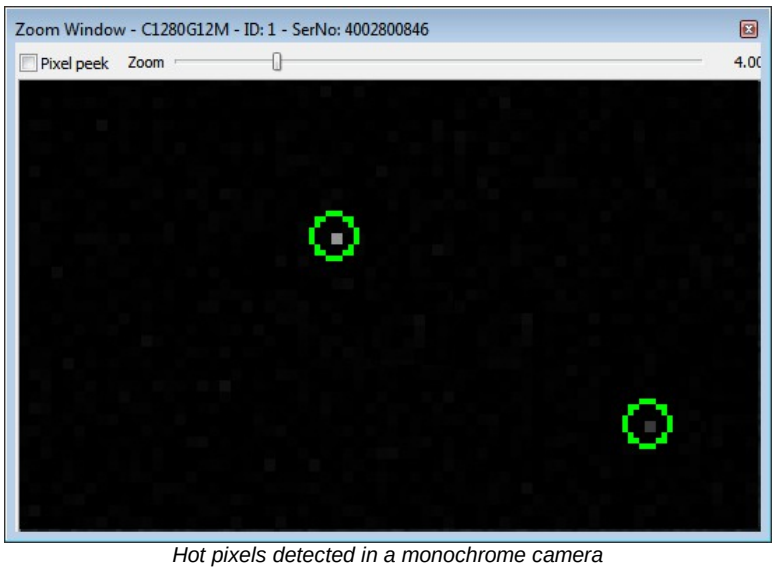
- Color conversion: `is_SetColorConverter()` 317
- uc480 Viewer: Format 99

## 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:**

- uc480 Viewer: Hot pixel correction 110
- uc480 Hotpixel Editor 123
- Programming: `is_HotPixel()` 260

## 2.3.5 Shutter Methods



**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) 47 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

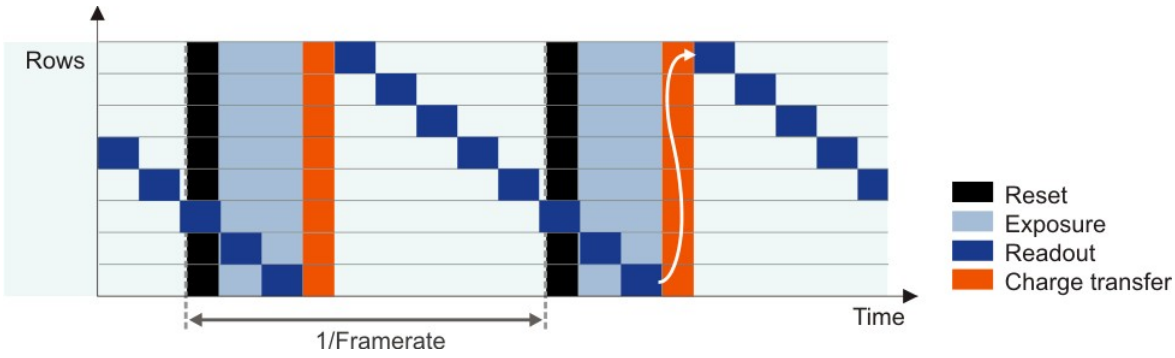
The sensor cells must not be exposed during the readout process. The sensors of the DCx Cameras have no mechanical shutters, but work with electronic shutter methods instead. Depending on the sensor type, either the rolling shutter method or the global shutter method is used.

**Global shutter**

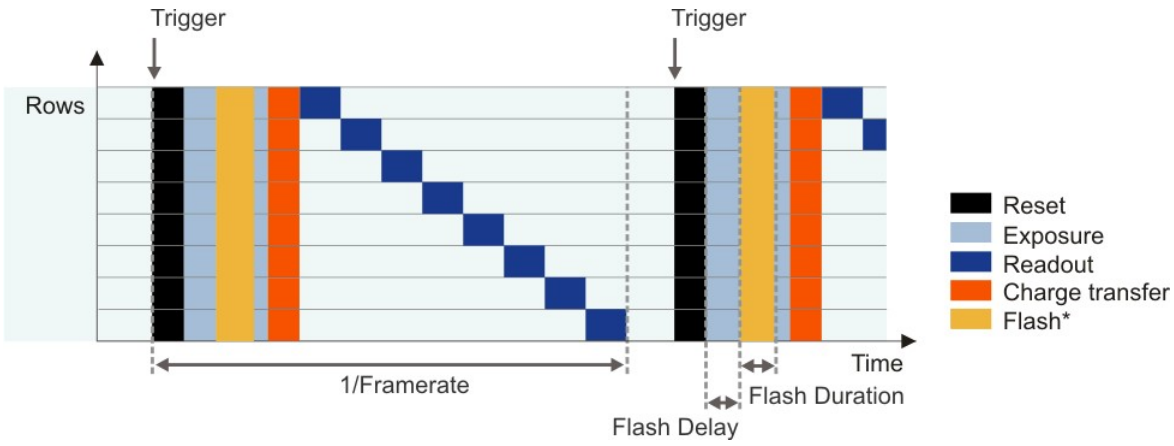
On a global shutter sensor, all pixel rows are reset and then exposed simultaneously. At the end of the exposure, all rows are simultaneously moved to a darkened area of the sensor. The pixels are then read out row by row.

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 104).

**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.

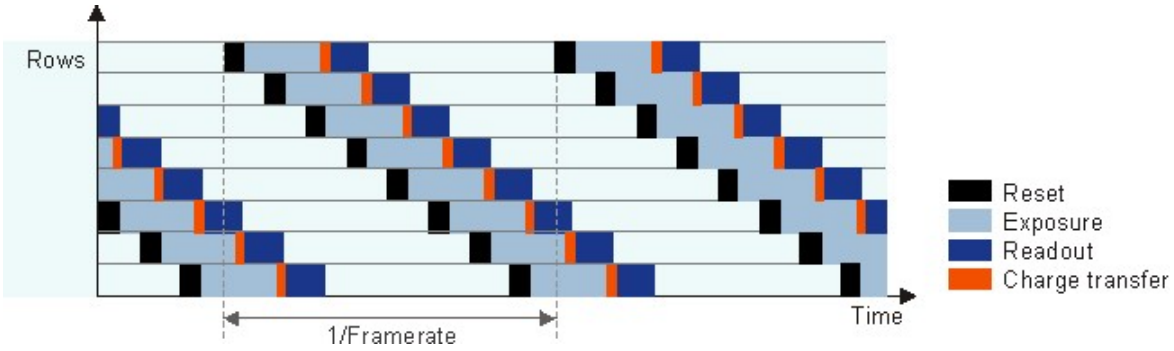
*Example for the rolling shutter effect with a moving car*

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.

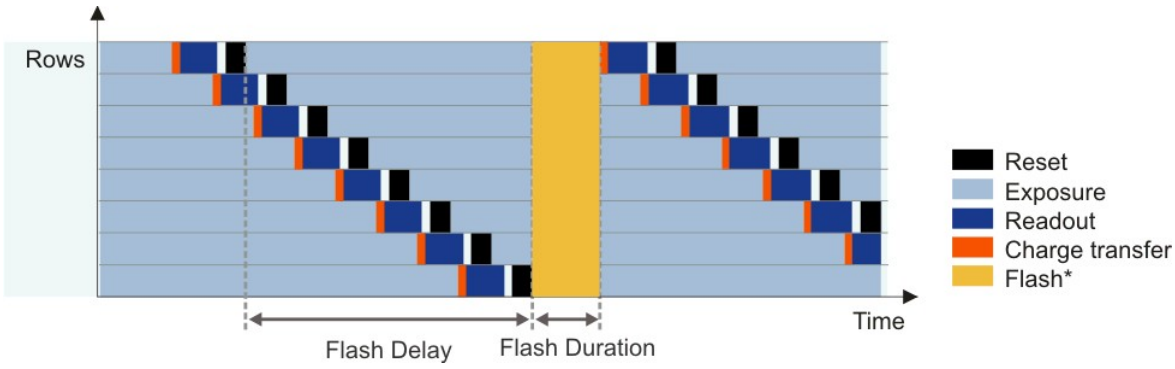
Rolling shutter sensors offer a higher pixel density compared to global shutter CMOS sensors. The rolling shutter system is used in DCC Cameras with high-resolution CMOS sensors.

**Note**

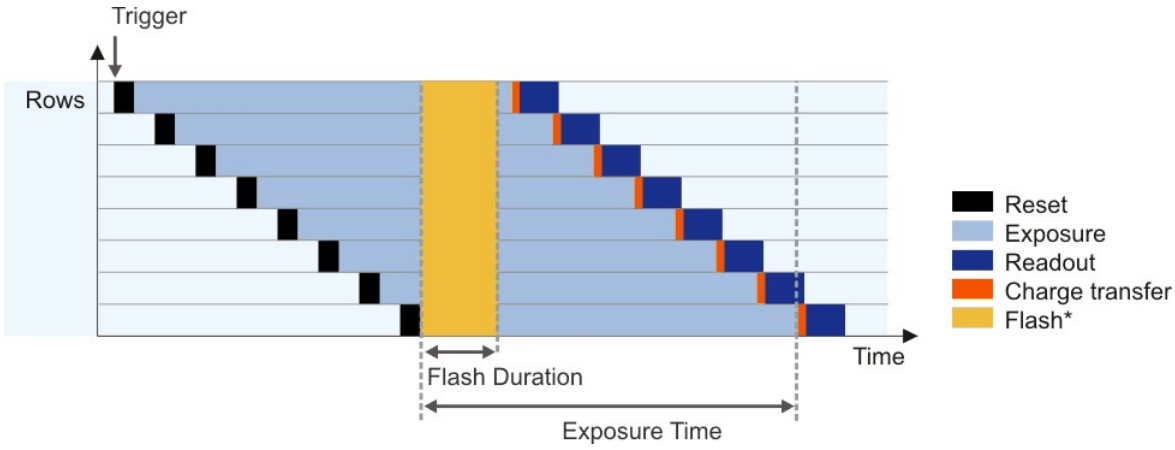
Some CMOS senors with global shutter can be operated also with rolling shutter. The operation in the rolling shutter mode is used to reduce the image noise. This function is only supported from the camera models DCC1240x/DCC3240x 461.




*Rolling shutter sensor in live mode*




*Rolling shutter sensor in live mode with global flash window*

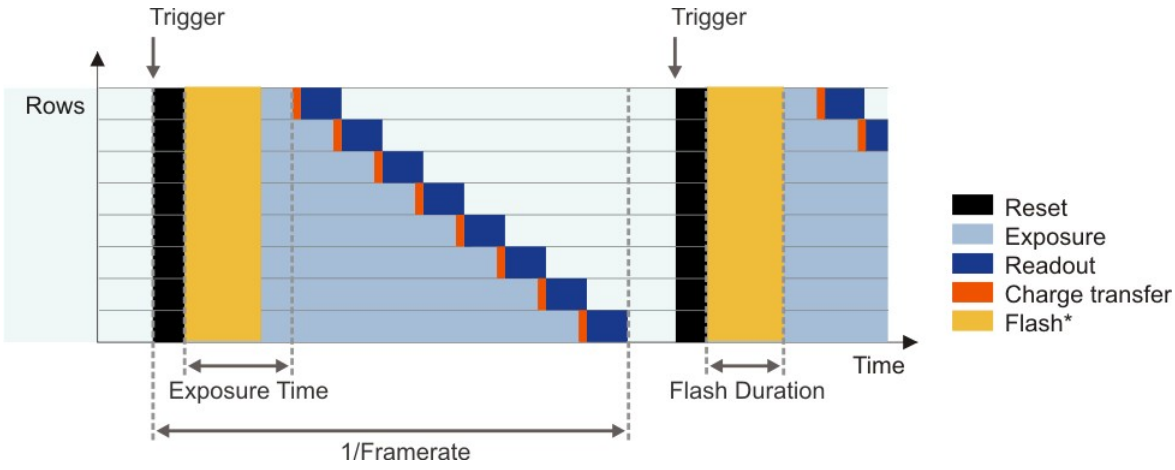



*Rolling shutter sensor in triggered mode with global flash window*

* Optional flash function. The start time and duration are defined by the flash delay and duration parameters (see also Camera settings: I/O 104).

**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 104).

## 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 26 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 461 models in form of the fast line scan mode. The triggered line scan mode is not supported by any camera model yet.

**See also:**

- uc480 Viewer: Properties > Shutter 115

**Programming:**

- Function: `is_DeviceFeature()` 192

## 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:

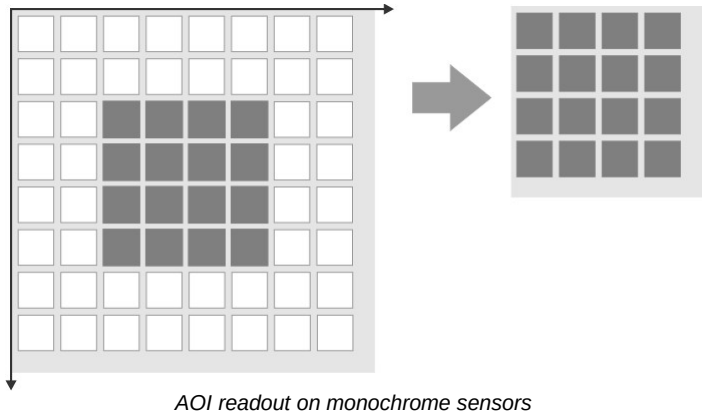
- Area of interest (AOI) 34
- Subsampling 37 (skipping) pixels
- Binning 38 (combining) pixels

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.

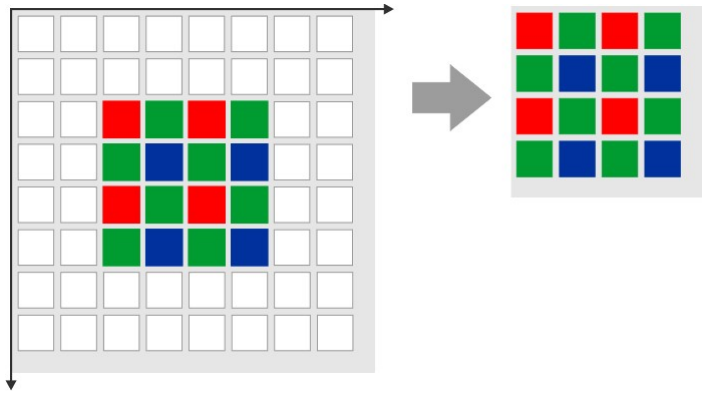
### 2.4.1 Area of Interest (AOI)

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 460 chapter.



*AOI readout on monochrome sensors*



*AOI readout on color sensors*

**Note**

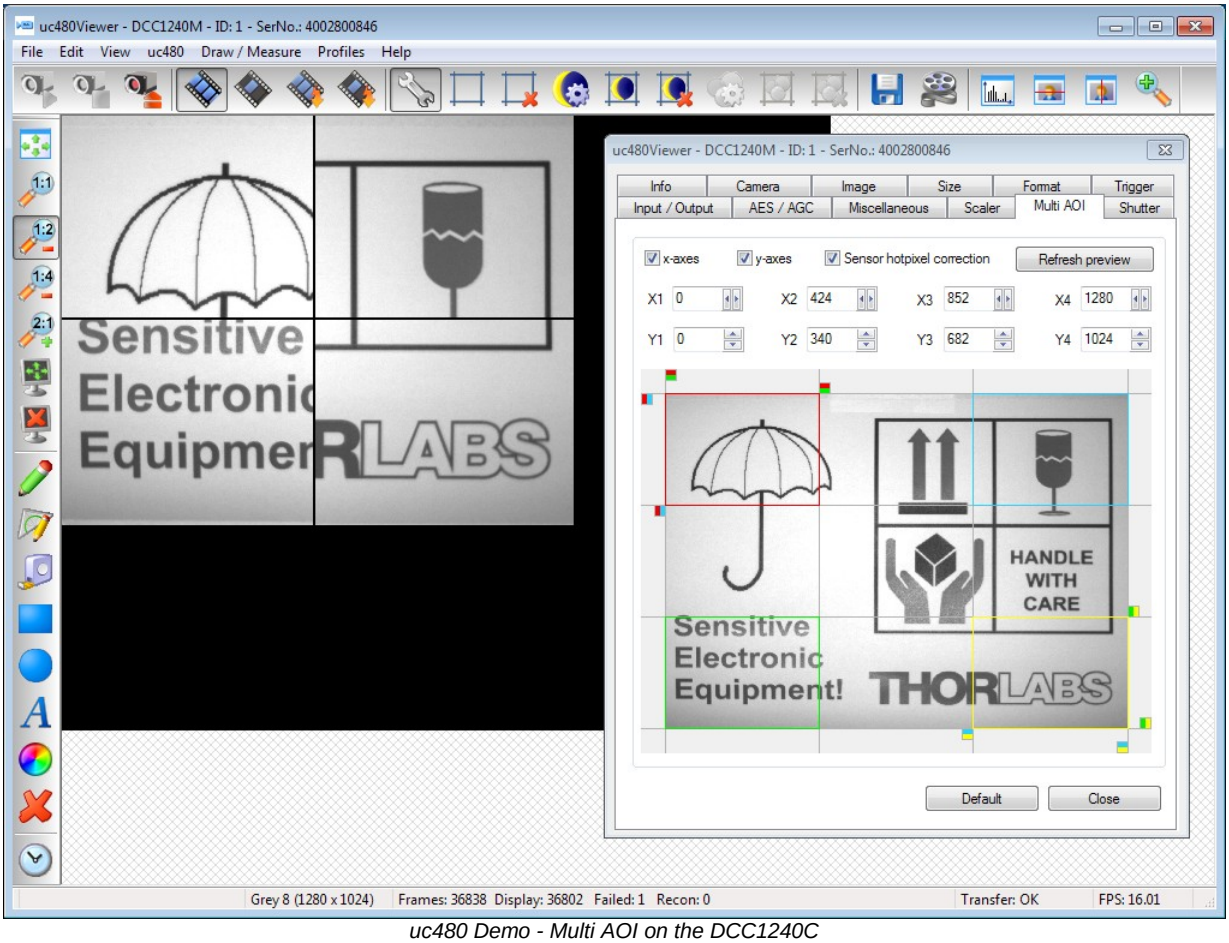
**Step widths for AOI definition (position grid)**: The available step widths for the position and size of image AOIs depend on the sensor. The values defining the position and size of an AOI have to be integer multiples of the allowed step widths.

For details on the AOI grids of the individual camera models, please see Camera and sensor data 460 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**

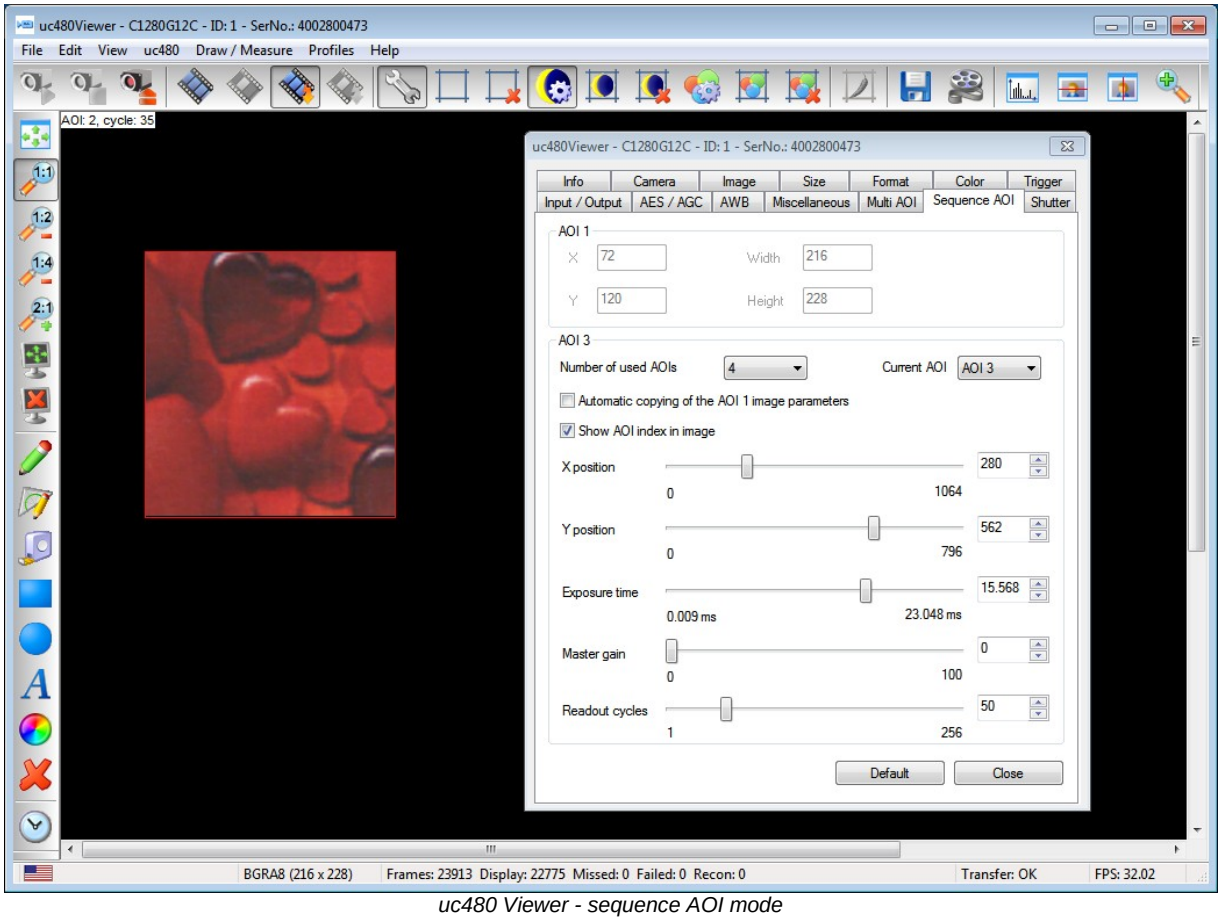
The Multi AOI function allows defining more than one AOI in an image and transferring these AOIs all at the same time. Only DCC1240x models support this feature. In the Multi AOI mode you can define two or four AOIs in one image and transfer them simultaneously. The AOIs are positioned side by side or one below the other, and share the same X or Y axis.




*uc480 Demo - Multi AOI on the DCC1240C*

### Sequence AOI mode

Apart from the multi AOI mode, DCC1240x and DCC3240x also support the sequence AOI mode. 113 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.




*uc480 Viewer - sequence AOI mode*

**See also:**

- uc480 Viewer: Size 97
- uc480 Viewer: Multi AOI 112
- uc480 Viewer: Sequence AOI 113
- Programming: `is_AOI()` 159

## 2.4.2 Subsampling

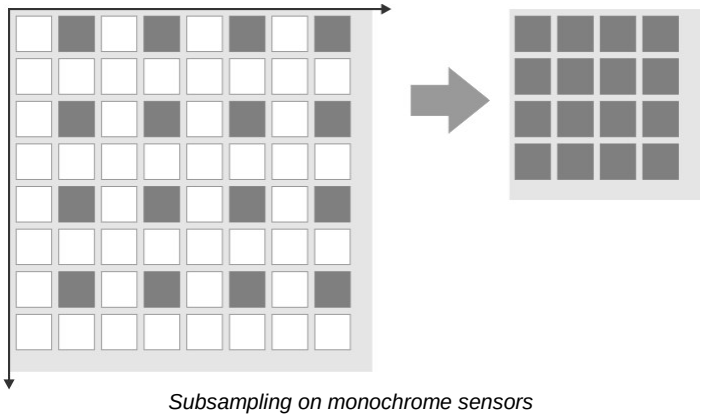
Subsampling is a technique that skips multiple sensor pixels when reading out image data. This reduces the amount of data to be transferred and enables higher camera frame rates. The captured image has a lower resolution but still the same field of view compared to the full-resolution image. This mode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

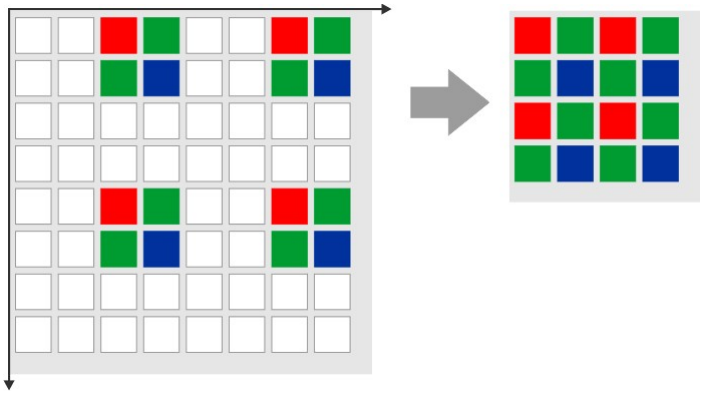
Monochrome sensors and some color sensors ignore the Bayer pattern and the color information gets lost (mono subsampling).

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 460 chapter lists the subsampling methods and factors supported by each camera model.



*Subsampling on monochrome sensors*



*Subsampling on color sensors*

## 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.

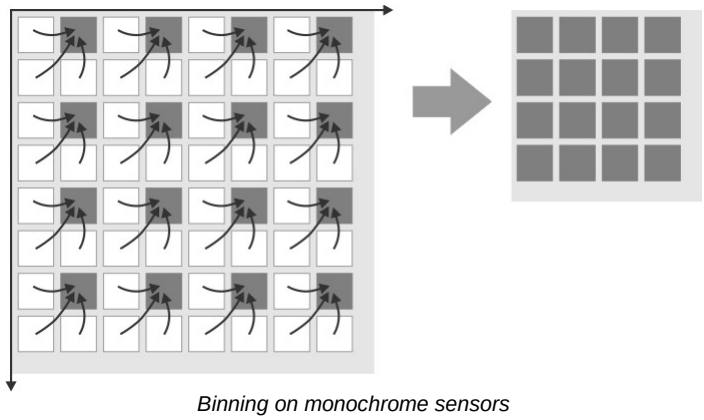
Color binning, as performed by most color sensors, combines only pixels of the same color (see also the Color filter (Bayer filter) 26 chapter). For some monochrome sensors, the camera also performs color binning, resulting in slight artifacts.

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

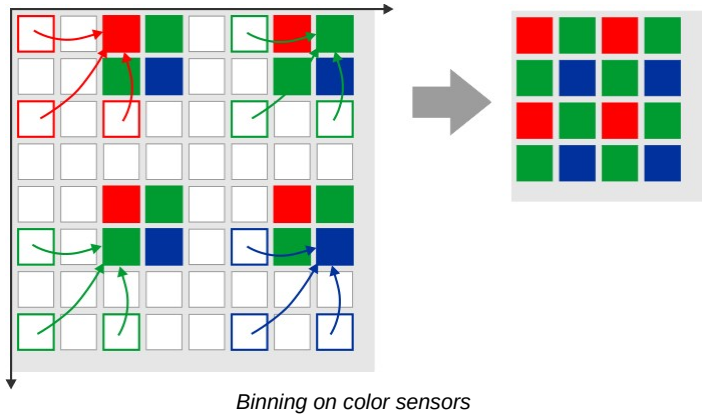
With CCD sensors, binning makes the images brighter because the pixel values are added up. With CMOS sensors, pixel values are usually averaged; this reduces image noise.

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 460 chapter lists the binning methods and factors the individual camera models support.



*Binning on monochrome sensors*



*Binning on color sensors*

# 2.5 Digitizing Images

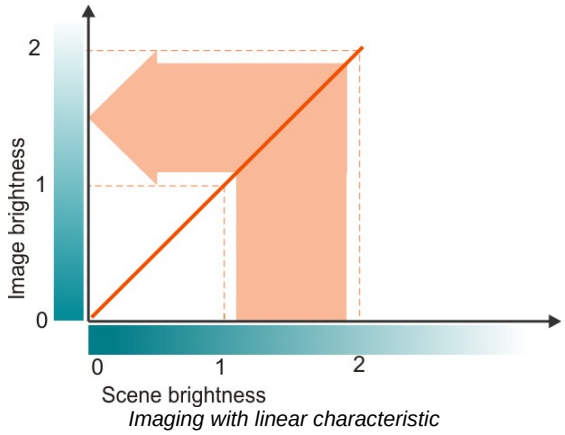


## 2.5.1 Characteristics and LUT

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:



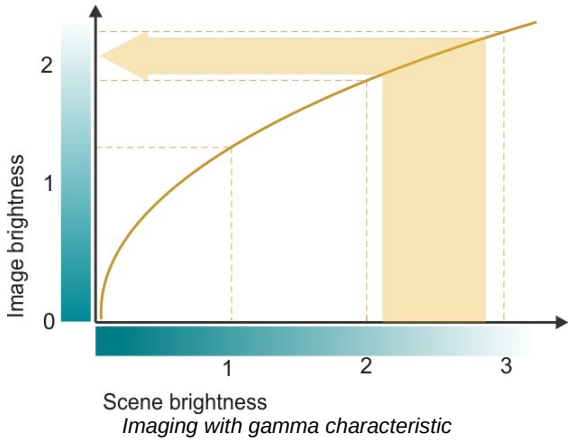

*Imaging with linear characteristic*

**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*

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*

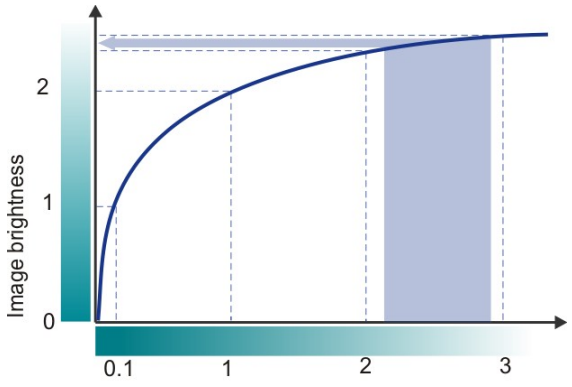
*Image with gamma characteristic*

**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.

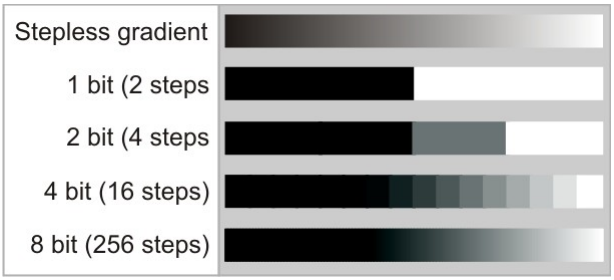
Using LUTs has the advantage that calculations can be done very fast. Typical applications of LUTs are enhancing image contrast, or gamma characteristics.


*LUT characteristic for binarizing an image*

## 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:

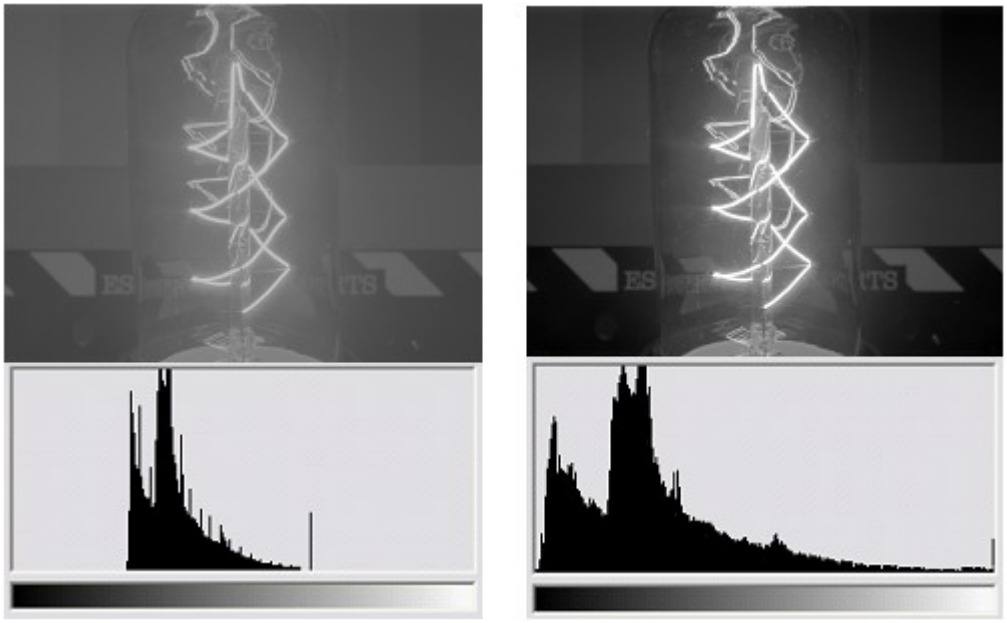


*Image capture and histogram with minimal contrast (le.) and with optimum contrast after a contrast adjustment (ri.)*

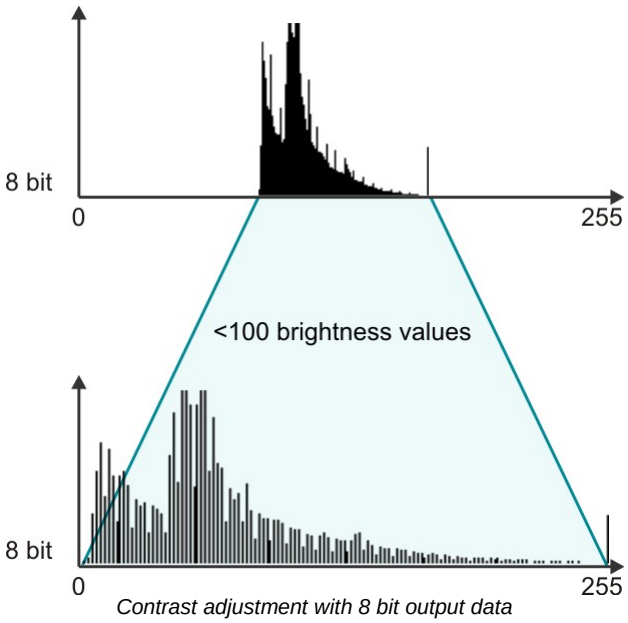
For improved display on the screen or when printed, the histogram can be spread to optimally utilize the possible brightness levels. For this purpose, the dark parts of the image are further darkened via an LUT characteristic and the light parts of the image are brightened. Thus the human eye can better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

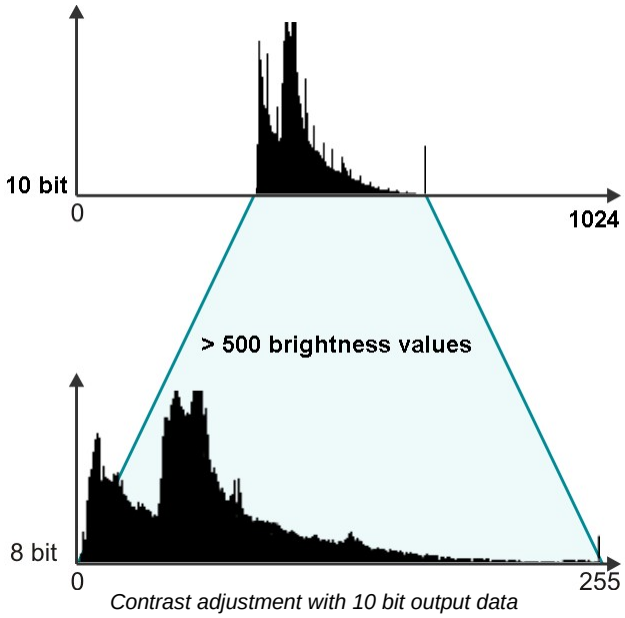



*Contrast adjustment with 8 bit output data*

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



## 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:**

- uc480 Viewer: Camera 93
- `is_PixelClock()` 294
- `is_SetFrameRate()` 329
- `is_Exposure()` 216

## 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 95 function (`is_SetGainBoost()` 331).

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 460).

**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() 303 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 uc480 Viewer: Histogram 86).

**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()`

276). 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 46). 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.

**See also:**

- `is_SetAutoParameter()` 306

### 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 459), 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 19
- uc480 Viewer: Input/output 104
- Specification: Electrical specifications 487

**Programming:**

- `is_IO()` 280

## 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 19, 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 487 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**

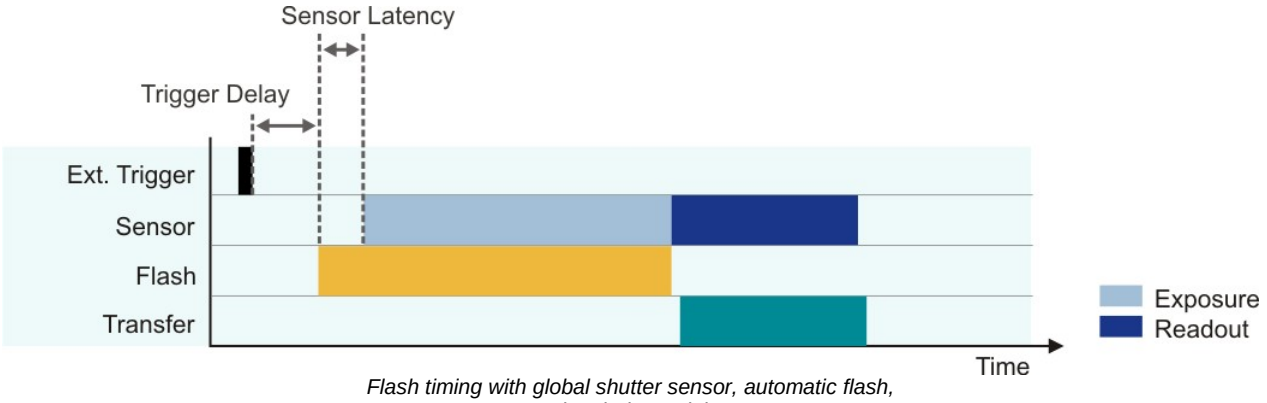
**Sensor latency and delay times** The sensor latency is due to a number of technical factors, including sensor type, image geometry, pixels clock and, with CCD sensors, the exposure time. The latency is constant for a specific combination of parameters.

Trigger delay 352, flash delay and flash duration 429 are optional and can be set by software.

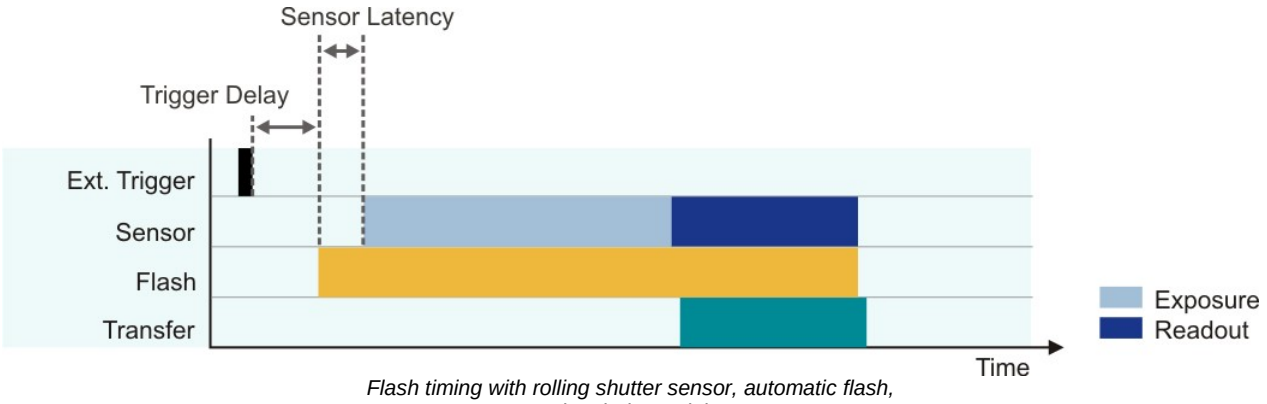
The following illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera model and the current parameter settings.

**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.




*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()` 280 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.

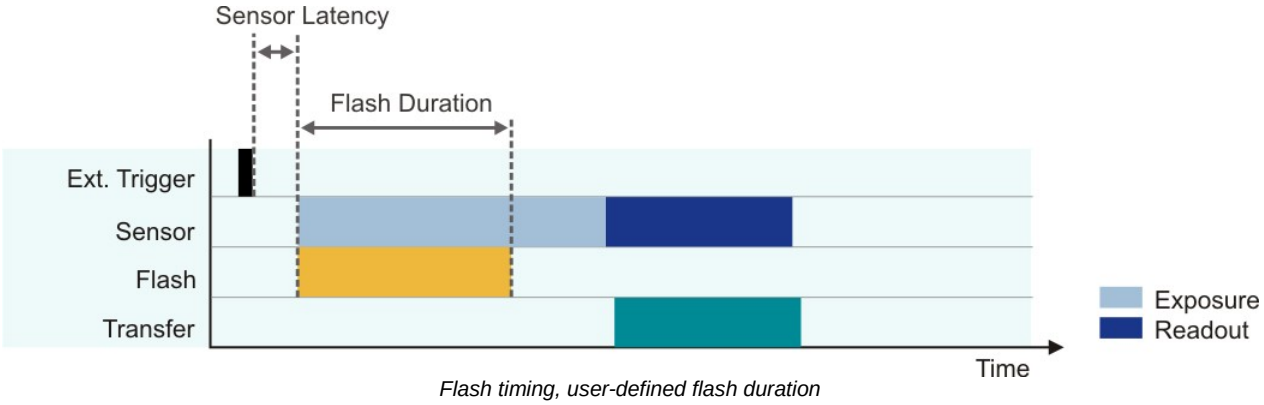
The disadvantage is that the flash signal has to be resynchronized whenever any settings for image geometry or camera timing change.

**Note**

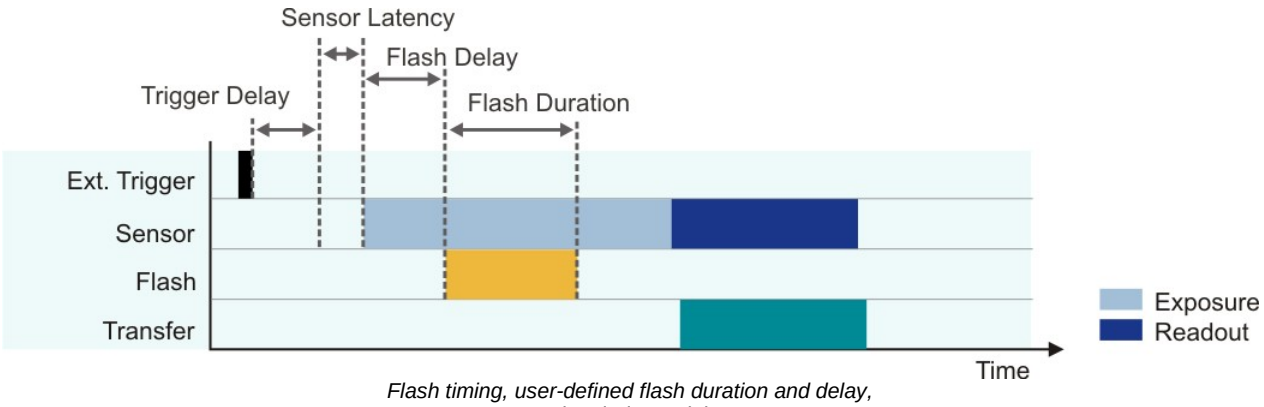
With rolling shutter sensors, you can avoid the rolling shutter effect 31 by selecting suitable delay and duration settings (global flash function). Using `is_IO()` 280, you can query the appropriate values.

**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*

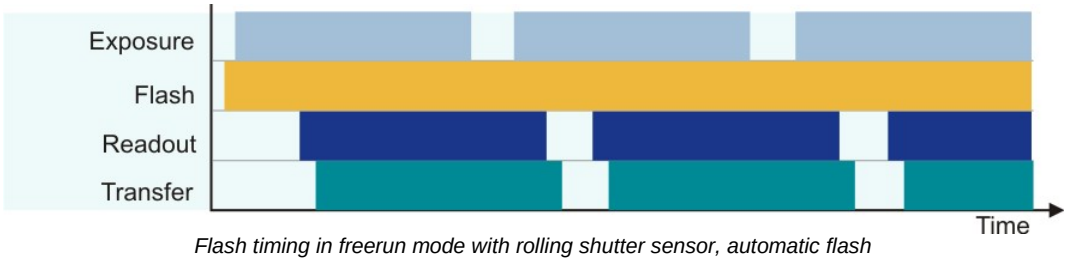



*Flash timing, user-defined flash duration and delay, optional trigger delay*

## 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).



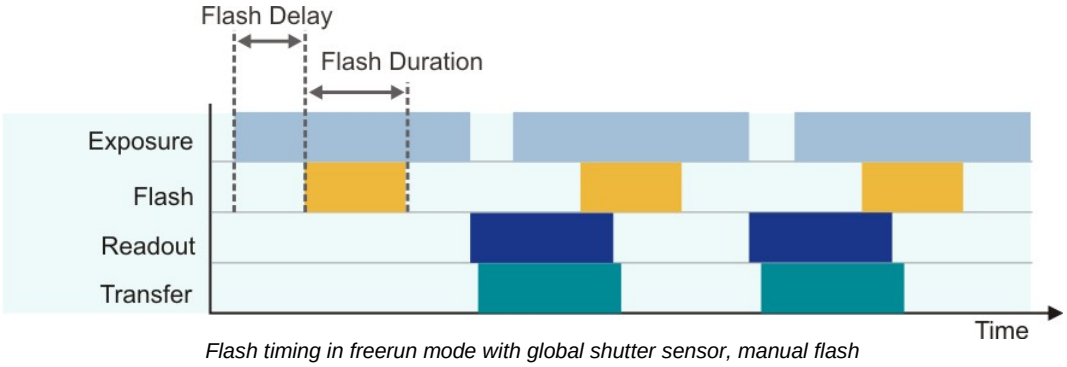

*Flash timing in freerun mode with rolling shutter sensor, automatic flash*

**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.




*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" 71 dialog box is provided in the uc480 Camera Manager. For the serial interface specifications, please refer to the DC3240x Serial Interface Wiring (RS-232) 495 chapters.

# 2.9 USB Interface



## 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.

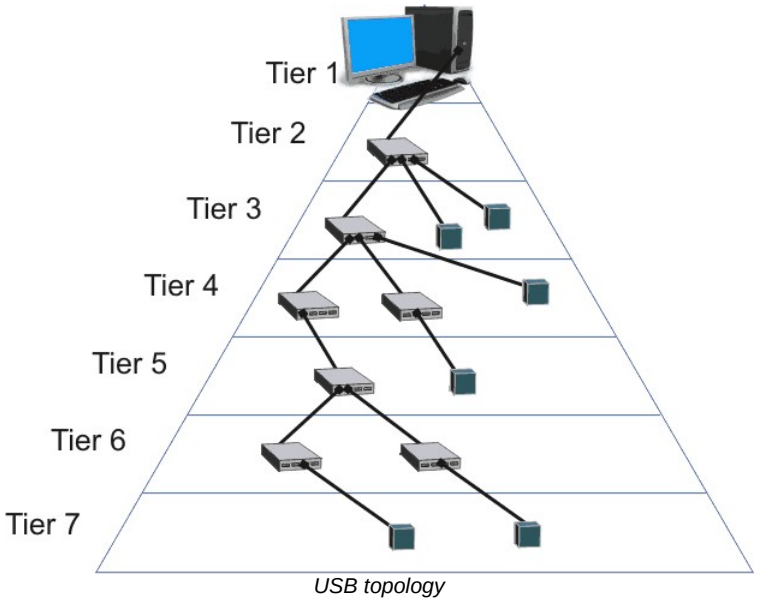
At first, the USB interface was designed to connect peripheral devices such as printers, mice, or keyboards. With the introduction of USB 2.0 in 2000, the transfer rate increased to 480 Mbit/s, making USB 2.0 suitable for connecting devices with higher data volumes (such as mass storage devices, scanners,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




*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.

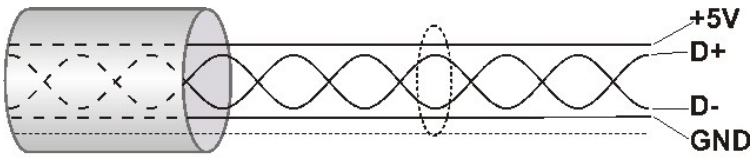
## 2.9.3 USB 2.0 Cabling and Connectors

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).

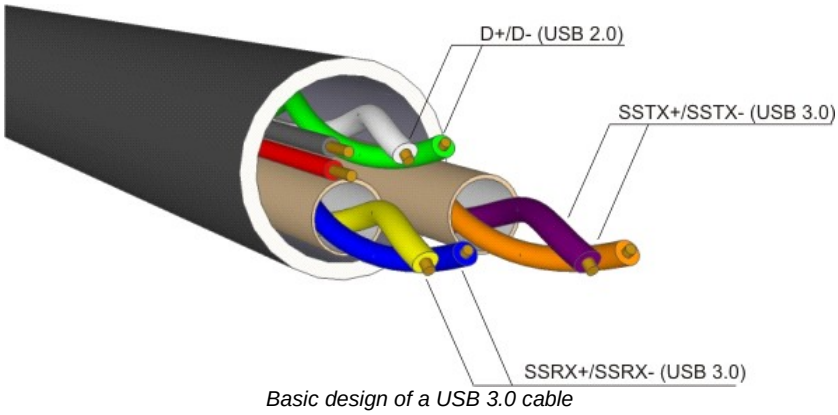
## 2.9.4 USB 3.0 Cabling and Connectors

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:

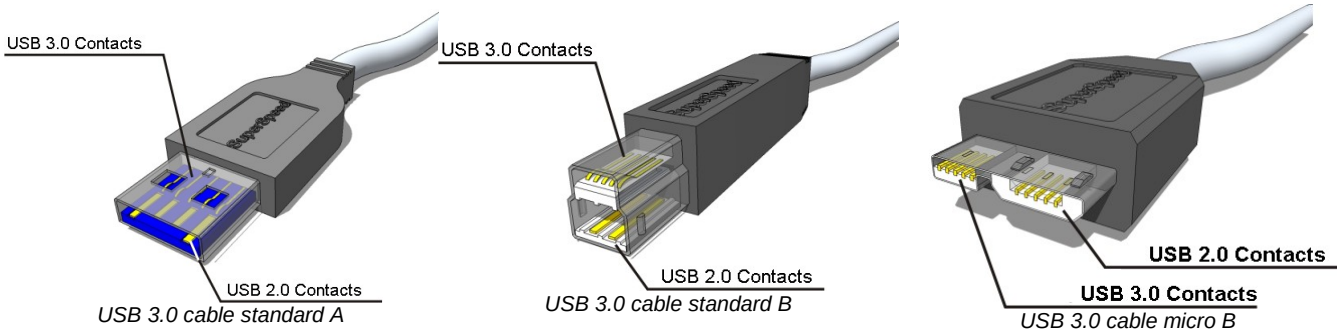
- SSTX+/-: SuperSpeed Transmit (data transfer from host to the device)
- SSRX+/-: SuperSpeed Receive (data ransfer from device to the host)
- D+/D-: data transfer (USB 2.0)
- +5 V/GND: power supply




*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.




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 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 110 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 54
- Installation and connection 56
- Installed uc480 programs
  - uc480 Camera Manager 67: The central tool for managing all connected DCx Cameras.
  - uc480 Viewer 75: A comprehensive viewer for exploring the camera functionality.
  - uc480 Player 119: A small program for playing AVI videos captured with the uc480 Viewer.
  - uc480 Hotpixel Editor 123: 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 127 chapter.

### Connect the camera

Install the latest version of uc480 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 56
- Troubleshooting 499
- Connection 61 - Status LED 501

### 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 67
- Assigning a camera ID in the camera manager 71
- Firmware and camera start 46

### 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 uc480 Viewer application. To run the application, simply double-click the corresponding icon on your Windows desktop.

When you select "uc480 > Initialize" on the menu bar, the connected DCx camera will immediately start capturing live images. The status bar at the bottom displays the frame rate and other important information.

If you are using a high-resolution camera, you can click "View > Render mode" on the menu bar to adjust the size of the rendered image to the application window.

**See also:**

- uc480 Viewer 75
- Camera basics: Operating modes 17

### Customize the key camera properties

Select "uc480 > Properties…" on the menu bar to open the dialog box for modifying the camera properties.

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. When you enable "Optimum", the optimum pixel clock will be set automatically.

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:**

- uc480 Viewer: Camera properties 92
- Camera basics: Camera parameters 44

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 "uc480 > Trigger mode…" 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:**

- uc480 Viewer: Camera properties 92
- Camera basics: Digital input/output 47
- Specifications: Electrical specifications 487

Save the camera settings and images

With uc480 Viewer, 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 using the supplied uc480 Player.

When you have made specific settings for a camera and want to save them so that you can use them again the next time you start the program – or any other uc480 program - select the "Save parameters" function to save all the camera's properties to an ini file or to the camera memory (parameter set 1/2). To load the saved settings, select the "Load parameters" option.

**See also:**

- uc480 Viewer: Record dialog 84
- uc480 Player 119

# 3.2 Installation and Connection



## 3.2.1 System Requirements

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

| | Minimum*1 | Recommended |
|---|---|---|
| CPU speed | 600 MHz | 2 x 2.4 GHz |
| Memory (RAM) | 256 MB | 2048 MByte |
| USB host controller | USB 2.0 high speed (480 Mbps)<br>USB 3.0 (4000 Mbps) for DC3240x cameras | USB 2.0 high speed (480 Mbps)<br>USB 3.0 (4000 Mbps) for DC3240x cameras<br>Intel® or NVIDIA® nForce mainboard chipset |
| Graphics card | Onboard graphics chip | AGP/PCIe graphics card<br>Latest version of Microsoft DirectX Runtime 9.0c |
| Operating system | Windows 7 32 or 64 bit<br>Windows XP 32 bit (Service Pack 2)<br><br>Linux (Kernel 2.6) | Windows 7 32 or 64 bit<br>Windows Vista 32 or 64 bit (Service Pack 1) *3<br>Windows XP 32 bit (Service Pack 3)<br>Linux (Kernel 2.6) |

*1 With the minimum system requirements the camera performance may be limited.

*3 DCC3240x USB 3.0 cameras are not supported under Windows Vista.

**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 PCs with energy saving CPU technology 500 for hints and possible solutions.

**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**

For DCx 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.

**Software Requirements under Linux**

For operating the DCx camera under Linux the following components must be installed:

| Component | Version |
|---|---|
| Linux-Kernel | 2.6.9 up to 2.6.24 |
| The standard C library `libc/glibc` | 2.0 or higher |
| GNU Compiler Collection `gcc` | 3.4 or higher |
| POSIX Thread Library<br>(POSIX Threads Enabled `libc`) | - |
| `bash` or `sh` shell<br>(for running the installation script) | - |
| Qt (for compiling the demo program) | - |

## 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 518

The LED on the back of the camera housing also indicates the USB board version (see DCx Status LED 501). In addition, the *uc480 Camera Manager* version 3.10 or higher provides information about the compatibility (see Camera Manager 67). An incompatible camera will be displayed as *free* and *not available*.

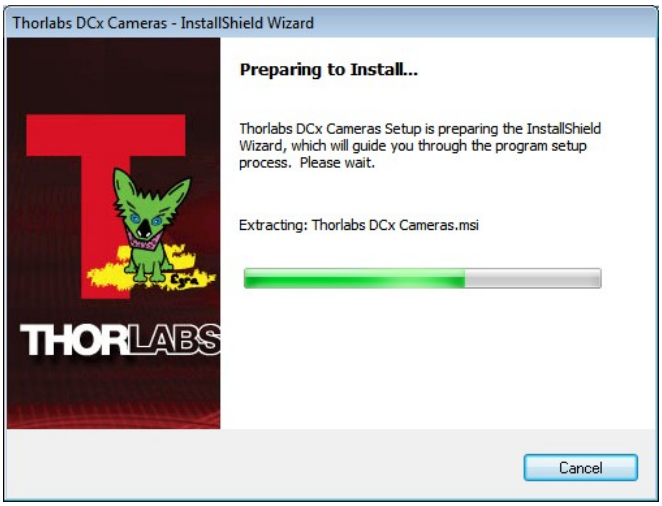
## 3.2.3 Installing the uc480 Software under Windows

**Attention**

1. You need administrator privileges to install the software.

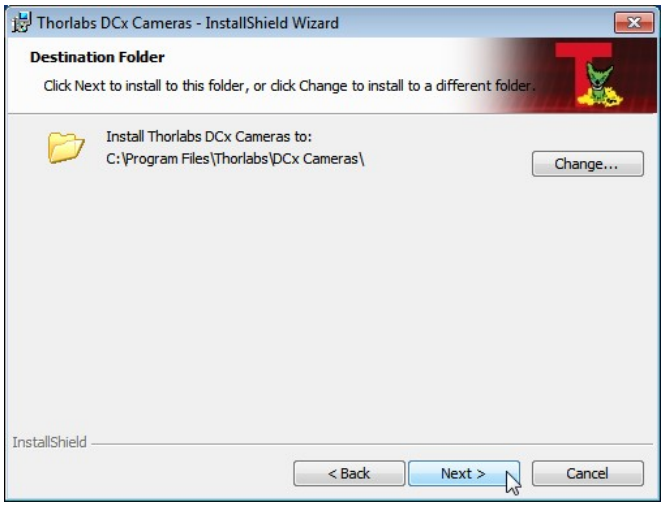
2. Please install the software prior to connect a DCx Cameras!

The software for DCx Cameras is delivered on a CD. Alternatively, or if the CD is lost, the software can be downloaded from Thorlabs' website. Please insert the delivered with the DCx Cameras CD to the drive of your PC and start the software installation:

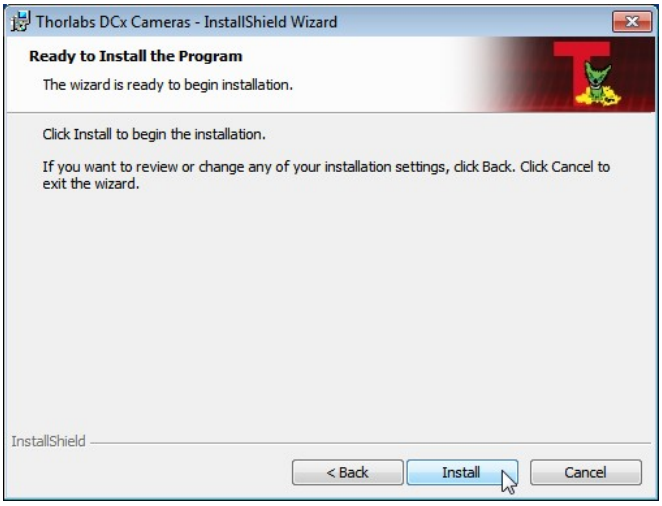




Click 'Next >' to continue.


Click 'I accept...' if you do so, then  'Next >' to continue.

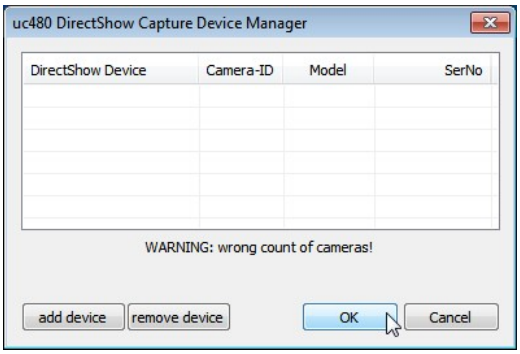



Click 'Next >' to continue.

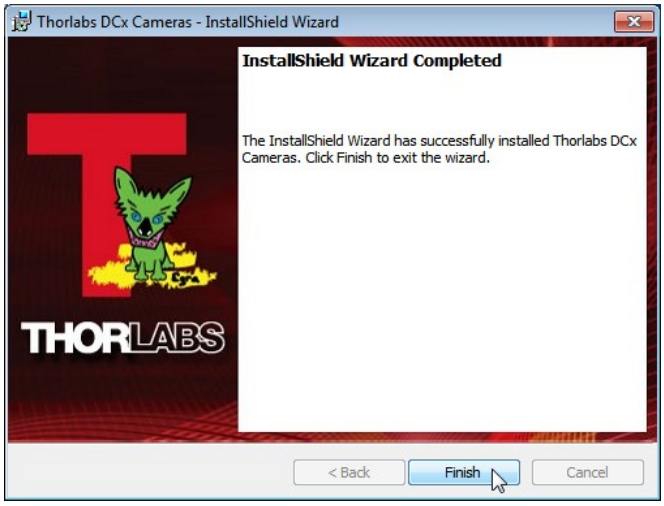



Click 'Install' to start the software installation.


Click 'Install' to allow the installation of the USB driver software. The following window appears:




This is a normal of the DirectShow driver, included in the installation package, because no camera is connected yet. Please click OK or Cancel to resume software installation.



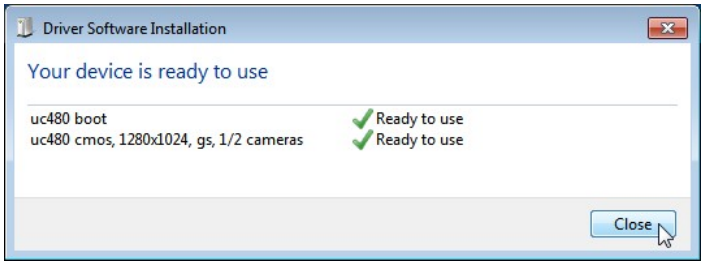

Click 'Finish'. The uc480 camera software is now installed and ready for use. For detailed description of the installed uc480 software components please see section Installed uc480 programs 67

### 3.2.4 Installing the uc480 Software under Linux

The installation of the uc480 software on Linux systems is described in the `Readme.txt` file contained in the uc480 driver download (tab "Drivers") for Linux.

### 3.2.5 Connecting a DCx Camera

Please install the software first as described in the Installing the uc480 software 58 section. 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:

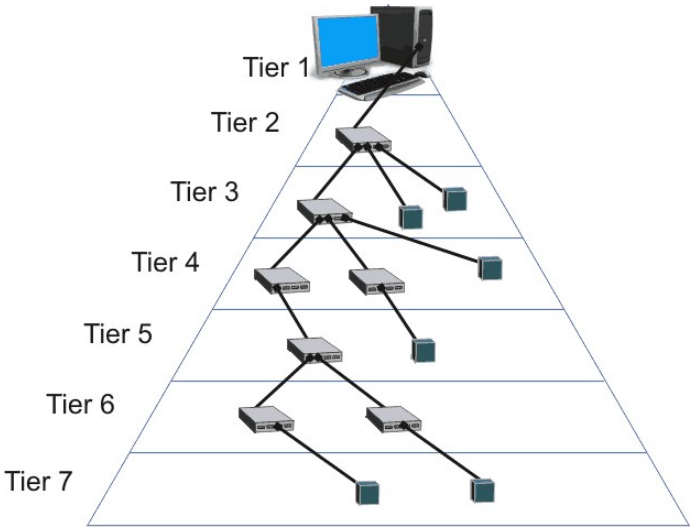



When the camera has been correctly installed, the LED on the back of the camera lights up green.

**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 67 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.

# 3.3 Application Notes by Camera Model

**Cameras with CMOS sensors**

- DCC1240x / DCC3240x Application Notes 63
- DCC1545M Application Note 66
- DCC1645C Application Notes 66

**Cameras with CCD sensors**

- DC223x Application Notes 66
- DC224x Application Notes 66

## 3.3.1 DCC1240x / DCC3240x Application Notes

For the technical specifications of this model go to: Camera and sensor data > DCC1240x / DCC3240x 461.

**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)" 115 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 33 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.

**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()` 310 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 461 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 33 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 26 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 line view 86 in the uc480 Viewer is very helpful for this.
- The master gain is disabled in 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 (see Shutter: Log mode 116) 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 464.

**Sensor**

- Sensor speed does not increase for AOI width <240 pixels.
- Extreme overexposure shifts the black level. Please deactivate the Auto offset function 95 in this 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 71 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 71 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 466.

**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 468.

**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 470.

**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 67: The central tool for managing all connected DCx cameras.
- uc480 Viewer 75: A comprehensive viewer for exploring the camera functionality.
- uc480 Player 119: A small program for playing AVI videos captured with the uc480 Viewer.
- uc480 Hotpixel Editor 123: A tool to edit the sensor hot pixel list stored in the camera.

### 3.4.1 uc480 Camera Manager

The uc480 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 uc480 Camera Manager can be accessed as follows:

- Start > Programs >uc480 Camera Manager
- Program icon on the desktop or Quick Launch toolbar
- Start > Settings > Control Panel > uc480 Camera Manager

Click in the figure to get help on the functions.


- Camera list
  The camera list 68 displays information on the connected DCx Cameras.

**Attention**

Under Windows XP (64 bit) and Windows Vista DC3240x Cameras are displayed in the uc480 Camera Manager, but they cannot be opened, because they are supported by Windows 7 and Windows XP (32 bit) only

- Control center
  In the control center 68, 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 uc480 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.


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:
  - The camera is not compatible with the driver. Please update the uc480 driver.
  - The driver has not properly detected (initialized) the camera. Please disconnect the camera from the PC and then reconnect it.
  - The camera is currently being removed from the Manager.
  - The camera reports that it is "Not operational".
- Type
  This column indicates the USB camera type.
- Cam.ID
  The camera ID 71 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.

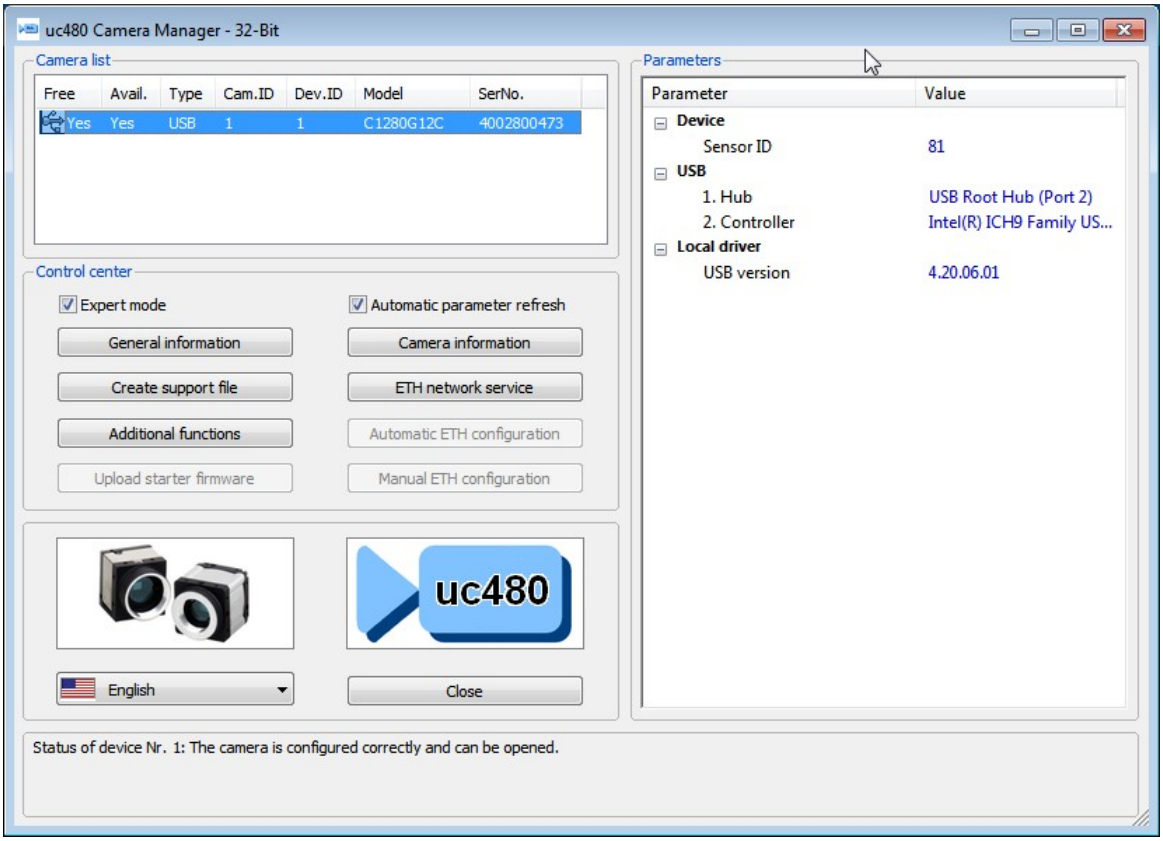
### 3.4.1.2 Control Center

- Expert mode

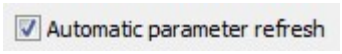
  When you select the Expert mode check box, the uc480 Camera Manager additionally displays the Parameters 74 box on the right. There you will find detailed information on the DCx camera selected in the camera list 68.

Click in the figure to get help on the functions.




- Automatic parameter refresh

  If you select the 

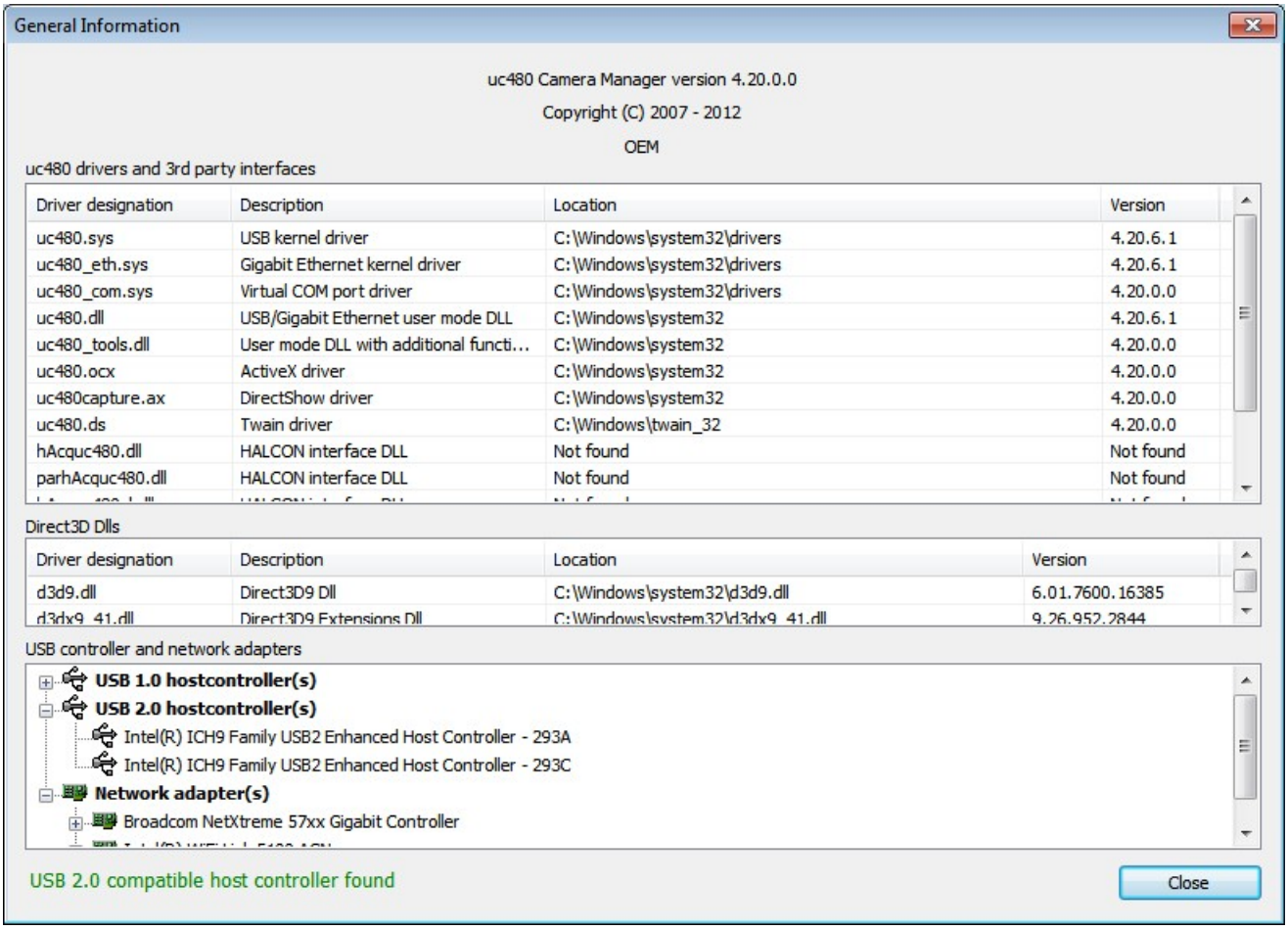
check box, the data shown in the tree structure is updated periodically. If the option is disabled, the data in the tree structure is only updated when a different camera is selected.

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

- General information 70
- Camera information 71
- Creating a support file 71
- Additional functions (COM port) 71

### 3.4.1.3 General Information

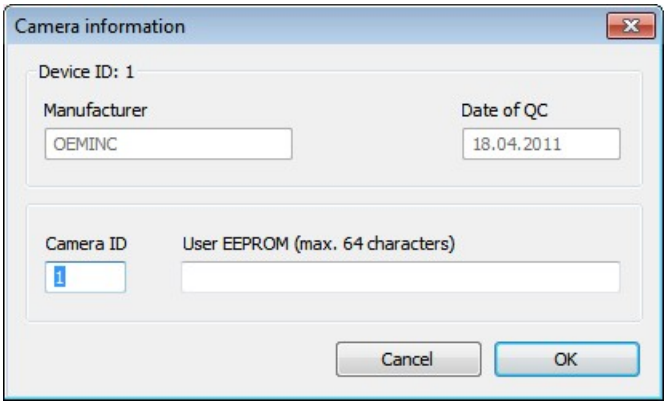
This dialog box provides information on the installed uc480 drivers and the available USB controllers and network adapters.




- 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 68).
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 518 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 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()` 183 and section troubleshooting 500).

**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()` 292 function or via the corresponding function in the uc480 Viewer.

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()` 183).

**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 518 before changing the value under "Bulk Transfer Size".

**COM Ports**

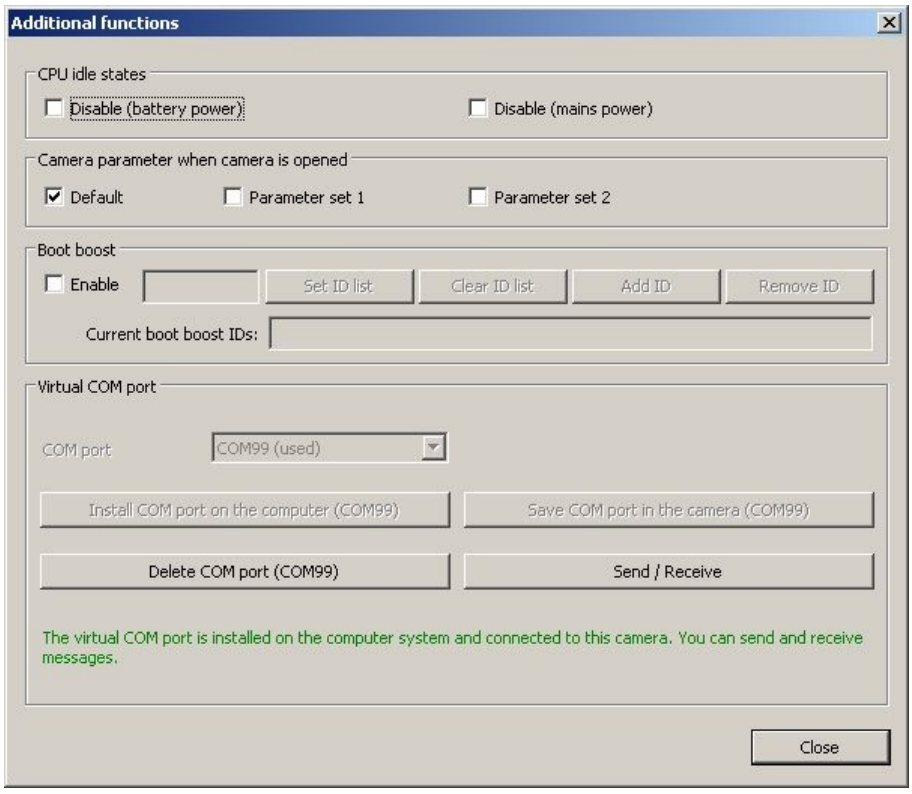
The "Additional functions" dialog box allows installing virtual COM ports for communication through the serial interface of a DCC3240x camera. The following sections show you how to set up and use the serial interface 495.

**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.

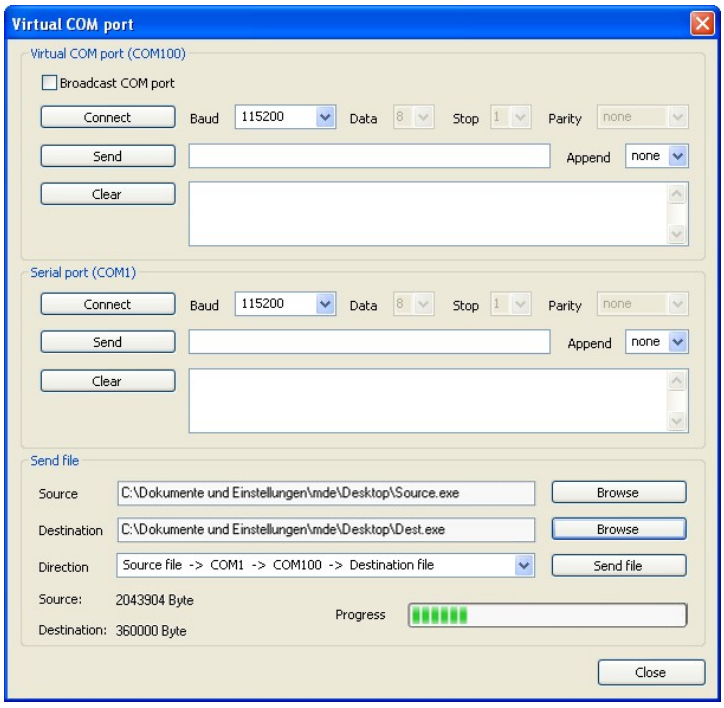
To send data via the serial interfaces of multiple cameras, you can use the broadcast port with number 255. Before connecting to the broadcast port, ensure that all the cameras that are to receive the broadcast have been opened.

**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 495 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).

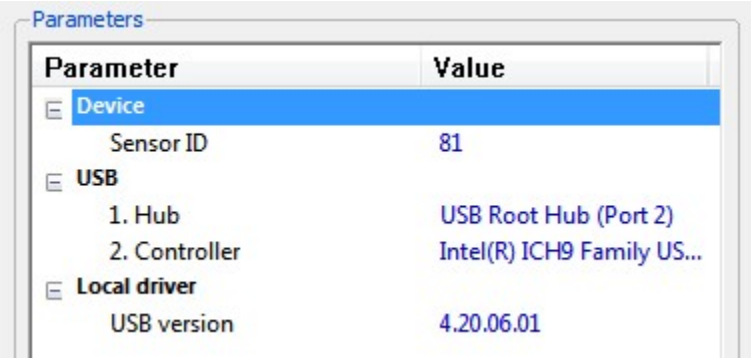
**Note**

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 68 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

### 3.4.2 uc480 Viewer

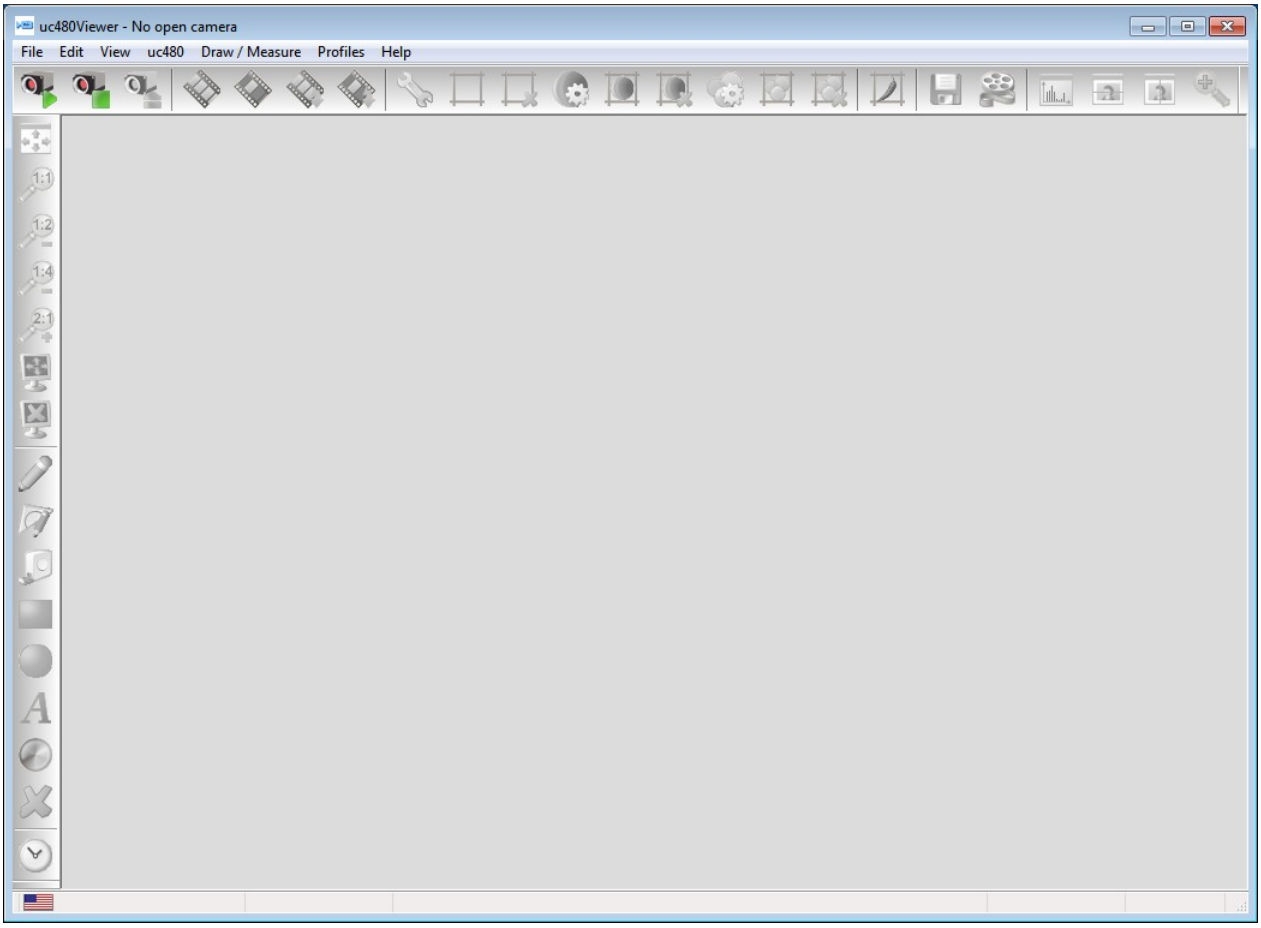
The uc480 Viewer application demonstrates the functionality and performance of the DCx Cameras. The application is part of the uc480 software package.

In uc480 Viewer, you can access all important camera settings and functions of the uc480 programming library. Apart from controlling and configuring the camera, you can record images as AVI files and save them as BMP or JPEG files.

uc480 Viewer can be accessed as follows:

- Start > All Programs > uc480 > uc480 Viewer
- Program icon on the desktop or Quick Launch toolbar

Click in the figure to get help on the functions.




**Note**

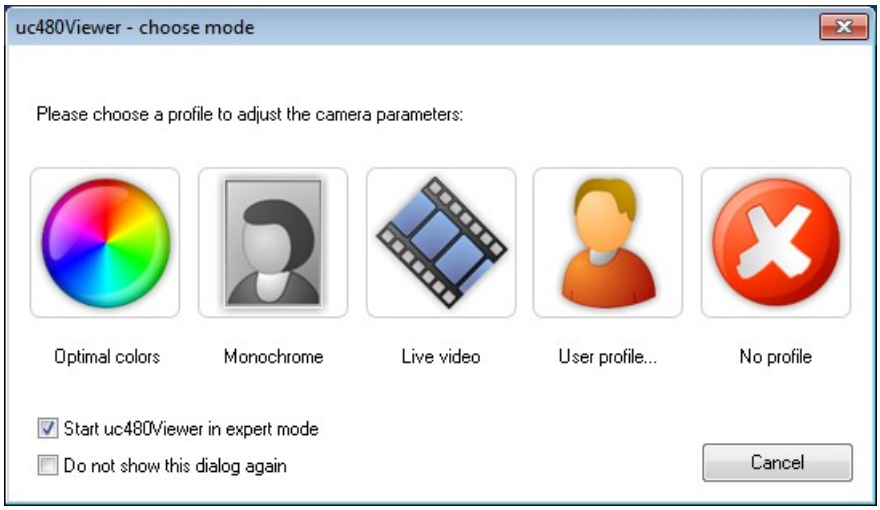
uc480 Viewer is currently only available for Windows operating systems. For Linux a version with reduced functionality of uc480 Viewer is available named **uc480 Demo.**

Please note that uc480 Viewer does not guarantee completeness and operational reliability in all modes and all computing environments. uc480 Viewer is intended solely for demonstrating the uc480 software library and camera functionality.

### 3.4.2.1 Start Dialog

When you start the application, the start dialog shown below appears by default. In this dialog you can:

- choose a profile to adjust the camera parameters
- start uc480 Viewer in expert mode




**Profiles for camera settings**

Profiles are a very quick and useful way to adjust the camera parameters to different common situations in image capture. The profiles are independent of the DCx camera model you are currently using. When you choose a profile, key camera parameters are configured for that specific situation to achieve optimum results. After installing uc480 Viewer, the following profiles are available:

- **Optimal colors**
  Choose this profile to optimize the image quality of a color camera. Parameters such as exposure, white balance and gain are set automatically. Color correction and gamma correction are enabled.
- **Monochrome**
  Choose this profile to optimize the image quality of a monochrome camera. Parameters such as exposure and gain are set automatically. Gamma correction and edge enhancement are enabled.
- **Live video**
  Choose this profile to record a smooth video. The frame rate is set to the maximum value. For high-resolution sensors, binning or subsampling is enabled to increase speed. Parameters such as exposure and gain are set automatically.
- **User profile**
  This button displays the "Open" dialog box where you can load your own profiles.
  For details on creating your own profiles, see the Creating profiles 117 chapter.
- **No profile**
  uc480 Viewer starts with the default settings configured for this camera model. No profile is used.

**Note**

A selected profile is not automatically loaded when you restart the application. You need to choose a profile again from the start dialog or Profiles menu 83.

In the File menu 80, you can choose if you want the start dialog displayed on program startup.

**Start uc480 Viewer in expert mode**

uc480 Viewer by default provides the basic icons and menus for operating the camera. If you select the "Start uc480 Viewer in expert mode" check box, additional functions, toolbars and menus are available in uc480 Viewer. This setting is recommended for advanced users.

**See also:**

- Creating profiles 117
- "Profiles" menu 83
- Customizing uc480 Viewer 85

### 3.4.2.2 Toolbars

The toolbars in uc480 Viewer provide the tools described below. Which of these tools are available depends on the mode you have selected (expert mode 76 on/off).

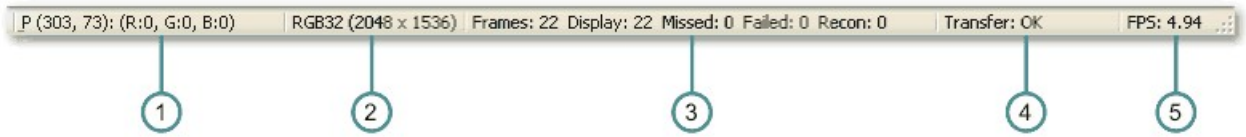
**Top toolbar**

| | |
|---|---|
|  | Open camera 79 and start in live mode |
|  | Open camera 79 |
|  | Close camera |
|  | Start/stop live video (freerun mode 17) |
|  | Snapshot in freerun mode |
|  | Start/stop continuous triggered capture 19 |
|  | Snapshot in trigger mode |
|  | Open the dialog box for setting the camera parameters 92 |
|  | Select AOI (Area of interest 34) |
|  | Delete selected AOI |
|  | Automatic brightness control 106 (AES/AGC) on/off |
|  | Set reference area for automatic brightness control |
|  | Delete reference area for automatic brightness control |
|  | Auto white balance 108 (AWB) on/off |
|  | Set reference area for auto white balance |
|  | Delete reference area for auto white balance |
|  | Measure sharpness in a defined AOI |
|  | Save image as bitmap |
|  | Open the dialog box for AVI recording 84 |
|  | Open/close Histogram 86 window |
|  | Open/close Horizontal Line View window |
|  | Open/close Vertical Line View window |
|  | Open/close Zoom 87 window |

**Left toolbar**

| | |
|---|---|
|  | Scale display to window size |
|  | Display at original size |
|  | Scale display down to half size |
|  | Scale display down to quarter size |
|  | Scale display up to double size |
|  | Show image at full screen size |
|  | Deactivate display |
|  | Draw freehand in image |
|  | Draw line in image |
|  | Measure distance in image |
|  | Draw rectangle in image |
|  | Draw circle in image |
|  | Add text to image |
|  | Choose colors for drawing functions |
|  | Clear all drawn elements |
|  | Show/hide time |

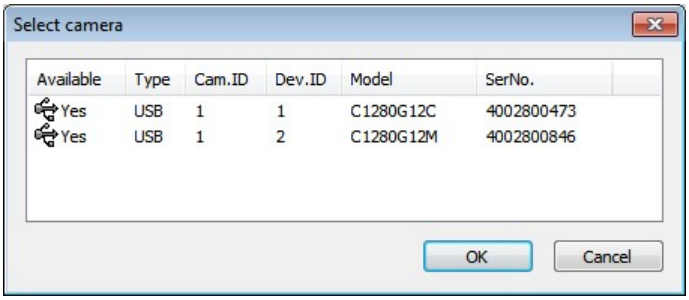
### 3.4.2.3 Status Bar



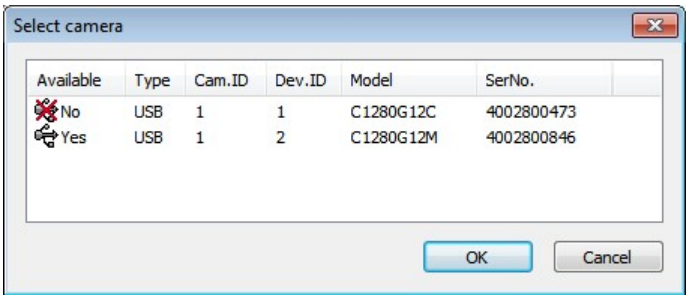

| | |
|---|---|
| 1 | Current cursor position in the Zoom/Pixel Peek window and color values at the cursor position |
| 2 | Defined color mode and image resolution |
| 3 | Counters<br>• Frames: Transferred images<br>• Display: Displayed images<br>• Missed: Hardware trigger events missed. This counter increments each time a hardware trigger is received in trigger mode, but the camera is not ready for image capture<br>• Failed: Transmission errors<br>• Recon.: This counter increments each time the open camera is removed and reconnected during operation. |
| 4 | Status of the current image data transfer (OK/Error) |
| 5 | Current frame rate (fps) of the camera |

### 3.4.2.4 Opening a Camera

Select the menu uc480 > Initialize 81 or click the corresponding icon on the Toolbar 77 to select (open) a connected camera. If only one camera is available, this camera is selected automatically. If more than one camera is connected, the "Select Camera" dialog box is displayed.




You can use multiple cameras simultaneously by opening multiple instances of uc480 Viewer. DCx cameras that have already been opened or that have not been correctly configured are marked "No" in the "Available" column.



### 3.4.2.5 Menus

Please choose a menu title:

- File 80
- Edit 80
- View 80
- uc480 81
- Draw/Measure 82
- Profiles 83
- Help 83

#### 3.4.2.5.1 File

The "File" menu contains functions for handling image, video and parameter files and for customizing uc480 Viewer.

| | |
|---|---|
| Load image ... | Load image from bitmap file (BMP) |
| Save image ... | Save image as BMP, JPEG or PNG file.<br>You can save images with a bit depth of more than 8 bit in the PNG format. |
| Record video sequence ... | Opens the Record dialog 84 box |
| Load parameters | Load parameters from an .ini file or from one of the camera's parameter sets |
| Save parameters | Save parameters to an .ini file or to one of the camera's internal parameter sets |
| Language | Select a language for the program. When you change the language, you need to restart uc480 Viewer to apply the new setting. |
| Mode | Select the uc480 Viewer user mode:<br>• Normal: The uc480 Viewer user interface shows the most important functions. Some advanced options are hidden.<br>• Expert: The uc480 Viewer user interface shows all options. See also Start dialog 76. |
| Customize | Opens the Customize 85 dialog box where you can make various settings for the startup behavior of uc480 Viewer |
| Show start dialog | If this option is selected, the profiles start dialog 76 shows on every start of uc480 Viewer. |
| Exit | Exit the demo program |

#### 3.4.2.5.2 Edit

The "Edit" menu contains the clipboard functions.

- Copy Ctrl+C: Copy the displayed image content to the Clipboard. Overlay data created using the Draw 82 function is also copied automatically.

#### 3.4.2.5.3 View

The "View" menu contains the options for setting the display mode and for opening the dialog boxes.

| | |
|---|---|
| Render mode | Image display |
| Disable | Deactivate display |
| Normal | Display at original size |
| Fit to window | Scale display to window size |
| Quarter size | Scale display down to quarter size |
| Half size | Scale display down to half size |
| Double size | Scale display up to double size |
| Mirror up/down | Mirror display on horizontal axis |
| Show only AOI | AOI is displayed without black border |

| | |
|---|---|
| Line view (horizontal) | Opens the Line view 86 window (row view) |
| Line view (vertical) | Opens the Line view 86 window (column view) |
| Histogram | Opens the Histogram 86 window |
| Zoom window | Opens the Zoom 87 window |
| Pixel peek window | Opens the Pixel peek 88 window |
| Waterfall window | Opens the Waterfall 88 window |
| Log window | Opens the Log window 89 |
| Image infos | Opens the Image infos 90 window |

### 3.4.2.5.4 uc480

The "uc480" menu contains functions for image capture and camera configuration.

| | |
|---|---|
| Initialize | Open camera 79 and show live image |
| Initialize and stop | Open camera |
| Standby | The camera changes to standby mode |
| Close | Close camera |
| Reset to defaults | Resets all values set in the demo program to the defaults |
| Start live video | Live video on/off |
| Snapshot | Snapshot from live video |
| Sync Trigger (Rising Edge) | Starts capture in freerun synchronization mode 20 on the specified edge of the trigger signal.<br>The freerun synchronization mode is currently not supported by DCx camera models. |
| Sync Trigger (Falling Edge) | |
| Single trigger | Triggered snapshot (software trigger) |
| Continuous trigger | Continuous triggered capture on/off<br>Trigger source hardware/software can be set in the corresponding property page 103 |
| Properties ... | Opens the camera properties dialog 92 |
| Auto contrast | Activate automatic brightness control 106 |
| Set new AES/AGC AOI | Define active area for automatic brightness control |
| Clear AES/AGC AOI | Clear active area defined for automatic brightness control |
| Auto whitebalance | Activate automatic white balance 108 |
| Set new AWB AOI | Define active area for automatic white balance |
| Clear AWB AOI | Clear active area defined for automatic white balance |
| Sharpness measure AOI | Draw an AOI and measure sharpness within the AOI |
| Set new AOI | After calling "Set new AOI", you can select an area of interest 97 with the mouse |
| Clear AOI | Clears the area of interest (reset to full frame) |
| User allocated image | The memory is allocated not via the SDK function `is_SetAllocatedImageMem()` 301, but by the application |
| Sound on transfer failed | Output an acoustic signal if a transmission error occurs |
| Error report | Errors are displayed in a dialog box |
| Clear counters | Reset the counters (see uc480 Viewer status bar) 79 |

### 3.4.2.5.5 Draw/Measure

The "Draw" menu contains functions drawing annotations as overlay on the live image.

**Note**

**Setting the measuring unit**

You can take simple measurements in uc480 Viewer. Before you start measuring, the camera must be calibrated to a scale. The []/pixel unit indicates the actual distance that corresponds to one pixel. To set the measuring unit, do the following:

1. Mount the camera on a stand. Place an object of known size below the camera (e.g. a ruler).
2. Make sure that the measuring unit is set to 1 at first. Then draw a dimension line over a known distance. The longer you make the dimension line, the higher the measurement accuracy will be.
3. The dimension line will show the length in pixels. Divide the actual distance by the number of pixels measured. Enter this value in the “Set measuring unit” field of the dialog box.

Example: Actual distance = 10 cm. Measured length in pixels = 200. Measuring unit = 10 cm/200 or 100 mm/200

| | |
|---|---|
| Select Color | Select color for drawings and text |
| Geometry | |
| Select line width | Set line width |
| Circle | |
| Show circles | Show/hide circles |
| New circle | Draw new circle |
| Delete circles | Delete circles |
| Freehand | |
| Show freehand lines | Show/hide freehand drawings |
| New freehand line | Draw new freehand |
| Delete freehand lines | Delete freehand drawing |
| Line | |
| Show lines | Show/hide lines |
| New line | Draw new line |
| Delete lines | Delete lines |
| Rectangle | |
| Show rectangles | Show/hide rectangles |
| New rectangle | Draw new rectangle |
| Delete rectangles | Delete rectangles |
| Measure | |
| Set measuring unit | Set scale |
| Show measures | Show/hide dimension lines |
| New measure | Draw new dimension line |
| Delete measures | Delete all dimension lines |
| Text | |
| Select font | Select font |
| Show texts | Show/hide texts |
| New text | Write new text |
| Delete text | Delete text |
| Time stamp | |
| Set time stamp position | Time stamp position<br>(top left, top right, bottom left, bottom right) |
| Show time stamp | Show/hide time stamp |

| | |
|---|---|
| Save image + drawings | Opens the "Save As" dialog box. The image is stored with all drawings, texts, measures and dimension lines |
| Load drawings | Loads saved drawings from a *.bin file. |
| Save drawings | Saves current drawings to *.bin file. |
| Delete all | Delete all drawings, texts, measures and dimension lines |

### 3.4.2.5.6 Profiles

The "Profiles" menu provides functions for managing profiles. When you start uc480 Viewer, the application searches the `C:\Program Files\Thorlabs\DCx Cameras\Profile\` folder for profile files and lists them in the Profiles menu 83. The profile names shown in the menu are named after the files.

| | |
|---|---|
| No profile | Disables the profile feature. The current settings remain unchanged. |
| Save current settings as profile | Opens the "Save as…" dialog box to save a profile (*.ucp file) |
| <Profile names> | List of profiles available in the program directory |

**Attention**

Do not remove or rename the profile files created in the `C:\Program Files\Thorlabs\DCx Cameras\Profile\` folder when you install the application. Otherwise program functions in uc480 Viewer may not work correctly.

**See also:**

- "Choose mode" start dialog 76
- Creating profiles 117

### 3.4.2.5.7 Help

| | |
|---|---|
| Hotkeys | Opens a window with a list of keyboard shortcuts available in uc480 Viewer. |
| Info about uc480 Viewer | Opens a window showing version information about uc480 Viewer and related files. |

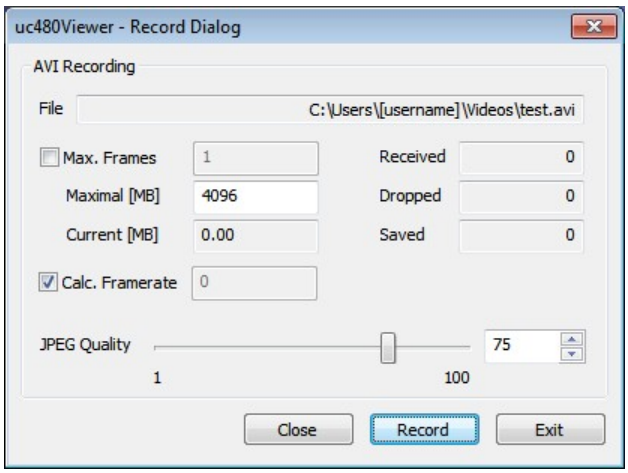
### 3.4.2.6 Dialog Boxes

Please choose a dialog box:

- Record dialog 84
- Customize 85
- Histogram window 86
- Horizontal/vertical line view 87
- Zoom window 87
- Pixel peek window 88
- Waterfall window 88
- Log window 89
- Image infos 90

#### 3.4.2.6.1 Recording a Video Sequence

Select the menu File > Record video sequence… 80 to open the uc480 record dialog box. This dialog box provides all the functions you need to create a video file (.avi) from a sequence of images captured with the DCx Camera.




**Note**

**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.

**AVI capture workflow**

Once the AVI file has been created, images transferred from the DCx Camera 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.

**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. Once the AVI file has been created, the following parameters of the input image can no longer be changed:

- Color format
- AOI, binning and subsampling
- Show only AOI

**Note**

AVI recording is only possible in the "Device Independent Bitmap (DIB)" display mode 21.

**Record dialog box**

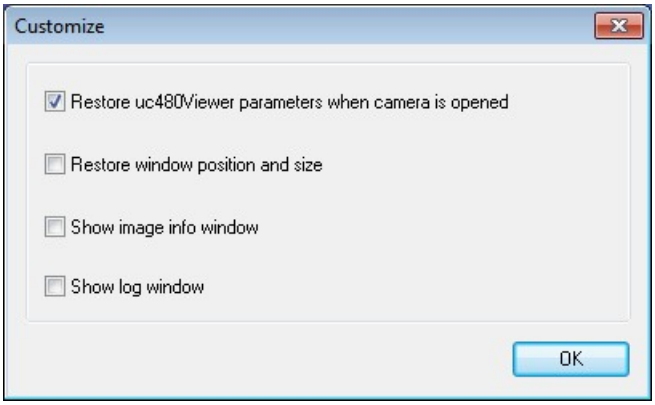
- File
  Name of the AVI file opened for recording.
- Max. Frames
  If you select this check box, you can set the number of frames after which recording should stop.
- Maximal [MB]
  Sets the maximum size for the AVI file. Recording stops when the AVI file reaches the size limit. You can edit the entry in this box (default: 1998).
- Current [MB]
  Indicates the current size of the AVI file.
- Received
  Indicates the number of images transferred by the camera.
- Dropped
  Indicates the number of images discarded during compression. A image is dropped if it arrives while compression is in progress.
- Saved
  Indicates the number of images saved to the AVI file.
- Calc. Framerate
  If you select this check box, the frame rate of the AVI file is determined automatically during recording. It can also be set manually. The frame rate value is stored in the AVI file and evaluated by the movie player. The usual value is 25 or 30 frames per second.
  The recording speed of the video depends on the selected color format, the image size and the compression level of the AVI file as well as the PC performance.
- JPEG Quality
  This slider sets the JPEG compression level (1 = lowest quality/highest compression, 100 = highest quality/ lowest compression, default = 75).
- Create.../Close
  Click this button to create a new AVI file for recording, and to close the file again when you are done. If you select an existing AVI file, the file contents will be overwritten during recording.
- Record/Stop
  Starts/stops AVI recording.
- Exit
  Closes the Record dialog box. A recording in progress is stopped.

**See also:**

- AVI function descriptions 359

### 3.4.2.6.2 Customize

Select the menu File > Customize… 80 to open the "Customize" dialog box. This dialog box provides options to define the behaviour of uc480 Viewer at program start.




The following options are available:

- Restore uc480 Viewer parameters when camera is opened
  If you select this check box, the currently set camera parameters will be restored when the same DCxCamera

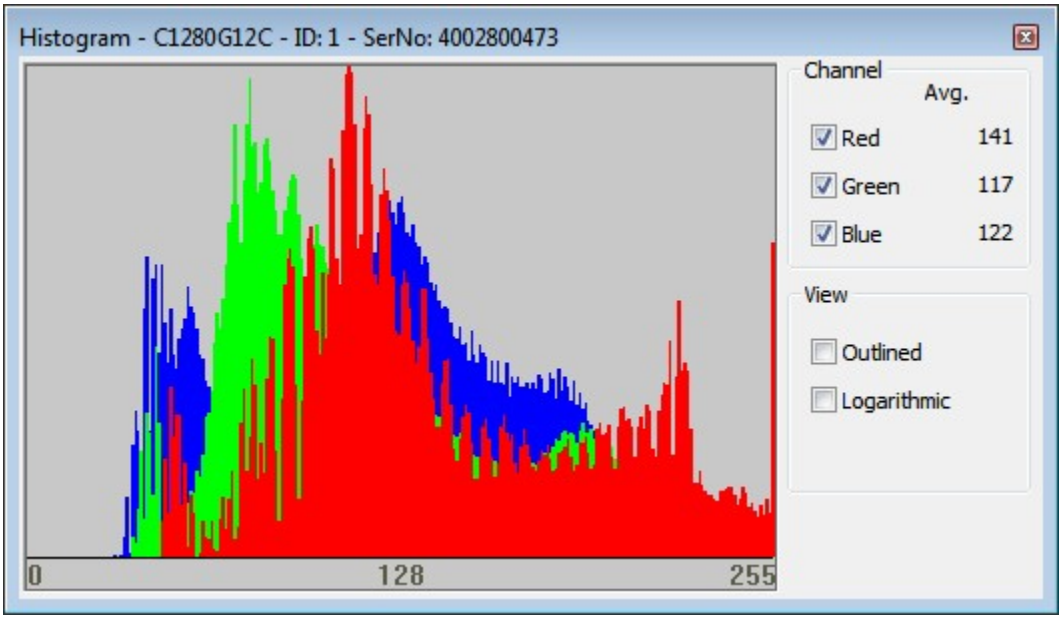
model is opened the next time.

- Restore window position and size
  If you select this check box, the size and position of the program window will be restored when uc480 Viewer is launched the next time.
- Show image info window
  If you select this check box, the Image infos 90 window will be opened on program start.
- Show log window
  If you select this check box, the Log window 89 will be opened on program start.

### 3.4.2.6.3 Histogram

Select the menu View > Histogram 80 to open the "Histogram" window. This window graphically displays the frequency distribution of the color values in the image captured by the camera.

For more information on histograms see the Bit depth and digital contrast adjustment 41 chapter.



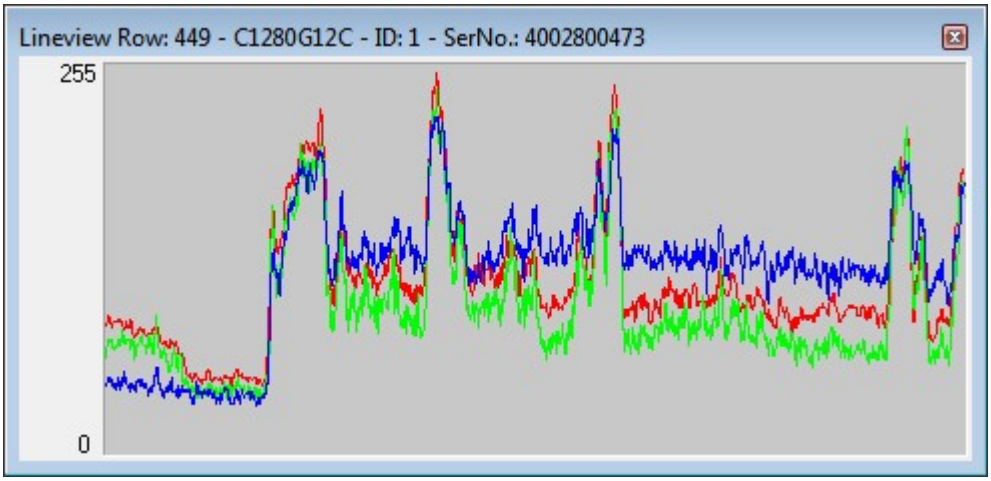

The histogram always displays 256 values per channel. For color modes with a bit depth of more than 8 bits, the function evaluates the 8 most significant bits (MSBs).

The following options are available:

- Channel
  With the "Red", "Green", and "Blue" check boxes, you can enable or disable the display for each color channel. Avg. displays the average of each color value.
  For monochrome images, the average grayscale value is displayed.
- Outlined
  The "Outlined" check box enables you to toggle the color value display between an area diagram and a line diagram.
- Logarithmic
  If you select this check box, the values are displayed with logarithmic scaling.
- Show Bayer RGB
  This function is only available when a color camera is running in "Direct raw Bayer" mode. If you select this check box, the histogram represents the individual Bayer color components as red, green and blue channels.

### 3.4.2.6.4 Horizontal / Vertical Line View

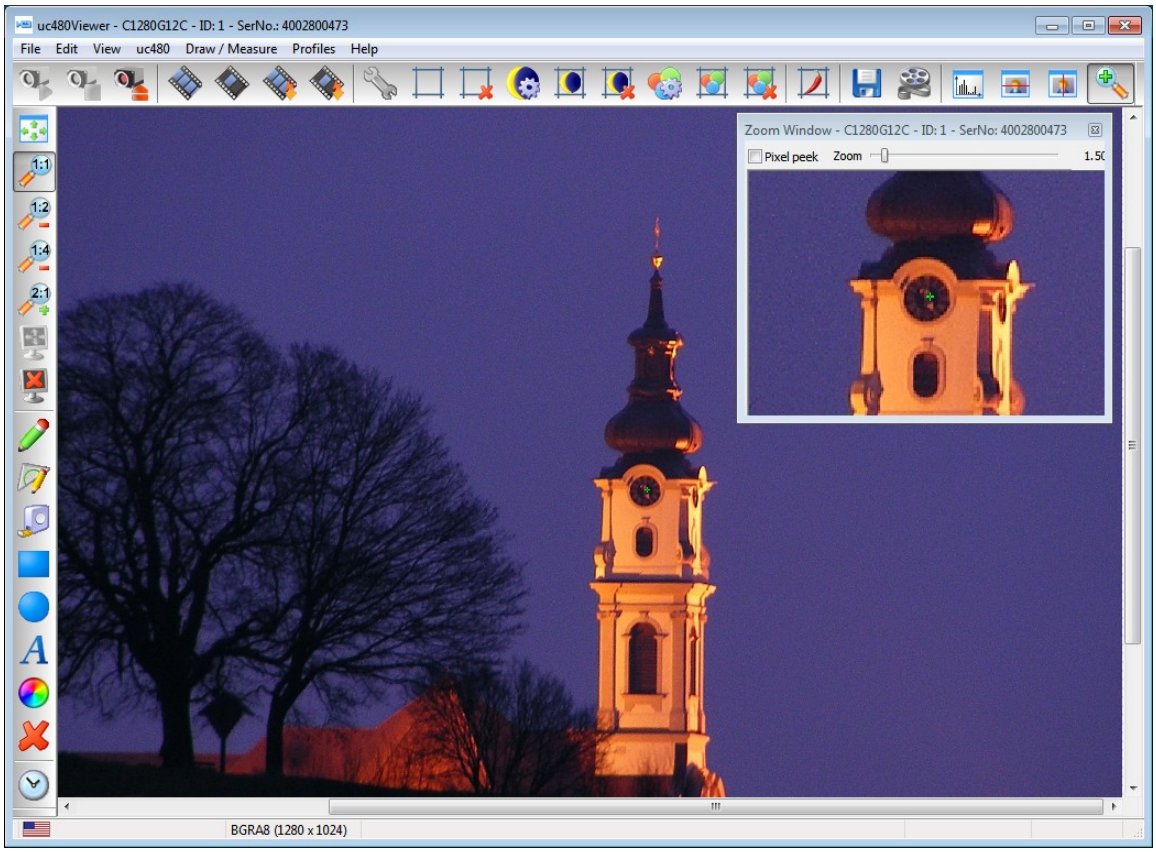
Select the menu View > Line view (horizontal/vertical) 80 to open the "Line View" windows, which show the color values of a pixel row or pixel column.




The line view always displays 256 values per channel. For color modes with a bit depth of more than 8 bits, the function evaluates the 8 most significant bits (MSBs).

### 3.4.2.6.5 Zoom

Select the menu View > Zoom window 80 to open the "Zoom" window. This window shows an enlarged view of the image area at the selected cursor position.

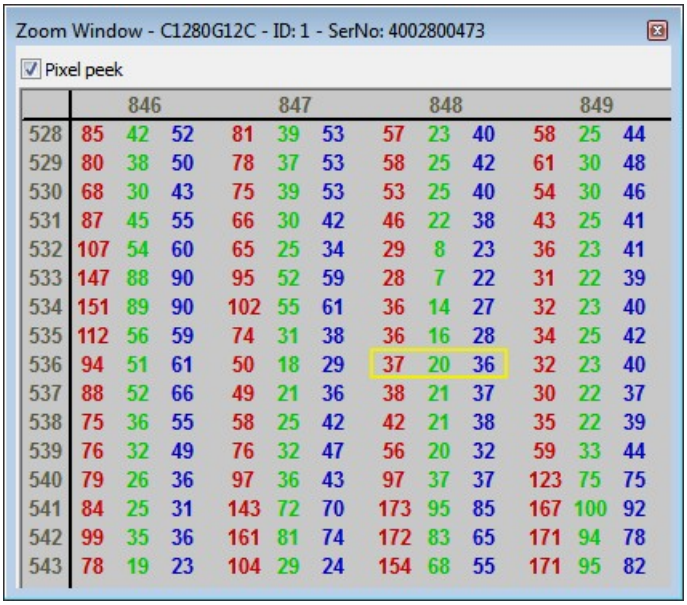



If you enable the "Pixel Peek" check box at the top of the zoom window, the color values at the cursor position are displayed (see Pixel peek window 88). Using the slider at the top of the window, you can set the zoom factor in the range between 0.25 and 20.00. The size of the image area depends on the selected size of the zoom window.

To set the cursor position you want to display in the window, place the cursor at that position in the image, hold the CTRL key and right-click. Alternatively, you can set the image position using the context menu.

### 3.4.2.6.6 Pixel Peek

Select the menu View > Pixel peek window 80 to open the pixel peek window. This window displays the color values at the selected cursor position and those of the neighboring pixels.




| | 846 | | | 847 | | | 848 | | | 849 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 528 | 85 | 42 | 52 | 81 | 39 | 53 | 57 | 23 | 40 | 58 | 25 | 44 |
| 529 | 80 | 38 | 50 | 78 | 37 | 53 | 58 | 25 | 42 | 61 | 30 | 48 |
| 530 | 68 | 30 | 43 | 75 | 39 | 53 | 53 | 25 | 40 | 54 | 30 | 46 |
| 531 | 87 | 45 | 55 | 66 | 30 | 42 | 46 | 22 | 38 | 43 | 25 | 41 |
| 532 | 107 | 54 | 60 | 65 | 25 | 34 | 29 | 8 | 23 | 36 | 23 | 41 |
| 533 | 147 | 88 | 90 | 95 | 52 | 59 | 28 | 7 | 22 | 31 | 22 | 39 |
| 534 | 151 | 89 | 90 | 102 | 55 | 61 | 36 | 14 | 27 | 32 | 23 | 40 |
| 535 | 112 | 56 | 59 | 74 | 31 | 38 | 36 | 16 | 28 | 34 | 25 | 42 |
| 536 | 94 | 51 | 61 | 50 | 18 | 29 | 37 | 20 | 36 | 32 | 23 | 40 |
| 537 | 88 | 52 | 66 | 49 | 21 | 36 | 38 | 21 | 37 | 30 | 22 | 37 |
| 538 | 75 | 36 | 55 | 58 | 25 | 42 | 42 | 21 | 38 | 35 | 22 | 39 |
| 539 | 76 | 32 | 49 | 76 | 32 | 47 | 56 | 20 | 32 | 59 | 33 | 44 |
| 540 | 79 | 26 | 36 | 97 | 36 | 43 | 97 | 37 | 37 | 123 | 75 | 75 |
| 541 | 84 | 25 | 31 | 143 | 72 | 70 | 173 | 95 | 85 | 167 | 100 | 92 |
| 542 | 99 | 35 | 36 | 161 | 81 | 74 | 172 | 83 | 65 | 171 | 94 | 78 |
| 543 | 78 | 19 | 23 | 104 | 29 | 24 | 154 | 68 | 55 | 171 | 95 | 82 |

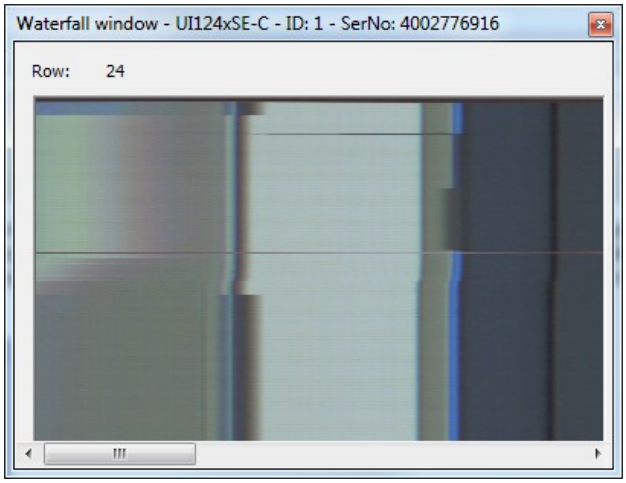
The color values at the cursor position are surrounded by a yellow rectangle. For monochrome images, the grayscale value is displayed. The window always displays 256 values per channel. For color modes with a bit depth of more than 8 bits, the function evaluates the 8 most significant bits (MSBs).

If you disable the Pixel Peek check box at the top of the window, the Zoom window is displayed (see Zoom Window 87).

To set the cursor position you want to display in the window, place the cursor at that position in the image, hold the Ctrl key and right-click. Alternatively, you can set the image position using the context menu.

### 3.4.2.6.7 Waterfall

Select the menu View > Waterfall window 80 to display the waterfall window. This window shows how a selected image line changes over time. For this purpose, the line at the selected cursor position is copied to the new window. With each new frame, all lines in the Waterfall window are moved one pixel down, and the new line is added at the top. This results in an image that flows from top to bottom and is useful for observing short-term image changes.

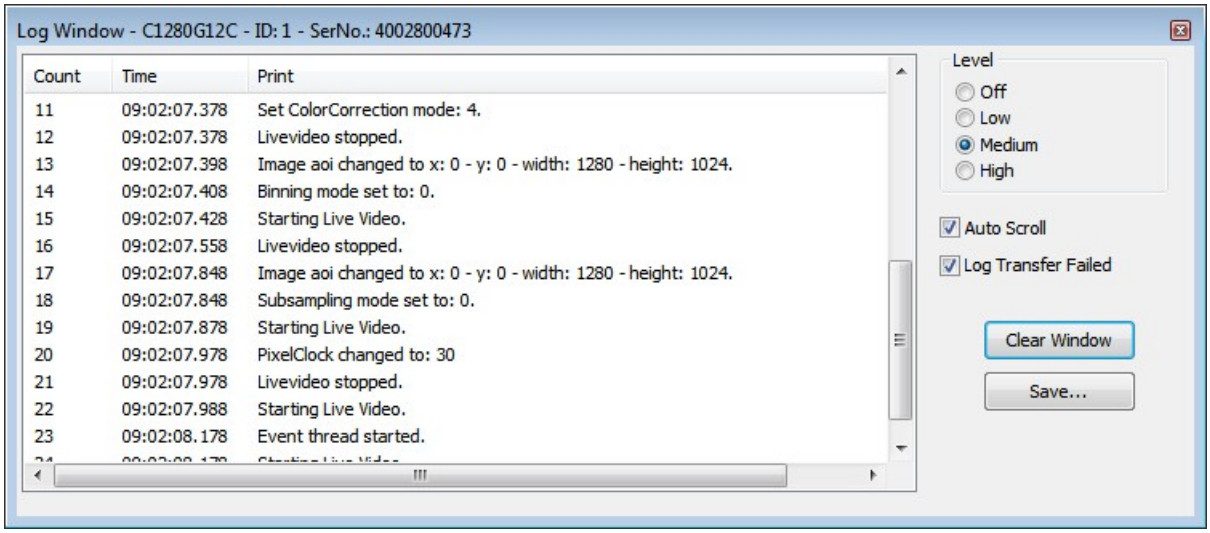



To set the cursor position you want to monitor in the Waterfall window, place the cursor at that position in the image, hold the Ctrl key and right-click. Alternatively, you can set the image position using the context menu.

### 3.4.2.6.8 Log

Select the menu View > Log window 80 to display the logged data. In this window events and messages are logged.

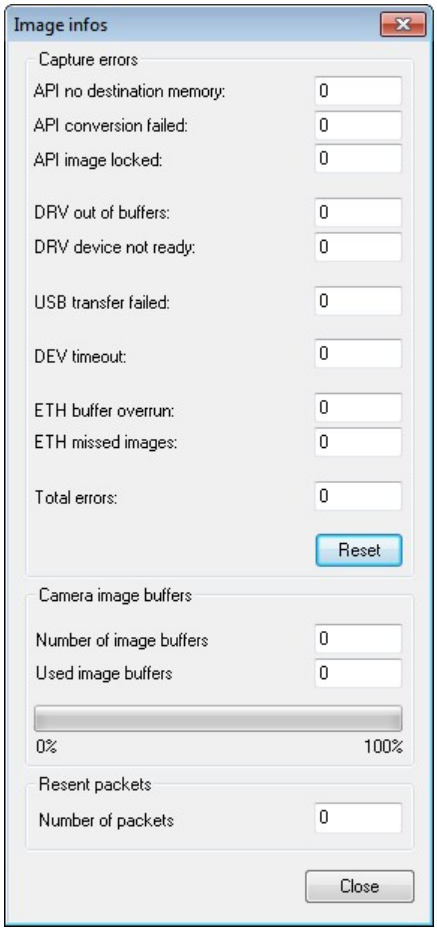



The following options are available:

- Level
  With these radio buttons, you choose the logging level (Off, Low, Medium, High). The last level used is saved when you close the window. The next time you open the log window, logging is performed at that level.
- Auto Scroll
  When you select the Auto Scroll check box, the display automatically scrolls up when new entries arrive so that the new entries can be read.
- Log Transfer Failed
  Select the Log Transfer Failed check box if you want to log transfer errors.
- Clear Window
  The Clear Window button deletes the current messages.
- Save
  The "Save" button opens the Windows "Save as" dialog box, allowing you to save the messages displayed in the log window in ASCII format (.txt file).

### 3.4.2.6.9 Image Infos

Select the menu View > Image infos 80 to display a dialog box containing information on the image capture.




**Capture errors**

This group box provides detail information on errors that occurred during an image capture process:

| Error | Description | # |
|---|---|---|
| API no destination memory | There is no destination memory for copying the finished image. | 1 |
| API conversion failed | The current image could not be processed correctly. | 2 |
| API image locked | The destination buffers are locked and could not be written to. | 3 |
| DRV out of buffers | No free internal image memory is available to the driver. The image was discarded. | 4 |
| DRV device not ready | The camera is no longer available. It is not possible to access images that have already been transferred. | 5 |
| USB transfer failed | The image was not transferred over the USB bus. | 6 |
| DEV timeout | The maximum allowable time for image capturing in the camera was exceeded. | 7 |
| ETH buffer overrun | Not applicable to DCx Cameras | 8 |
| ETH missed images | | 9 |

| # | Possible cause | Remedy |
|---|---|---|
| 1 | Not enough destination memory allocated or all destination buffers locked by the application | • Reduce the frame rate so that there is more time to process the filled destination memory |
| 2 | Internal error during internal processing of the image | - |
| 3 | All destination buffers locked by the application | • Reduce the frame rate so that there is more time to process the filled destination memory |
| 4 | 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<br>• Disable resource-intensive API image pre-processing functions (e.g. edge enhancement, color correction, choose smaller filter mask for software color conversion) |
| 5 | The camera has been disconnected or closed | - |
| 6 | Not enough free bandwidth on the USB bus for transferring the image | • Reduce the pixel clock frequency<br>• Operate fewer cameras simultaneously on a USB bus<br>• Check the quality of the USB cabling and components |
| 7 | The selected timeout value is too low for image capture | • Reduce the exposure time<br>• Increase the timeout |

**Camera image buffers, Resent packets**

Not applicable to DCx Cameras.

**See also:**

- Troubleshooting 499
- Programming: `is_GetImageInfo()` 244

### 3.4.2.7 Properties

When you select uc480 > Properties 81 from the main menu, a dialog box opens where you can set the DCx camera parameters. Changes made to camera and image settings here will take effect immediately.

When you close a camera in uc480 Viewer, the current settings are written to the Windows Registry. They will be loaded the next time you open a camera of the same type (see also the Customize 85 chapter). To save the settings to the camera or to an ini file, select File > Save parameters 80 from the main menu. To load settings, select the "Load parameters" option.

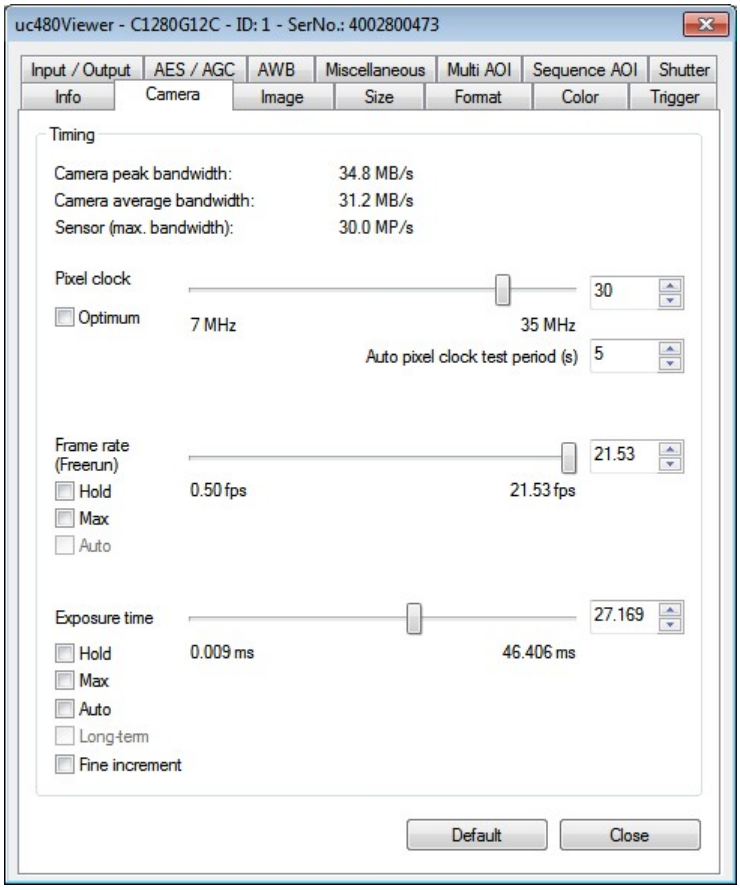
**Note**

Depending on the camera model opened and the user mode 80 set not all of the property pages might be available.

- Camera 93
- Image 95
- Size 97
- Format 99
- Color 102
- Trigger 103
- Input/output 104
- AES/AGC (automatic brightness control) 106
- AWB (auto white balance) 108
- Miscellaneous 110
- Multi AOI 112
- Sequence AOI 113
- Shutter 115

### 3.4.2.7.1 Camera

This tab provides parameters for settings the pixel clock frequency, frame rate and exposure time for your DCx camera (see also Pixel clock, frame rate and exposure time 44).




**Timing**

- Camera peak bandwidth
  Maximum required bandwidth in MByte/s (peak load).
- Camera average bandwidth
  Required average bandwidth in MByte/s. The average bandwidth is calculated from the following data: Image size, image format, frame rate, and interface-related protocol overhead.
- Sensor (max. bandwidth)
  Maximum data volume in MPixel/s created by the sensor.

  With USB 2.0 cameras, the upper limit depends on the USB chipset on the mainboard/USB card and on the number of USB devices connected. If transfer errors occur, reduce the pixel clock frequency.
- Pixel Clock
  Sets the clock rate at which the image data is read from the sensor. Changes to this parameter affect the frame rate and the exposure time.
  Many CMOS sensors allow higher pixel clock frequencies in binning/subsampling mode.
  - o Optimum
    When you select this check box, the highest possible pixel clock is determined and set automatically. The optimum pixel clock is the clock rate at which no transfer errors occur during the time (in seconds) set in the "Auto pixel clock test period" box. The longer you set the test period, the more reliable the determined pixel clock becomes. The total time it takes to automatically set the pixel clock is a bit longer than the test period setting.

    **Note**

    If the "Exposure (AES)" (automatic exposure) option is enabled in the AES/AGC 106 tab the pixel clock cannot be set manually.
- Frame rate (Freerun)
  Sets the frame rate in freerun mode. The available frame rate range depends on the pixel clock setting.

- Hold
  When you select this check box, the frame rate will remain constant if the pixel clock changes. If the frame rate cannot be maintained, it is set to the nearest possible value.
- Max
  The camera is operated at the maximum frame rate that is possible at the current pixel clock setting.
- Auto
  Select this check box to activate the auto frame rate function. This function is only available when auto exposure shutter 106 is enabled.

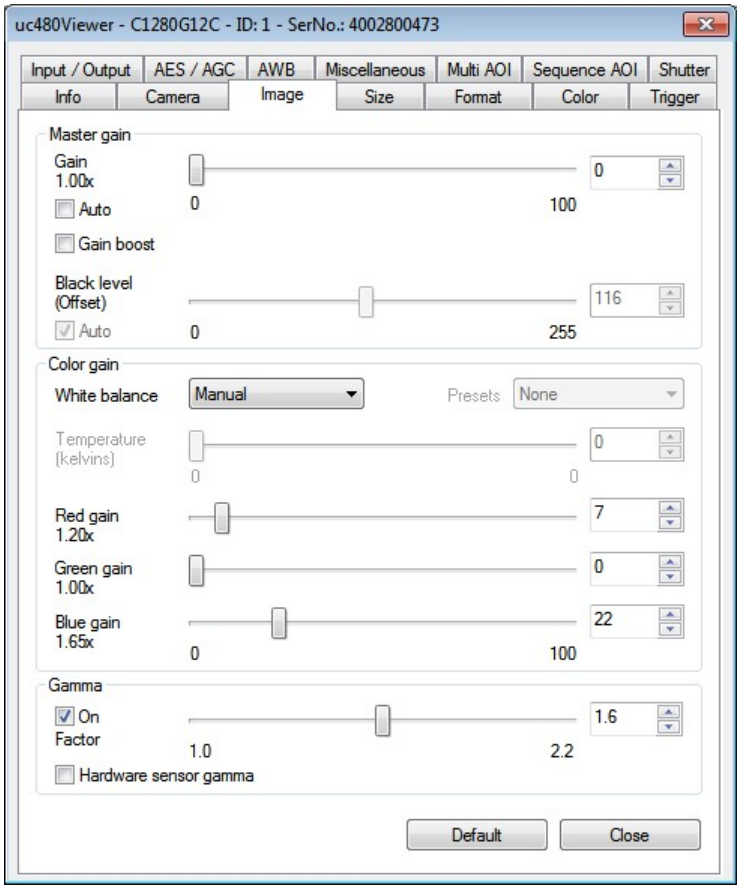
- Exposure time
  Sets the exposure time. The available exposure time range depends on the pixel clock setting and the frame rate. A low frame rate setting allows long exposure times. A high frame rate setting reduces the maximum possible exposure time. In combination with flash an exposure time of 1 ms has been proven.
  - Hold
    When you select this check box, the exposure time will remain constant if the frame rate changes. If the exposure time cannot be maintained, it is set to the nearest possible value.
  - Max
    The camera is operated at the maximum exposure time that is possible at the current frame rate.
  - Auto
    Select this check box to activate the auto exposure shutter 106 function. If the "Auto" check box is selected, the exposure time and pixel clock can no longer be adjusted manually. Selecting the "Hold" or "Max" check box deselects the "Auto" check box.
  - Long-term
    If you select this check box, you can set an exposure time of up to 10 minutes on CCD cameras.
  - Fine increment
    Select this check box to set the exposure time in fine increments. This option is not supported by all camera models.

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.2 Image

On this tab you can set the sensor gain parameters for your DCx camera (see also Gain and offset 44).




#### Master gain

The following functions control the analog image signal gain and the black level. The analog adjustments are made directly in the sensor, which achieves better results than image adjustments via software.

- Gain [0 ... 100]
  Gain for overall image brightness. Some camera models have no master gain.
  Master gain = 100 means maximum gain; the actual factor is displayed. A gain factor of 1x disables master gain. The maximum possible gain factor depends on the model you are using.
  - o Auto
    Select this check box to activate the automatic gain control 106 function. Manually changing the master gain setting disables the "Auto" function.
  - o Gain boost
    Additional analog camera hardware gain. The gain factor ranges between 1.5x and 2x, depending on the camera model.
- Black level (offset) [0 ... 255]
  Offset for the black level of the sensor signal. The sensor adjusts the black level of the pixels automatically by default. If the environment is very bright, it can be necessary to adjust the black level manually. High gain may offset the black level. Only an additive offset is possible (increase of the black level).
  - o Auto
    The black level is automatically corrected by the sensor (recommended).

**Note**

With cameras featuring both master gain and RGB gain, the two gain factors are multiplied. Very high gain values can be achieved in this way. If you want to use the RGB sliders for color adjustment, we recommend setting green gain to 0 and using only red and blue gain.

#### Color gain

You can fix a setting (in kelvins) for the color temperature 180 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.

- White balance
  - Manual
    Set the RGB gains manually.
  - Auto
    Enable the AWB (auto white balance) 108 function.
  - Default gains
    Default gain values for this camera model.
  - sRGB D50
    sRGB (standard RGB) color space with a white point of 5000 kelvins (warm light)
  - sRGB D65
    sRGB (standard RGB) color space with a white point of 6500 kelvins (mid daylight)
  - CIE RGB E
    CIE-RGB color space with standard illumination E
  - ECI RGB D50
    ECI-RGB color space with a white point of 5000 kelvins (warm light)
  - 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.
- Presets
  This selection box offers several presets for common illumination types.
- Temperature (Kelvin)
  Here, you can set a color temperature value directly.

For color cameras, the following functions control the analog gain for the RGB color channels. The analog control is performed in the sensor and achieve better results than software based image correction.

- Red gain [0 ... 100]
  Amplifies the red color values
- Green gain [0 ... 100]
  Amplifies the green color values
- Blue gain [0 ... 100]
  Amplifies the blue color values

**Gamma**

This function activates the gamma function and sets the factor for the gamma curve. The gamma function emphasizes dark image areas according to a non-linear curve.

- Hardware sensor gamma
  This option enables gamma correction by the hardware, using a fixed factor.

**Default**

Click this button to reset all parameters to the model-specific defaults.

**See also:**

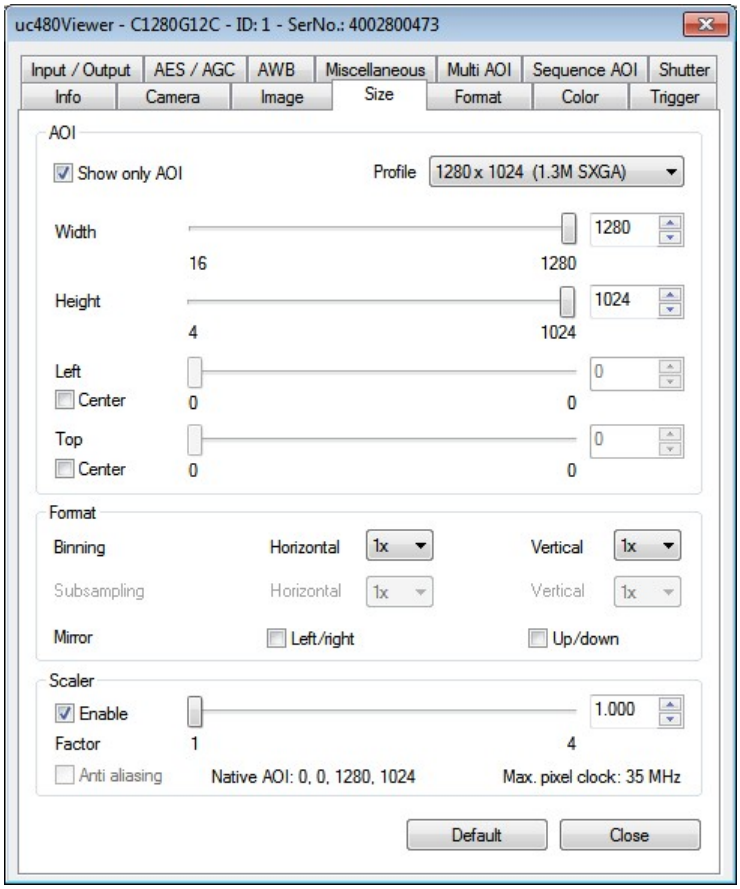
- Characteristics and LUT: Gamma 39

**Programming:**

- Color Temperature: `is_ColorTemperature()` 180

### 3.4.2.7.3 Size

On this tab, you can set the image size parameters for your DCx camera (see also Reading out partial images 34).




**AOI**

These parameters allow selecting the size and position of the area of interest 34.

- Profile
  With this selection box, you can set several common image formats on CMOS sensors. The driver then sets the selected format using either AOI, binning / subsampling, or scaler, so that it achieves the best image quality.
- Show only AOI
  The AOI is displayed without a black border. Internally, the image is managed at the AOI resolution and not the full sensor resolution. This function saves memory and computing time when rendering the image.
  - o Width
    Sets the AOI width.
  - o Height
    Sets the AOI height.
  - o Left
    Sets the left-hand position of the AOI.
    - Center
      Select this check box to center the AOI horizontally.
  - o Top
    Sets the top position of the AOI.
    - Center
      Select this check box to center the AOI vertically.

**Horizontal format / vertical format**

With these options, you can select the binning 38 and subsampling 37 settings for the image.

- Binning Horizontal/Vertical
  These radio buttons allow setting the binning factor. 460 The image resolution is then reduced by the selected factor. You can use binning to increase the frame rate.

- Subsampling Horizontal/Vertical
  These radio buttons allow setting the subsampling factor. The image resolution is then reduced by the selected factor. You can use subsampling to increase the frame rate.
- Mirror Left/right | Up/down
  Select this check box to flip the image horizontally/vertically.

**Scaler**

With these options, you can configure the image scaling settings provided by specific DCxCamera sensors. These options are not available for all camera models.

- Enable
  Enables the sensor's image scaling feature.
- Factor
  With this slider, you can set the scaling factor. 460
- Anti aliasing
  The anti aliasing function smoothes the image edges during scaling.
- Native AOI
  Shows the native image size without the effect of the scaler.
- Max. pixel clock
  Shows the maximum possible pixel clock with scaler.

**Note**

**Binning and subsampling in color and monochrome cameras**: Some color cameras perform only mono binning/subsampling due to the sensors they use. If mono binning or subsampling is used in a color camera, the color information will be lost.

Some monochrome cameras perform only color binning/subsampling due to the sensors they use. If color binning or subsampling is used in a monochrome camera, image artifacts might become visible.

**Default**

Click this button to reset all parameters to the model-specific defaults.

**See also:**

- Basics: Reading out partial images 34

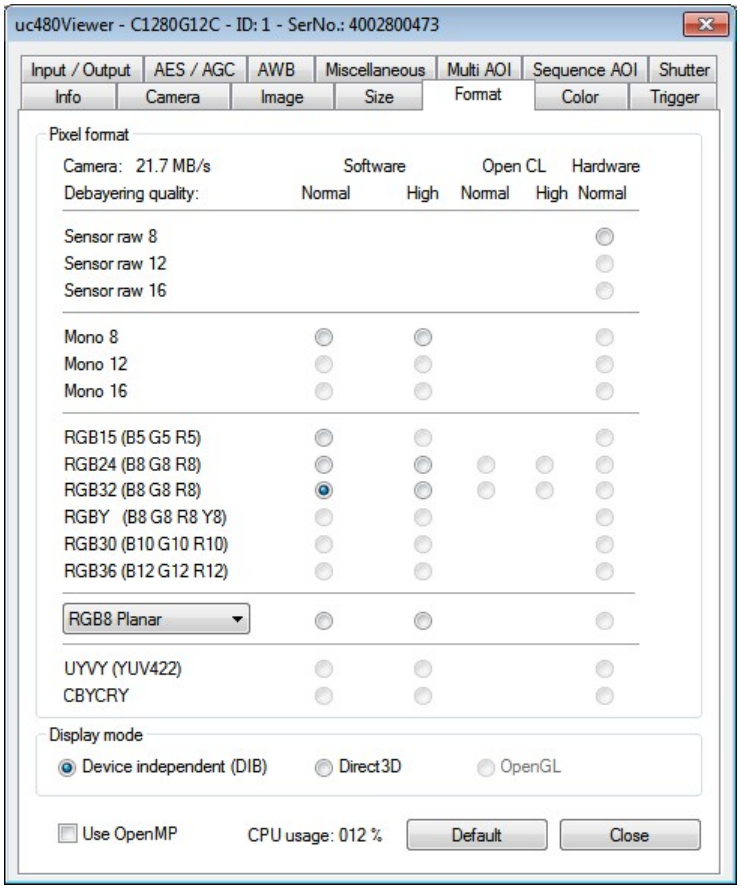
**Programming:**

- Area of interest: `is_AOI()` 159
- Binning: `is_SetBinning()` 310
- Subsampling: `is_SetSubSampling()` 347
- Profiles/image formats: `is_ImageFormat()` 267

### 3.4.2.7.4 Format

On this tab you can set parameters for the color format and display mode of your DCx camera.




**Pixel format**

With these settings you define the target format to which you want to convert the sensor's raw data (raw Bayer). The Color formats 502 chapter provides a detailed list of all DCxCamera color formats and their representation in the memory.

- Debayering quality
  With this setting you select the conversion algorithm.
  - Software
    - Normal
      The conversion is performed by software in the PC. The standard filter mask is used for the conversion.
    - High
      The conversion is performed by software in the PC. A large filter mask is used for the conversion.
  - Open CL
    - Normal
      The conversion is performed on the graphic card's processing unit. The standard filter mask is used for the conversion.
    - High
      The conversion is performed on the graphic card's processing unit. A large filter mask is used for the conversion.
  - Hardware Normal
    The conversion is performed in the camera. The standard filter mask is used for the conversion. This radio button is not available for DCx cameras.

**Note**

- Color conversion with Open CL is not supported by all graphic cards.
- Open CL is not supported by mono cameras.

With the format radio buttons you specify the format in which the image data are written to the memory. The following formats are available:

- Sensor raw 8
  Direct output of the sensor's raw data (8 bits per pixel). If you are using a color camera, the pattern of the Bayer color filter is visible. With monochrome cameras, raw Bayer corresponds to the grayscale format with the exception of the LUT/gamma curves.
- Sensor raw 12
  Direct output of the sensor's raw data (12 bits per pixel, starting from the least significant bit (LSB)).
- Sensor raw 16
  Direct output of the sensor's raw data (12 bits per pixel, starting from the most significant bit (MSB)).
- Mono 8
  Output of a grayscale image to which the LUT/gamma curve has been applied (8 bits per pixel).
- Mono 12
  Output of a grayscale image to which the LUT/gamma curve has been applied (12 bits per pixel, starting from least significant bit (LSB)).
- Mono 16
  Output of a grayscale image to which the LUT/gamma curve has been applied (12 bits per pixel, starting from most significant bit (MSB)).
- RGB5 (5 5 5)
  Output of an image converted according to RGB 15 (5 bits per pixel for R, G and B)
- RGB565 (5 6 5)
  Output of an image converted according to RGB 16 (5 bits per pixel for R and G, 6 bits per pixel for B)
- BGR8 (8 8 8)
  Output of an image converted according to RGB 24 (8 bits per pixel for R, G and B)
- BGRA8 (8 8 8)
  Output of an image converted according to RGB 32 (8 bits per pixel for R, G and B, 8 bit padding)
- BGRY8 (8 8 8 8)
  Output of an image converted according to RGB 24 (8 bits per pixel for R, G and B) and an additional gray channel (8 bits per pixel)
- BGR10 (10 10 10)
  Output of an image converted according to RGB 30 (10 bits per pixel for R, G and B, 2 bit padding (MSB = 0))
- BGR12 (12 12 12)
  Output an image converted according to RGB 36 (12 bit per pixel for R, G and B, 4 bit padding)
- BGRA12 (12 12 12)
  Output an image converted according to RGB 48 (12 bit per pixel for R, G, B and alpha channel, 4 bit padding)
- RGB8 Planar
  Output an image converted according to planar RGB 8.
- RGB8 Planar (red)
  Output the red color component of an image converted to planar RGB 8
- RGB8 Planar (green)
  Output the green color component of an image converted to planar RGB 8
- RGB8 Planar (blue)
  Output the blue color component of an image converted to planar RGB 8
- UYVY (YUV422)
  Output of an image converted according to YUV (8 bits per pixel for U, Y, V and Y)
- CBYCRY
  Output of an image converted according to YUV (8 bits per pixel for Cb, Y, Cr and Y)

**Note**

We recommend 32 bit RGB mode for TrueColor applications. Y8 mode is usually used for monochrome applications.

**Note**

Images with a bit depths of more than 8 bits can be saved in the PNG format. 12 bit formats are converted in 16 bits.

**Display mode**

With these radio buttons you select the display mode for the image.

- Device independent (DIB)
  The processor actively renders the image. This color format is supported by all graphics hardware and is recommended for applications that will be used on different PCs.
- Direct3D (Desktop color)
  In this mode the images are written directly to an invisible area of the graphics card, mixed with optional overlay image data and displayed by the card without load on the CPU. The mode also allows scaling the images in real time.

  **Note**

  The display mode Direct3D is only supported by graphics cards with DirectX functionality.
- OpenGL
  OpenGL is a cross-platform and cross-language programming interface for developing 2D and 3D computer graphics. Similar to Direct3D the capabilities of the graphics card are used to increase the performance.

**Open MP**

- Use OpenMP
  OpenMP (Open Multi-Processing) 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 26, are distributed across several processor cores to accelerate execution. The use of OpenMP, however, increases CPU load.

**Default**

Click this button to reset all parameters to the model-specific defaults.

**See also:**

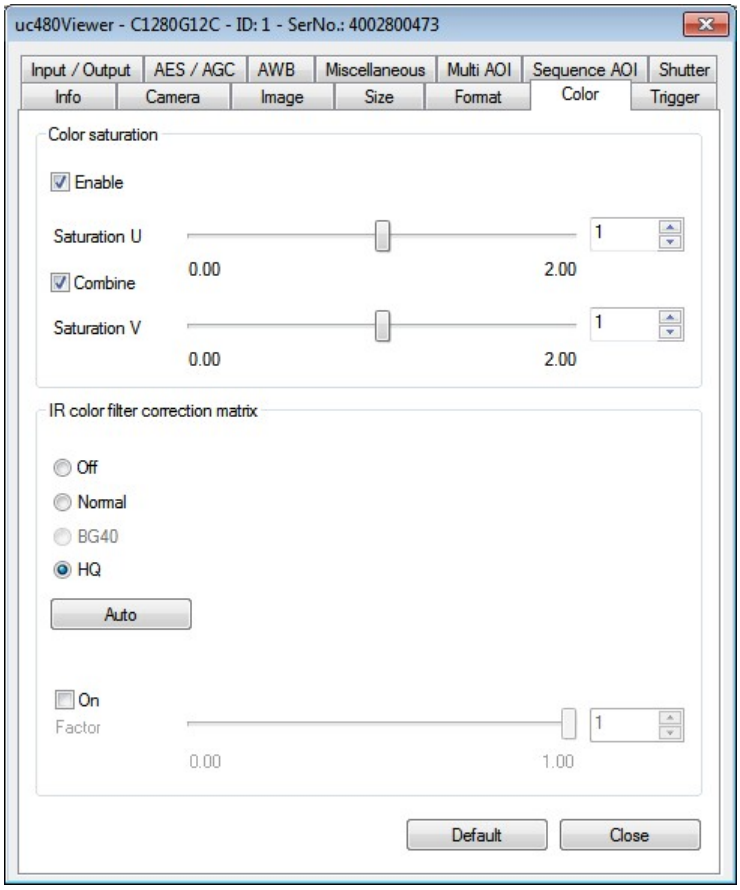
- Basics: Color filter (Bayer filter) 26
- Basics: Display modes 21
- Basics: Bit depth and digital contrast adjustment 41
- Basics: Characteristics and LUT 39
- Programming: `is_SetColorMode()` 319
- Programming (Open MP): `is_Configuration()` 183
- Color formats in memory: Appendix: Color formats 502
- Graphics card: System requirements 56

### 3.4.2.7.5 Color

This tab provides color rendering settings for your DCx camera (see also Color filter 26).

**Note**

This tab is only available if a color camera is opened in the uc480 Viewer.




#### Color saturation

This function enables and configures color saturation control.

In the YUV format, color information (i.e. the color difference signals) is provided by the U and V channels. In the U channel they result from the difference between the proportion of blue and Y (luminance), in the V channel from the difference between the proportion of red and Y. For the use in other color formats than YUV, U and V are converted using a driver matrix.

- Combine
  Selecting this check box synchronizes the two "Saturation U" and "Saturation V" sliders.

#### IR color filter correction matrix

When using color cameras with IR filter glass, you need to set the appropriate color correction matrix to ensure correct color rendering. The driver detects the IR filter type and sets this value automatically ("Auto" button). You can also select the correction matrix manually.

#### Sensor color correction

This function corrects the color values of a pixel. The colors are rendered more accurately after the color crosstalk of the individual Bayer pattern 26 filters has been eliminated by the color correction. The color correction factor is stepless adjustable between 0 (no correction) and 1 (full correction).

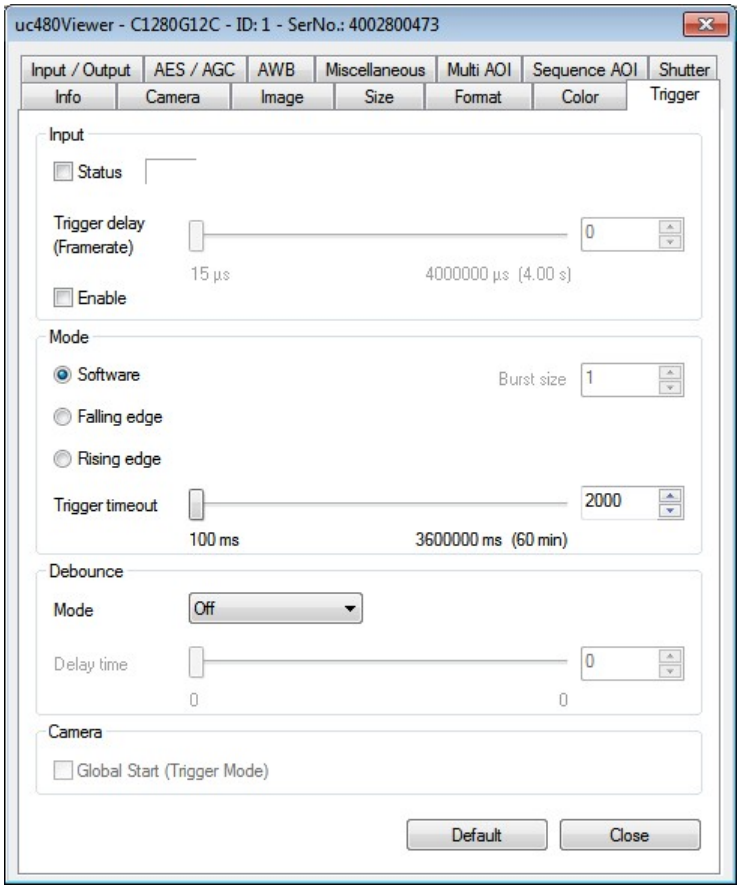
Activating the sensor color correction may increase CPU load.

#### Default

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.6 Trigger

This tab provides the settings for triggered image capture with your DCx camera (see also Digital in-/output (trigger/ flash) 47).




**Input**

- Status
  If you select this check box, the signal level applied at the camera's trigger input is polled and displayed.
- Trigger delay (frame rate)
  Select this check box to set a delay between the arrival of a software or hardware trigger signal and the start of exposure.

**Mode**

With these radio buttons, you choose which trigger mode you want to activate in the camera:

- Software
  The camera is running in software trigger mode without a signal applied. The images are captured continuously.
- Falling edge
  The camera captures an image on the falling edge of the signal applied to the trigger input.
- Rising edge
  The camera captures an image on the rising edge of the signal applied to the trigger input.
- Trigger timeout [ms]
  Specifies a timeout for the trigger mode. If the camera does not receive a trigger signal within this time, a timeout message is transmitted and the transmission error counter is incremented.

**Debounce**

This function is not relevant for DCx cameras.

**Camera**

- Global start (trigger mode)
  If you select this check box, all rows of a rolling shutter sensor are exposed simultaneously. Activating Global start 31 only makes sense when using a flash. This function is not supported by all models.

**Note**

**Frame rates in trigger mode**: Please note that the frame rate in trigger mode is always lower than in freerun mode. This is due to the sequential transmission. First the exposure takes place, then the transfer. A new exposure can only be performed after the transmission is completed. Therefore, the freerun mode is faster.

High trigger rates are achieved only with short exposure times and a high pixel clock setting.

**Default**

Click this button to reset all parameters to the model-specific defaults.

**See also:**

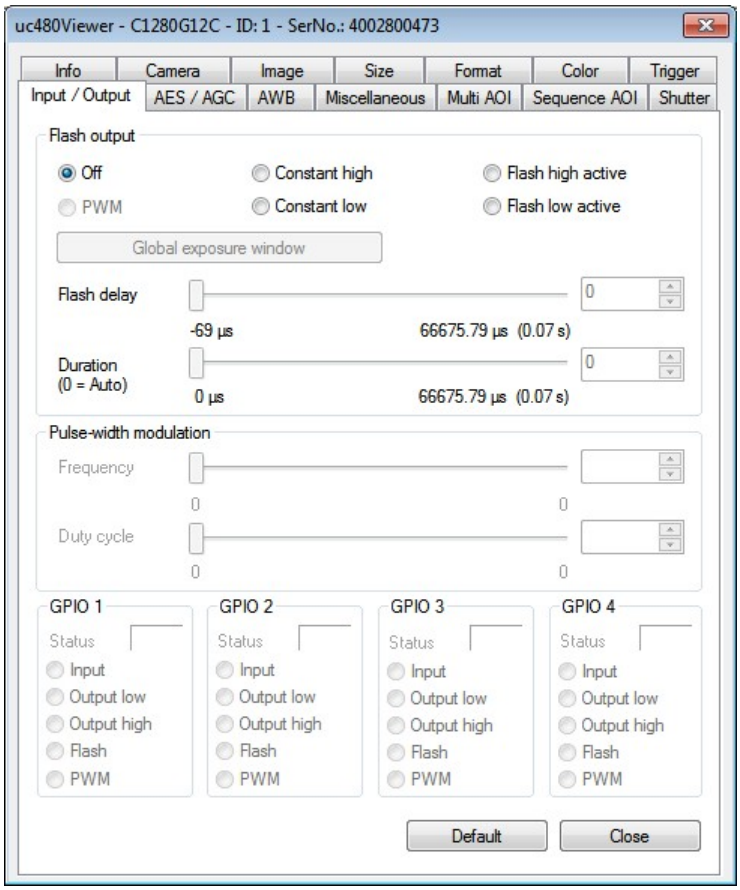
- Basics: Shutter methods: Global start 32

**Programming:**

- Trigger: `is_SetExternalTrigger()` 327

### 3.4.2.7.7 Input / Output

On this tab, you can set the parameters for the flash output and the GPIOs on your DCx camera (see also Digital in-/output (trigger/flash) 47).




**Attention**

When you are using the DCx Camera's flash function, you need to re-enable the flash (i.e. disable and then activate it again) whenever you change the pixel clock setting or horizontal image geometry. This is necessary to newly synchronize the internal timing settings of the flash output with the start of sensor exposure.

**Flash output**

With these radio buttons, you choose which digital output function you want to activate on the camera:

- Off
  The digital output is disabled.
- Constant high

The digital output is set to "High" regardless of the exposure.

- Constant low
  The digital output is set to "Low" regardless of the exposure.
- Flash high active
  The digital output is set to "High" during the exposure.
- Flash low active
  The digital output is set to "Low" during the exposure.
- Global exposure window
  The values for flash delay and duration are calculated to trigger the flash when all sensor rows are exposed simultaneously for sensors with rolling shutters. If you are using the "Global start" function, the flash delay has to be set to 0 (see also Shutter methods 31).
- Flash delay
  Sets the delay for the digital output. After an exposure has started, activating the digital output is delayed by the time set in flash delay.
- Duration (0 = Auto)
  Sets the switching time of the digital output. The digital output is activated for the time set in "Duration". The value 0 means that the signal is active over the entire exposure time (default).
  Requirement in freerun mode: The "Global exposure window" must be enabled.

**Pulse-width modulation (only DC320x Cameras)**

The pulse-width modulation (PWM) can be controlled by two parameters (frequency and duty cycle) and can be set on the flash output and the additional GPIOs respectively (see also `is_IO()` 280).

- Frequency (in Hz)
  Frequency of the pulse-width modulation (PWM)
- Duty cycle (in %)
  Duty cycle of the pulse-width modulation (PWM width)

**GPIO**

The GPIOs (General Purpose I/O) of the DCx Cameras can be used as inputs or outputs. See the Model comparison 459 list for the camera models which support GPIO.

- Status
  Polls the signal level applied to the GPIO.
- Input
  Sets the GPIO as trigger input.
- Output low
  Sets the GPIO as output. The output signal is statically set to "low".
- Output high
  Sets the GPIO as output. The output signal is statically set to "high".
- Flash
  Sets the GPIO as flash output. The settings you made in the "Flash output" box are used.

**Note**

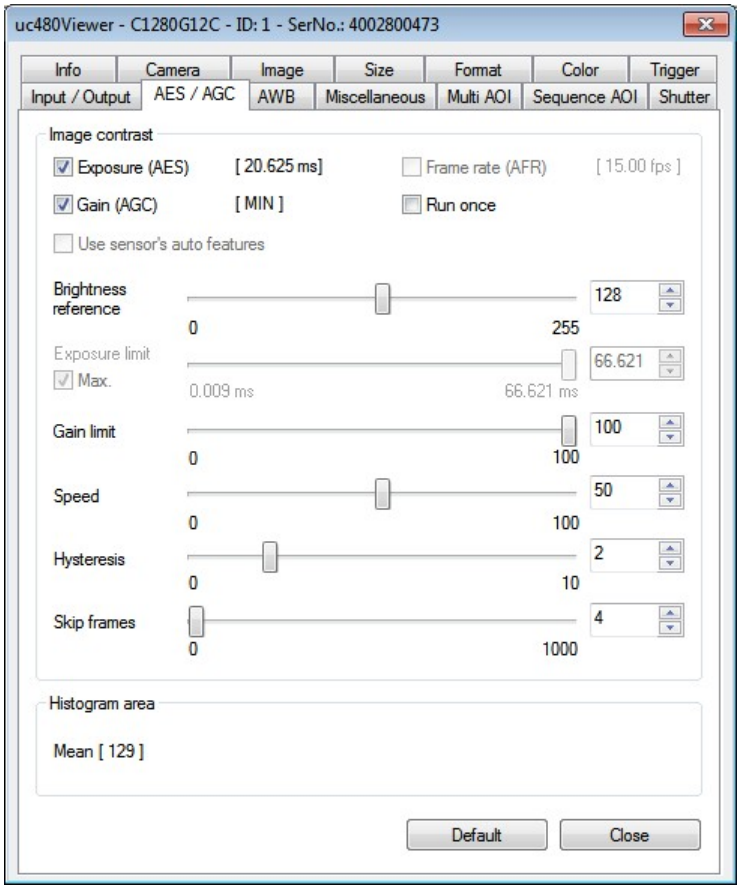
Detailed information on wiring the DCxCamera inputs and outputs is provided in the "Electrical Specifications" section of the Specifications 458 chapter.

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.8 AES/AGC (Automatic Brightness Control)

On this tab, you can set parameters for automatically adjusting the exposure time and sensor gain of your DCx camera (see also Automatic image control 45).




## Notes

- **Auto Exposure Shutter (AES)** automatically adjusts image brightness based on the exposure shutter setting. Long exposure times may cause motion blur.
- **Auto Gain Control (AGC)** automatically adjusts image brightness based on the hardware gain control 44 setting of the camera sensor. You can activate this function in addition to AES if ambient light conditions are poor. High gain settings may cause artefacting.
- **Auto Frame Rate (AFR)** adjusts the frame rate to allow longer exposure times (see also Pixel clock, frame rate and exposure time 44). Selecting this check box might decrease the frame rate substantially. This function is only available when AES is active.

**Image contrast**

Use the following settings to configure automatic brightness control:

- Exposure (AES)
  Enables automatic adjustment of image brightness, i.e. the exposure time is automatically adjusted to achieve the preset brightness reference value.

  **Note**

  If AES is enabled together with AGC and AFR, AES has the highest priority, i.e. AES is set first and then AGC and AFR.
- Framerate (AFR)
  Enables the framerate control. This option is only available when AES is active.
  If the maximum exposure time is not sufficient for a brighter image, you can use AFR to reduce the frame rate and so extend the exposure time.
- Gain (AGC)
  Enables the automatic gain, 44 i.e. the control automatically adjusts the gain to achieve the set brightness reference value.

- Run once
  The control is switched off automatically if the specified target value is reached or nearly reached, and the set value does not change after three measurements.
- Use sensor's auto features
  This option enables on some sensors the internal brightness adjustment.
- Brightness reference
  Defines the target value which should be set.
  Under normal conditions, a reference value of 128 is sufficient.
- Exposure limit
  Sets the upper limit for the exposure time. This upper limit is not exceeded if AES is enabled. If AES and AFR is enabled then the upper limit is ignored.
- Gain limit
  Sets the maximum gain limit. This upper limit is not exceeded when AGC is enabled.
- Speed
  Sets the adjustment speed. The higher the speed control is set, the faster the adjustment is. Setting the speed control to 0 disables the adjustment functionality. If the value is set to high the adjustment starts to oscillate. In this case you have to reduce the adjustment speed.
- Hysteresis
  Defines the difference by which the actual value can differ from the target value. If the target value is set e.g. to 128 and the hysteresis is set to 2, so there will be no regulation as long as the actual value is between 126 and 130 (see also Automatic image control: Hysteresis 46).
  If the value for hysteresis is too small, the adjustment starts to oscillate. It is recommended to keep the standard value of 2.
- Skip frames
  Number of frames that will be skipped during automatic image control when freerun mode is active (see also Automatic image control: Control speed 46). The higher the frame rate, the higher the value should be. If the value is too small, the adjustment may starts to oscillate, or there is an overload of the system. In this case increase the value.

**Histogram area**

The histogram area specifies which area of the image will be used for calculating the average grayscale value of the image. You can set the size of the histogram area by using the tools on the toolbar. 77

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.9 AWB (Automatic White Balance)

On this tab, you can set parameters for automatically adjusting the white balance of your DCx camera (see also Automatic image control 45).

**Note**

This tab is only available if a color camera is opened in the uc480 Viewer.

To carry out white balancing, 45 aim the camera at a surface of a uniform gray color. You can perform white balancing either manually with the RGB gain control or by using the "Auto White Balance (AWB)" function.

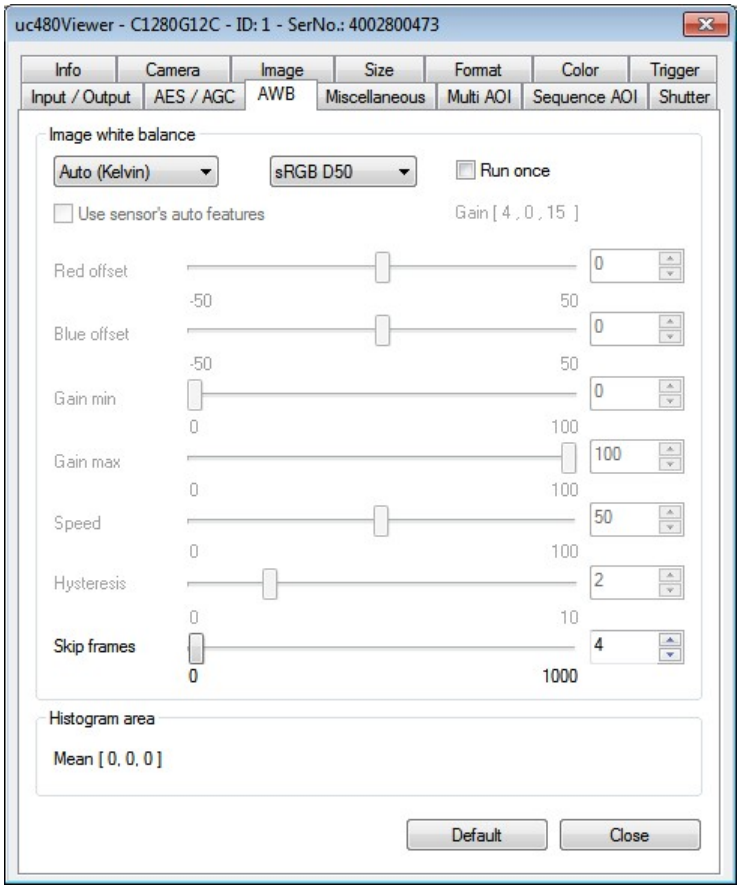



**Image white balance**

- Drop down list for AWB mode
  - o Off
    The automatic white balance is disabled.
  - o Auto (Gray World)
    If this option is enabled, the RGB gains are so controlled that the three colors have the same average.
  - o Auto (Kelvin)
    If this option is enabled, the RGB gains are controlled by predefined color temperatures.
- Drop down list for color spaces
  This drop down list is only available if AWB mode "Auto (Kelvin)" is selected. Here, you can select the color space e.g. sRGB D50 or CIE RGB E.
- Run once
  The control is switched off automatically if the specified target value is reached or nearly reached, and the set value does not change after three measurements.
- Use sensor's auto features
  This option enables on some sensors the internal white balance.
- Red offset/Blue offset
  Shifts the respective color intensity relative to green
  Example: If the Gray World algorithm controls to an average value of 65 and a blue offset is set of -40, the blue gain so is adjusted that blue has an average value of 25.

- Gain min
  Sets the minimum gain. This limit is not exceeded, i.e. a base gain is set for the white balance control. Via this minimal gain color cameras without master gain 95 can reproduce the gain.
- Gain max
  Sets the upper adjustment limit. This limit is not exceeded.
- Speed
  Sets the adjustment speed. The higher the speed control is set, the faster the adjustment is. Setting the speed control to 0 disables the adjustment functionality. If the value is set to high the adjustment starts to oscillate. In this case you have to reduce the adjustment speed.
- Hysteresis
  Defines the difference by which the actual value can differ from the target value. If the target value is set e.g. to 128 and the hysteresis is set to 2, so there will be no regulation as long as the actual value is between 126 and 130 (see also Automatic image control: Hysteresis 46).
  If the value for hysteresis is too small, the adjustment starts to oscillate. It is recommended to keep the standard value of 2.
- Skip frames
  Number of frames that will be skipped during automatic image control when freerun mode is active (see also Automatic image control: Control speed 46). The higher the frame rate, the higher the value should be. If the value is too small, the adjustment may starts to oscillate, or there is an overload of the system. In this case increase the value.

**Histogram area**

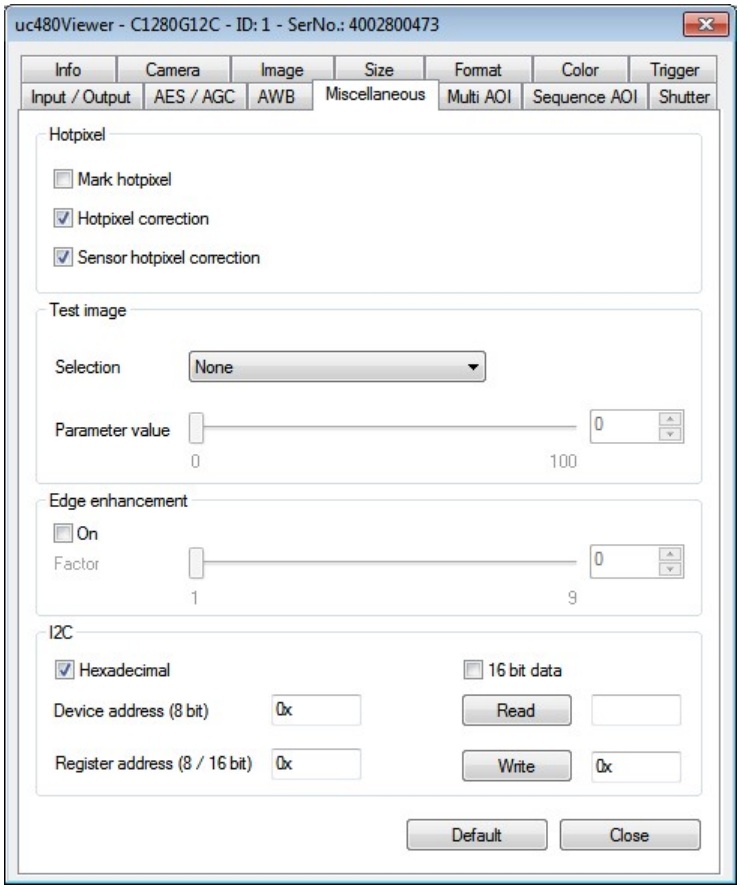
The histogram area specifies which area of the image will be used for calculating the average value of the image. You can set the size of the histogram area by using the tools on the toolbar. 77

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.10 Miscellaneous

This tab provides parameters for setting miscellaneous functions for your DCx camera.




**Note**

**Sensor hotpixel correction and DCC1240x / DCC3240x**: If sensor hot pixel correction and Multi AOI are enabled, the sensor displays a four pixel wide black line between the AOIs.

**Hotpixel**

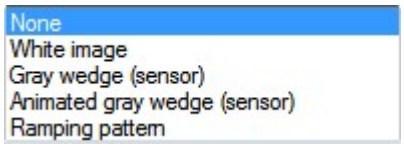
This function corrects the sensor hotpixels specified in the camera EEPROM.

- Mark hotpixel
  If you select this check box, the hotpixel positions stored in the camera are graphically represented in the image.
- Hotpixel correction
  Select this check box if you want to enable a software-based hotpixel correction.
- Sensor hotpixel correction
  Enables the sensor's internal hotpixel correction for some models.

**Test image**

The camera transmits a selectable test image that you can use for testing the data transmission. You can choose from different types of test images, depending on the camera type.

- Selection




- Parameter value
  You can adjust the appearance of some of the test images with the "Parameter value" slider.

**Note**

- Manually changing the pixel clock will disable the test image mode.
- Animated test images are ideal for testing recorded sequences.
- With some sensors, the sensor gain setting has an influence on the test image.
- A white test image is recommended to check the camera's maximum load on the USB bus. Due to the transmission process, completely white camera images require somewhat more bandwidth on the USB bus than completely black images.
- To turn on hotpixel correction does not make sense for a test image.
- The test images are transferred with a bit depth of 8 bit.

**Edge enhancement**

- This function activates a software filter that emphasizes the edges in the image.
  Enabling the edge enhancement function increases the CPU load during image capture.

**I2C**

This function is not relevant for DCx Cameras.

**Default**

Click this button to reset all parameters to the model-specific defaults.

**See also:**

- Basics: Hot pixels 28

**Programming:**

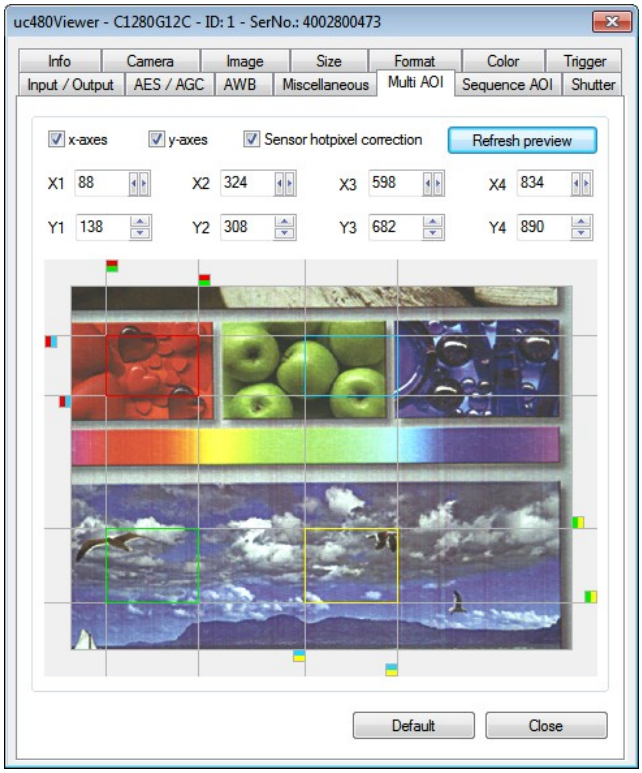
- Hotpixel: `is_HotPixel()` 260
- Edge enhancement: `is_EdgeEnhancement()` 206
- Test image: `is_SetSensorTestImage()` 345

### 3.4.2.7.11 Multi AOI

This tab provides settings for configuring the Multi AOI feature provided by specific DCxCamera sensors. The Multi AOI function allows defining more than one AOI in an image and transferring these AOIs all at the same time (see Area of interest (AOI) 35 in the "Basics" chapter).

**Note**

This tab is only available when a DCC1240x or DCC3240x Camera has been opened in uc480 Viewer.




**Notes**

- When the Multi AOI function is enabled, no changes can be made to the image size settings (e.g. binning, subsampling, scaling). Disable Multi AOI to change the size of an image and then re-enable the function.
- If sensor hotpixel correction and Multi AOI are enabled, the sensor displays a four pixel wide black line between the AOIs.
- 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 461 chapter.
- It is possible to switch the AOI in the horizontal direction.

- X-axes/Y-axes
  Select one of the two check boxes to create two AOIs along the X or Y axis. Select both check boxes to create four AOIs.
- Sensor hotpixel correction
  This check box enables the sensor's internal hotpixel correction function.
- Refresh preview
  Refreshes the preview image shown at the bottom of the tab.
- X1...X4/Y1...Y4
  Defines the X and Y values in pixels for the individual AOIs.
- Preview window
  Displays a preview image and allows graphically setting the individual AOIs by moving the colored boxes that are displayed at the end of each axis.

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.7.12 Sequence AOI

On the "Sequence AOI" tab you do the settings for the sequence AOI mode of some DCx Cameras. In this mode you can capture e.g. the same AOI with different parameter sets – exposure, gain, etc. (see Basics: Area of interest (AOI) 35).

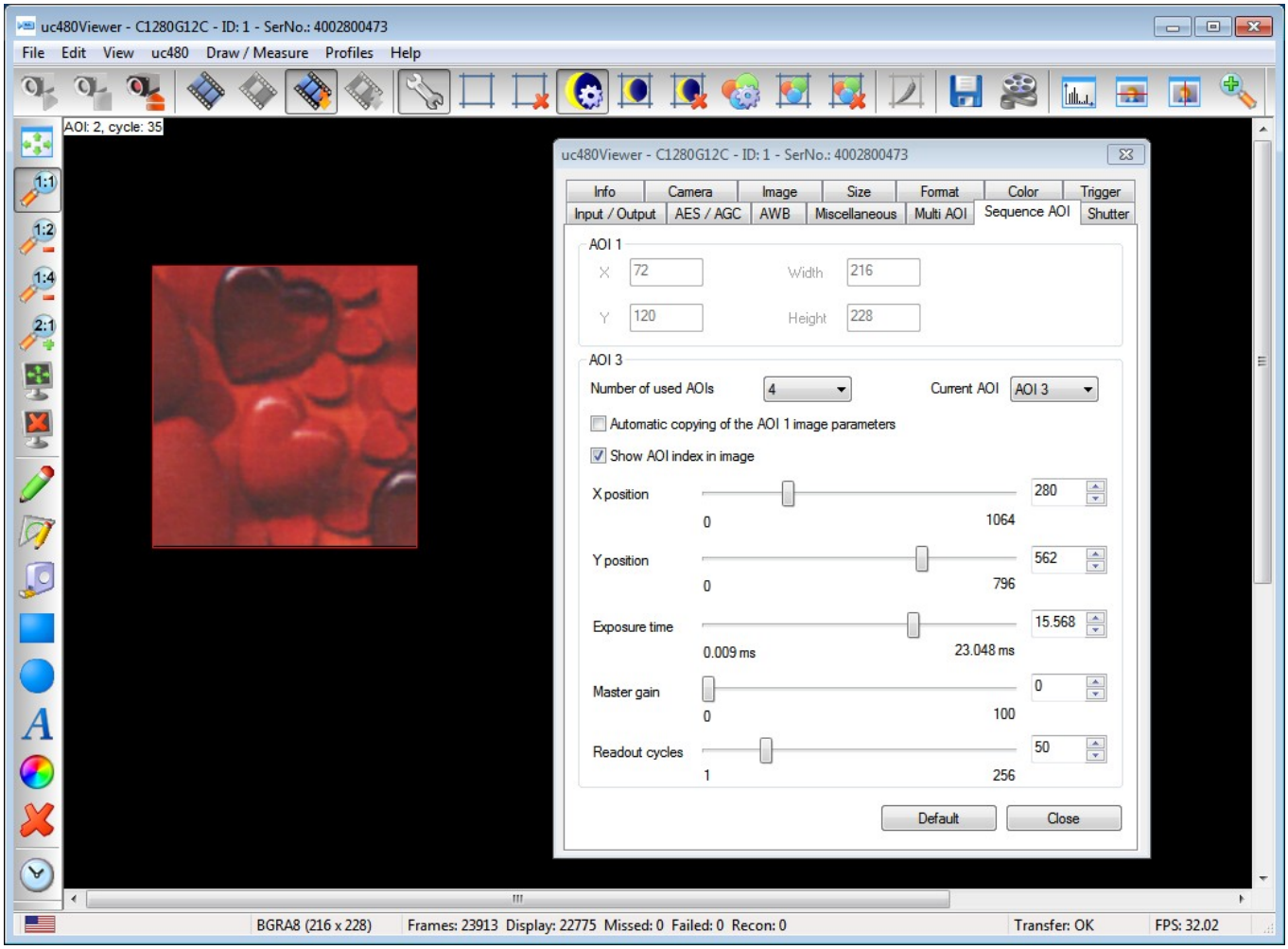
**Note**

- The sequence AOI mode is only supported by DCC1240x and DCC3240x camera models.
- Note that the sequence AOI mode cannot be used in combination with scaler, subsampling or binning.

In contrast to the Multi AOI 112 the different AOIs are captured one after another in sequence AOI mode. The sensor readouts all AOIs sequentially and transfers the images as e.g. the same AOI is readout four times. The frame rates are identical, but you can define different settings. For every AOI exposure time and master gain can be set individually. For example you can capture the same AOI at first with a short exposure time and after this with a long exposure time. An advantage of the sequence AOI mode is that you need no time for moves of the position and it is not necessary to reprogram the AOI.

Beside the normal AOI (AOI 1) you can define up to three additional AOI 2-4. These additional AOI have the same size as AOI 1, but you can set different parameters for:

- Position of AOI
- Exposure time
- Gain
- Readout cycles (number of images)




After you have defined an AOI, open the "Sequence AOI" tab. In the upper area you see the position and size of AOI 1. In the area below you define the further AOIs.

- Number of used AOIs
  Here, you define the numbers of additional AOI. If you select further AOIs, the parameters of AOI 1 are copied for

the new AOI initially.

- Current AOI
  Here, you select the AOI which parameters you want to edit. You can only change the parameters of AOI 2, 3 or 4.
- Automatic copying of the AOI 1 image parameters
  If you change AOI 1, e.g. set a new exposure time, these changes are copied to AOI 2 etc. if this option is activated. Otherwise the changes of AOI 1 have no influence on the further AOIs.
- Show AOI index in image
  Activate this option see in the left upper corner the number of the AOI. By default the font has the color black.
- X position/Y position
  Moves the current AOI in vertical or horizontal direction respectively. An overlapping of the different AOIs is possible.
- Exposure time
  Changes the exposure time of the current AOI.
- Master gain
  Setting the gain for the current AOI.
- Readout cycles
  Here, you can define for AOI 2-4 how often they are readout and the images are transferred. AOI 1 is always readout once.

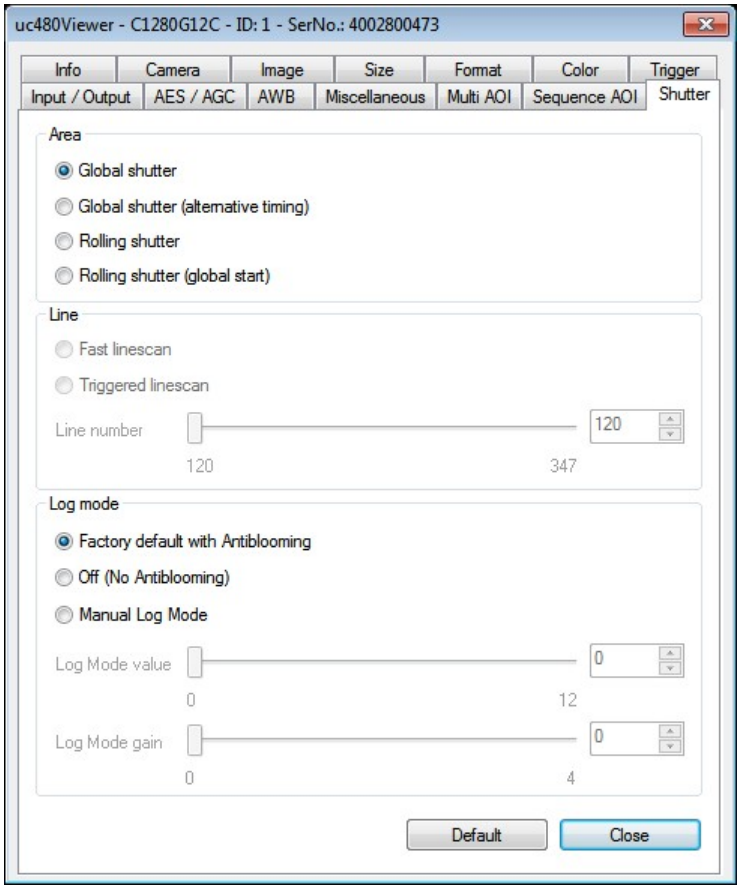
**Hint**

If capturing the same AOI with different exposure times, you can create via software a HDR image from the single images.

### 3.4.2.7.13 Shutter

**Note**

This tab is only available when a DCC1240x or DCC3240x Camera has been opened in uc480 Viewer and provides parameters for setting the readout modes provided these camera's sensors (see also Shutter methods 30 and Line scan mode 33 in the "Basics" chapter).




## Area

- Global shutter
  The camera is operated in global shutter mode. 30 This mode allows capturing fast-moving objects without geometric distortions.
- Global shutter (alternative timing)
  When using the camera in global shutter mode there can be in some cases problems with the black level. In the mode "Global shutter (alternative timing)" the sensor holds the black level as in rolling shutter mode. This mode should not be used with a frame rate below 2 fps.
- Rolling shutter
  The camera is operated in rolling shutter mode. 31 This mode reduces image noise in image capture. To capture fast-moving objects in rolling shutter mode, you might need a flash.
  Note that the flash timing can conflict with the sensor exposure if you use flash in rolling shutter mode without global start.
- Rolling shutter (global start)
  The camera is operated in rolling shutter mode but with global start. 32 As in this mode the exposure of all lines is started at the same time, this mode is especially suitable for flash.

For further information see "Basics": Shutter methods 30.

**Line**

**Note**

Currently, only the monochrome DCC1240M and DCC3240M/N camera models support the "fast line scan" mode.

- Fast linescan
  Enables the sensor's fast line scan mode.

- Line number
  Defines the line to be scanned

For further information see "Basics": Line scan mode 33.

**Log mode**

The Log mode is a special mode of the DCC1240x and DCC3240x camera models. It defines the threshold defines at which the linear sensitivity changes into a logarithmic characteristic.

At very short exposure times (less than 0.1 ms) there may occur e.g. so-called crosstalk effects in the global shutter mode, which have the effect that the image content appears brighter in the vertical from top to bottom. This effect can be avoided by the Log mode.

- Factory-default with anti-blooming
  Recommended setting
- Off (no anti-blooming)
  At longer exposure times without gain, it may happen that the sensor does not achieved white. This is because of the sensor automatism to prevent overexposure (blooming). Remedy offers the use of a gain factor of 1.4x or the mode "Off (no anti-blooming)."
  For short exposure times this mode should not be used, otherwise the image (particularly in global shutter mode) can be extremely overexposed.
- Manual Log mode
  In this mode it is possible to display the information in the overexposed image areas. This mode is effective for exposure times below 5 ms.
  - LogMode value
    Using a slide control with 13 levels you can adjust the threshold at which the linear sensitivity pass over into a logarithmic characteristic. Here, the value "0" represents the lowest active level and "12" corresponds to the highest level.
  - LogMode gain
    When using the manual Log mode no master gain is possible. The gain can be adjusted with the LogMode gain. The gain can be adjusted in 5 levels. The level "3" corresponds to a gain factor of 1 for monochrome or color sensors. For the NIR sensor the level "1" corresponds to a gain factor of 1.
    Example: A low level results in a low gain and may display more details in the overexposed image areas. A higher level gives a higher gain, thereby a darker image can be brightened (see also DCC1240x / DCC3240x Application Notes 63).

**Default**

Click this button to reset all parameters to the model-specific defaults.

### 3.4.2.8 Creating profiles

**Profiles for setting camera parameters in uc480 Viewer**

Profile files for uc480 Viewer are text files with the file name extension *.ucp. As opposed to uc480 parameter files 504 (ini files), profiles are not tied to a specific camera model and do not contain all the camera settings. Some parameters in profiles are relative settings (e.g. exposure 0-100 %) to ensure compatibility with all camera models.

You can create new profiles and load them from the start dialog 76. When you start uc480 Viewer, the application searches the `C:\Program Files\IDS\uc480\Program\` folder for profile files and lists them in the Profiles menu 83. The profile names shown in the menu are named after the files.

**Attention**

Do not remove or rename the profile files created in the `C:\Program Files\Thorlabs\DCx Cameras\Profile\` folder when you install the application. Otherwise program functions in uc480 Viewer may not work correctly.

**Note**

The profiles are a feature of uc480 Viewer. They cannot be used through the uc480 API.

**Structure of a profile file for setting camera parameters**

Profile files have the structure described below. They do not have to contain all the parameters listed here.

| Parameter | Description |
|---|---|
| **[DISPLAY]** | |
| PIXEL_FORMAT | Color format<br>see also `is_SetColorMode()` 319 |
| RENDER_MODE | Render mode (DIB, Direct3D, OpenGL)<br>see also Image display modes 21 and `is_SetDisplayMode()` 322 |
| FIT_TO_WINDOW | Fits the image to the window |
| [TIMING] | |
| PIXELCLOCK | Pixel clock in MHz<br>see also Pixel clock, frame rate, exposure time 44 and `is_PixelClock()` 294 |
| FRAMERATE | Frame rate 0-100 %<br>see also `is_SetFrameRate()` 329 |
| EXPOSURE | Exposure time 0-100 %<br>see also `Is_Exposure()` 216 |
| **[COLOR]** | |
| CORRECTION_FACTOR | Color correction factor 0-100 %<br>see also `is_SetColorCorrection()` 317 |
| SATURATION_U | Color saturation 0-2x<br>see also `is_SetSaturation()` 342 |
| SATURATION_V | Color saturation 0-2x<br>see also `is_SetSaturation()` 342 |
| **[IMAGE_PARAMETERS]** | |
| EDGE_ENHANCEMENT | Edge enhancement<br>see also `is_EdgeEnhancement()` 206 |
| GAMMA | Gamma correction<br>see also `is_SetGamma()` 332 |
| **[AES_AGC_AFR]** | |
| EXPOSURE | Auto exposure shutter<br>see also `is_SetAutoParameter()` 303 |
| GAIN | Auto gain control<br>see also `is_SetAutoParameter()` 303 |
| FRAMERATE | Auto frame rate setting for auto exposure shutter<br>see also `is_SetAutoParameter()` 303 |

| Parameter | Description |
|---|---|
| **[AWB]** | |
| WHITEBALANCE | Auto white balance<br>see also is_SetAutoParameter() 303 |

### Example of a profile file

```
[DISPLAY]
PIXEL_FORMAT        = 0         // 0 = RGB32, see uc480.h for defined formats
RENDER_MODE         = 1         // 1 = IS_SET_DM_DIB, 4 = IS_SET_DM_DIRECT3D, see uc480.h for defined render modes
FIT_TO_WINDOW       = 1         // 0 = No scaling, 1 = Scale image to window

[TIMING]
PIXELCLOCK          = 30        // MHZ
FRAMERATE           = 100       // 0 = min, 100 = max
EXPOSURE            = 50        // 0 = min, 100 = max

[COLOR]
CORRECTION_FACTOR   = 100       // 0 = 0.0, 100 = 1.0
SATURATION_U        = 120       // 0 = 0.0, 200 = 2.0
SATURATION_V        = 120       // 0 = 0.0, 200 = 2.0

[IMAGE_PARAMETERS]
EDGE_ENHANCEMENT    = 0         // 0 = disable, 1 = weak, 2 = strong
GAMMA               = 160       // 100 = 1.0, 220 = 2.2

[AES_AGC_AFR]
EXPOSURE            = 1         // 1 = on, 0 = off
GAIN                = 1         // 1 = on, 0 = off
FRAMERATE           = 0         // 1 = on, 0 = off
SPEED               = 50        // 0 = min, 100 = max
SKIPFRAMES          = 0         // 0 = min, 100 = max

[AWB]
WHITEBALANCE        = 1         // 1 = on, 0 = off
SPEED               = 50        // 0 = min, 100 = max
SKIPFRAMES          = 0         // 0 = min, 100 = max
```

### See also:

- "Choose mode" start dialog 76
- "Profiles" menu 83

## 3.4.3 uc480 Player

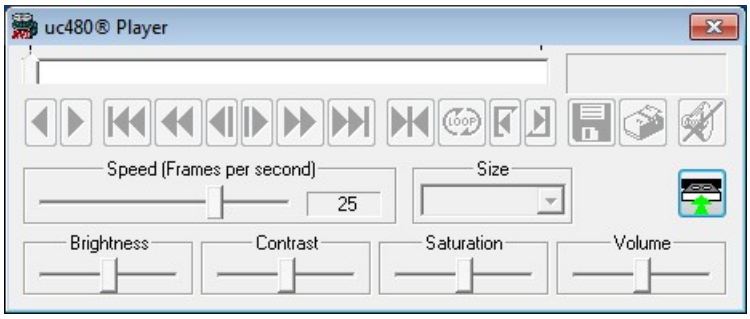
Using the uc480 Player, you can open and play back AVI files, e.g. created with the uc480 Viewer 75 program, in MJPEG format. Images stored in JPG/BMP format can also be displayed.

The uc480 Player can be accessed as follows: Start > All Programs > uc480 > uc480 Player

**Note**

Only one instance of the uc480 Player can be opened at a time.

After program start, the uc480 Player will display the following dialog box:

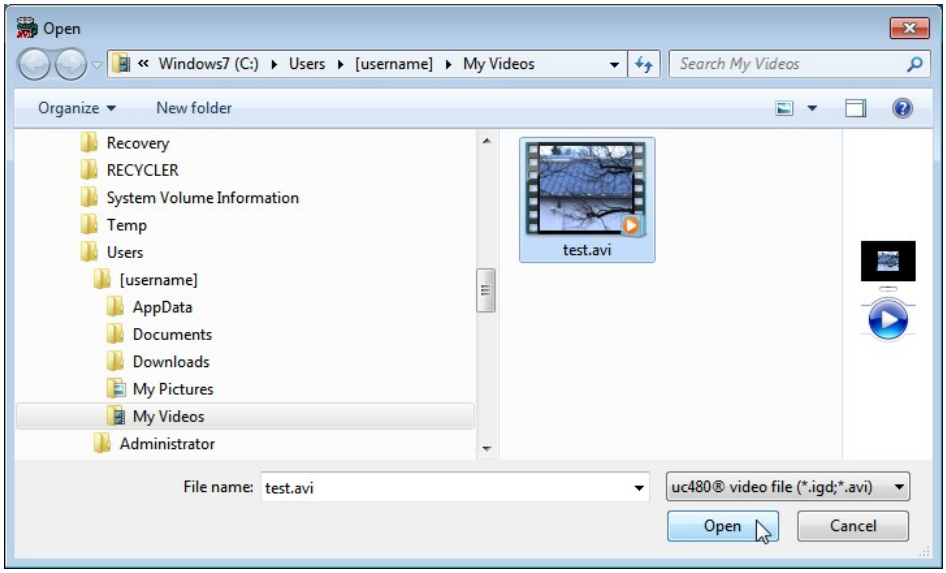



The user interface of the uc480 Player adjusts to the language of the operating system. After the player has started, only the button for loading a video file is active. How to load a video file will be explained in the following section.

### 3.4.3.1 Loading an AVI file

After clicking the button, the "Open File" dialog box opens where you can select one or more files to be opened. If you select multiple files, they will be played back one after the other in alphabetical order.

Alternatively, AVI files can also be opened and played back simply by drag and drop. To do this, drag the files with the left mouse button pressed into the uc480 Player dialog box; then, release the mouse button.




Once the required file has been opened, another window for the video film appears on top of the uc480 Player. You can move this window freely around the screen, independently of the player window.

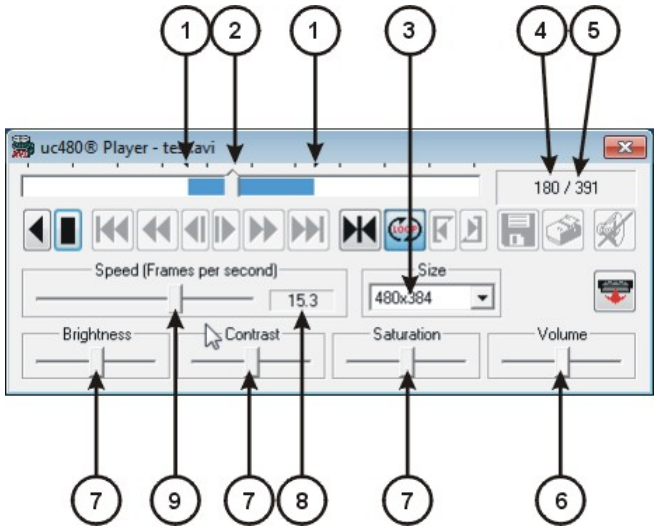
When you place the mouse pointer inside the video window, its display changes to a small magnifying glass and the zoom function of the uc480 Player is enabled. Using the left mouse button, you can now select an area within the video image that will be resized to fill the window (Zoom In), even during playback. Double-clicking the left mouse button inside the window will revert the display to its original size (Zoom Out).


### 3.4.3.2 Overview of the Operation Controls

The buttons in the uc480 Player user interface are for the most part self-explanatory and are based on the keys and symbols of a standard video recorder.

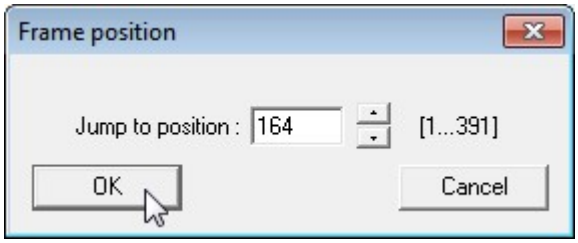
| | |
|---|---|
|  | Reverse: play video backwards |
|  | Play: play video forwards |
|  | Stop: stop playing the video (symbol appears after you click the Play button). The last frame will be frozen. |
|  | Jump to start of video |
|  | Fast rewind |
|  | One frame back |
|  | One frame forward |
|  | Fast forward |
|  | Jump to end of video |
|  | Go to specific frame. When you click this button, a small dialog box will open where you can enter the frame number. |
|  | Start Loop mode (blue text) |

| | |
|---|---|
|  | Stop Loop mode (red text) |
|  | Start of playback loop 122 |
|  | End of playback loop 122 |
|  | Save current frame as BMP file or JPEG file |
|  | Print current frame |
|  | Sound on/off |
|  | Open video file |
|  | Close video file |




| | |
|---|---|
| 1 | Position markers for the defined playback loop 122 |
| 2 | Current position in video file |
| 3 | Size of the video display window in pixels |
| 4 | Current frame number |
| 5 | Number of frames in video file |
| 6 | Volume control |
| 7 | Frame display parameter settings. Each slider can be reset to its default setting by clicking it with the right mouse button. The settings are also applied to the subsequent files. |
| 8 | Current playback speed setting |
| 9 | Playback speed in relation to the recording speed from 0.1 to 200 fps. The values are set in increments. |

Using the  button, you can jump directly to a specific frame. To do so, enter a numerical value between 1 and the total number of frames in the video sequence.




### 3.4.3.3 Loop Mode

When using the uc480 Player, you can select specific periods of time within the video sequence and play them back in an endless loop. To do this, click the button. This enables the two icons for marking the start and end of the playback loop. To select the start position, use the mouse to drag the position marker to the desired start position in the endless loop and then click the button. Then, set the position marker to the desired end position using the same method. Click the button to complete defining markers.

### 3.4.3.4 Video Window and Full Screen Mode

The video window is displayed dynamically. The possible display sizes are determined based on the capture resolution and the screen resolution.

**Note**

The video is played back at the same aspect ratio that was used for the capture.

The full screen mode is activated by pressing the keys CTRL+F. In the full screen mode you can control the playback via the keyboard. You leave the full screen mode by pressing ESC or CTRL+F.

**Key combinations in full screen mode**

| | |
|---|---|
| CTRL + F | Start/stop full screen mode |
| ESC | Quit full screen mode |
| CTRL + O | Open video file |
| <- (Arrow key left) | One frame back |
| -> (Arrow key right) | One frame forward |
| Space | Start/stop video playback |

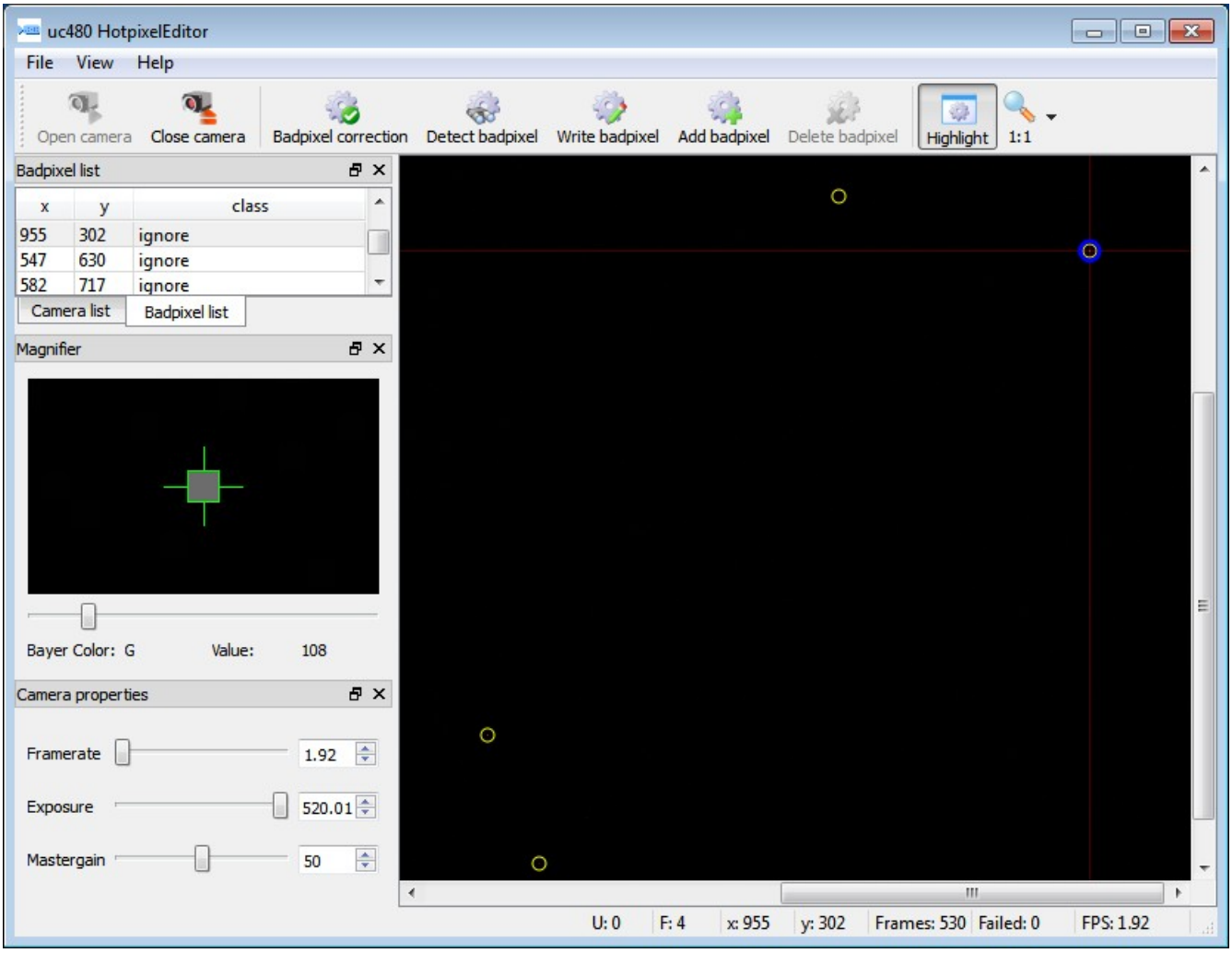
### 3.4.4 uc480 Hotpixel Editor

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 28). 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:

- Start > All programs > uc480 > uc480 Hotpixel Editor

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 125
  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.

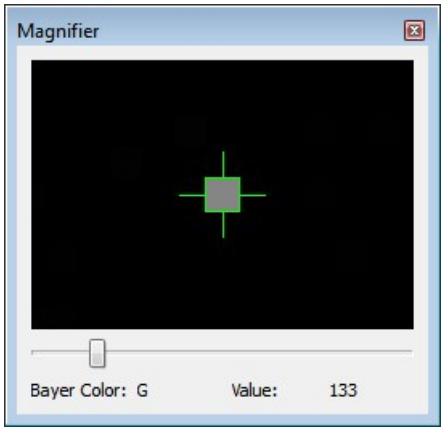
**Tab Badpixel List:**

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

Class "user": added by the user

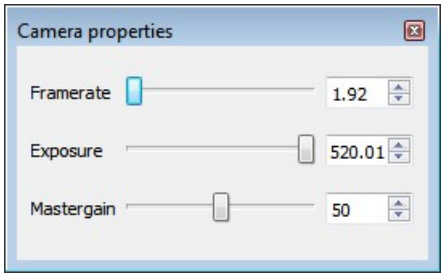
Class "ignore": factory calibrated hot pixels

**Magnifier function**



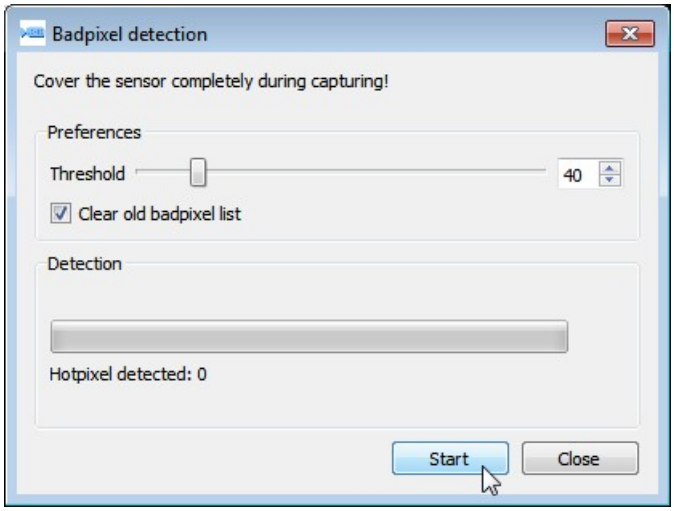

The magnifier allows to exactly mark hot pixels. "Value" displays the relative pixel intensity.

**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.

**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 67, the uc480 Viewer 75 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 127
- How to proceed 129
- Function descriptions 155
- AVI function descriptions 359
- Obsolete functions 379
- Programming notes 451
- Lists 454

# 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 129 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 451 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 21. Under Linux the OpenGL mode 21 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()` 322 to select the display mode. You can then customize the Direct3D mode by using `is_DirectRenderer()` 198.

**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()` 157, add them to a memory sequence 132, if required, and then activate a memory with `is_SetImageMem()` 337 before each image capture. To show the image on-screen, call the `is_RenderBitmap()` 297 function after each completed image capture. From the events or messages 141 you can see when an image is available for display.

**See also:**

- How to proceed: Display mode selection 133

### Capture images

Recording live images with the DCxCamera is very simple. Just call the `is_CaptureVideo()` 177 function and the camera captures the live images at the default frame rate. To capture single frames, use the `is_FreezeVideo()` 223 function. Every DCx camera of course also provides different trigger modes for image capture. Use `is_SetExternalTrigger()` 327 to activate the desired mode before starting the image capture.

**See also:**

- How to proceed: Image capture 134

### Adjust the frame rate, brightness and colors

To change the frame rate, for example, you call `is_SetFrameRate()` 329. With `is_SetColorMode()` 319 you set the color mode. Image brightness is adjusted through the exposure time set with `is_Exposure()` 216. You can also implement automatic control 147 of image brightness and other parameters by using `is_SetAutoParameter()` 303.

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()` 317). 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()` 303 function.

**See also:**

- How to proceed: Setting camera parameters 144

### Save an image

Use the `is_ImageFile()` 264 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 149

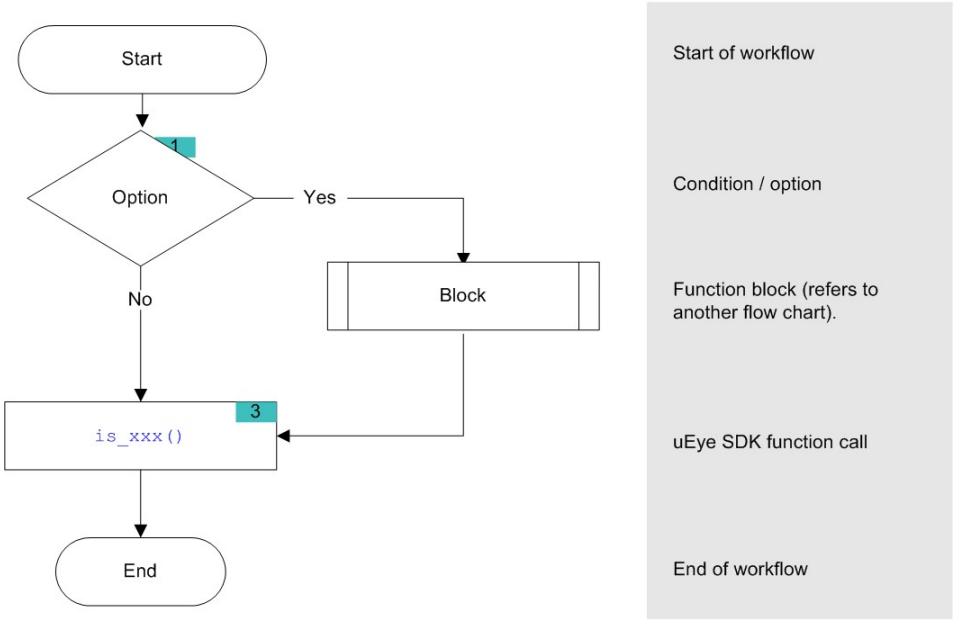
### Close the camera

When you want to exit your application, close the camera with `is_ExitCamera()` 213. 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()` 292 function before closing the camera. The next time you start the application, you can simply load the settings again by using the same function.

## 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 129: Before you open one or more DCx Cameras, we recommend querying some key information.
- Opening and closing a camera 130
- Allocating an image memory 130: 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 132 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()` 249 | Determines the number of cameras connected to the system. |
| `is_GetCameraList()` 233 | 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()` 326 | Enables/disables dialog messages for error output. |

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

| | |
|---|---|
| `is_CameraStatus()` 172 | Returns the event counters and other information.<br>Enables standby mode. |
| `is_GetCameraInfo()` 231 | Returns the camera information of an opened camera. |
| `is_GetDLLVersion()` 238 | Returns the version of the `uc480_api.dll`. |
| `is_GetOsVersion()` 250 | 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() 273 | Hardware initialization |
| is_ExitCamera() 213 | 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() 313 | 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 127).

| | |
|---|---|
| is_AllocImageMem() 157 | Allocates an image memory. |
| is_SetAllocatedImageMem() 301 | The user provides pre-allocated memory for image capturing. |
| is_FreeImageMem() 222 | Releases an allocated image memory. |

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

| | |
|---|---|
| is_SetImageMem() 337 | Makes an image memory active. |

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

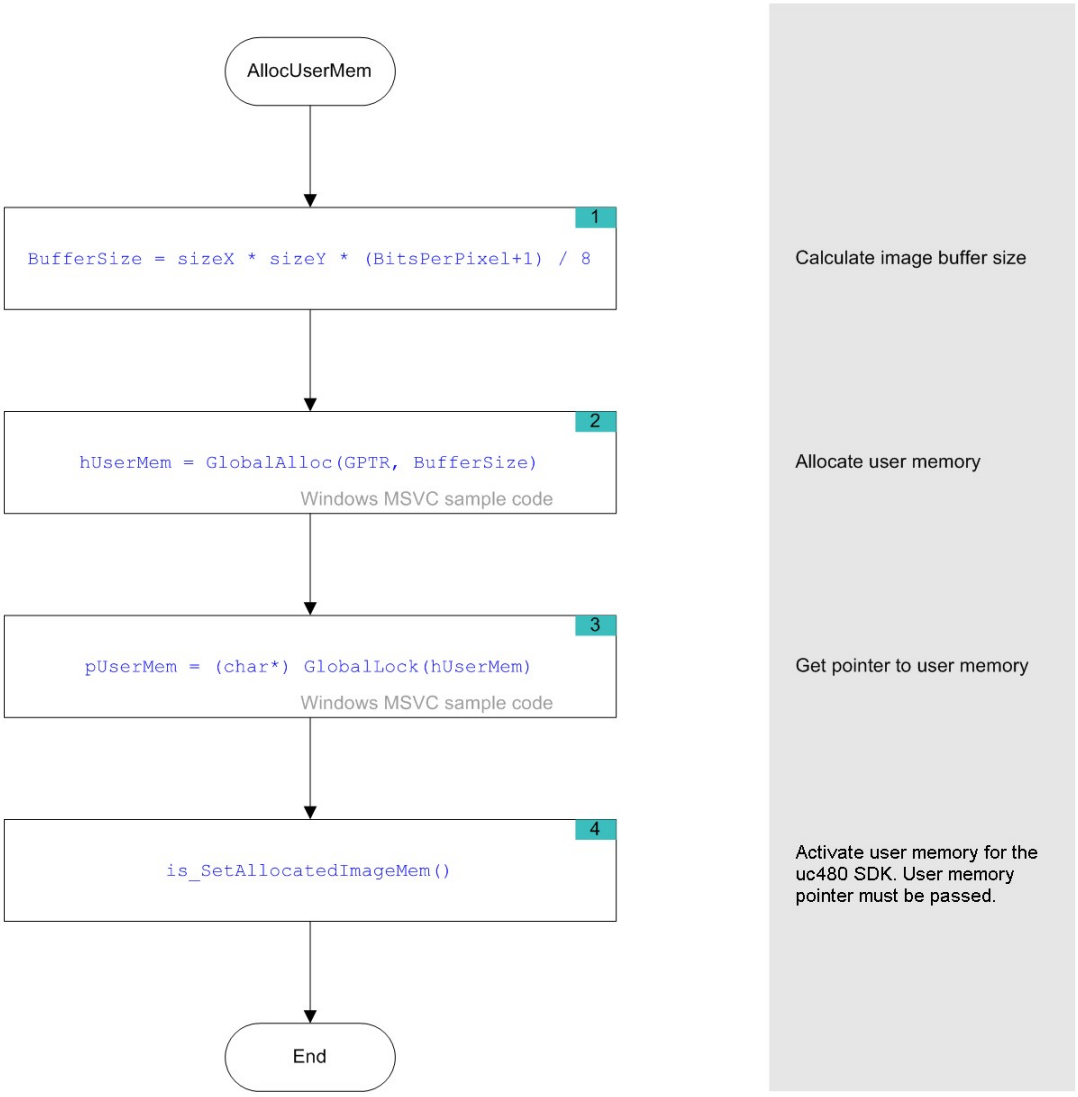
| | |
|---|---|
| is_CopyImageMem() 190 | Copies the image to the user-defined memory. |
| is_CopyImageMemLines() 191 | Copies selected image lines to the user-defined memory. |
| is_GetActiveImageMem() 225 | Returns the number and address of the active image memory. |
| is_GetImageMem() 247 | Returns the pointer to the starting address of the image memory. |
| is_GetImageMemPitch() 248 | Returns the line offset used in the image memory. |
| is_InquireImageMem() 279 | Returns the properties of an image memory. |

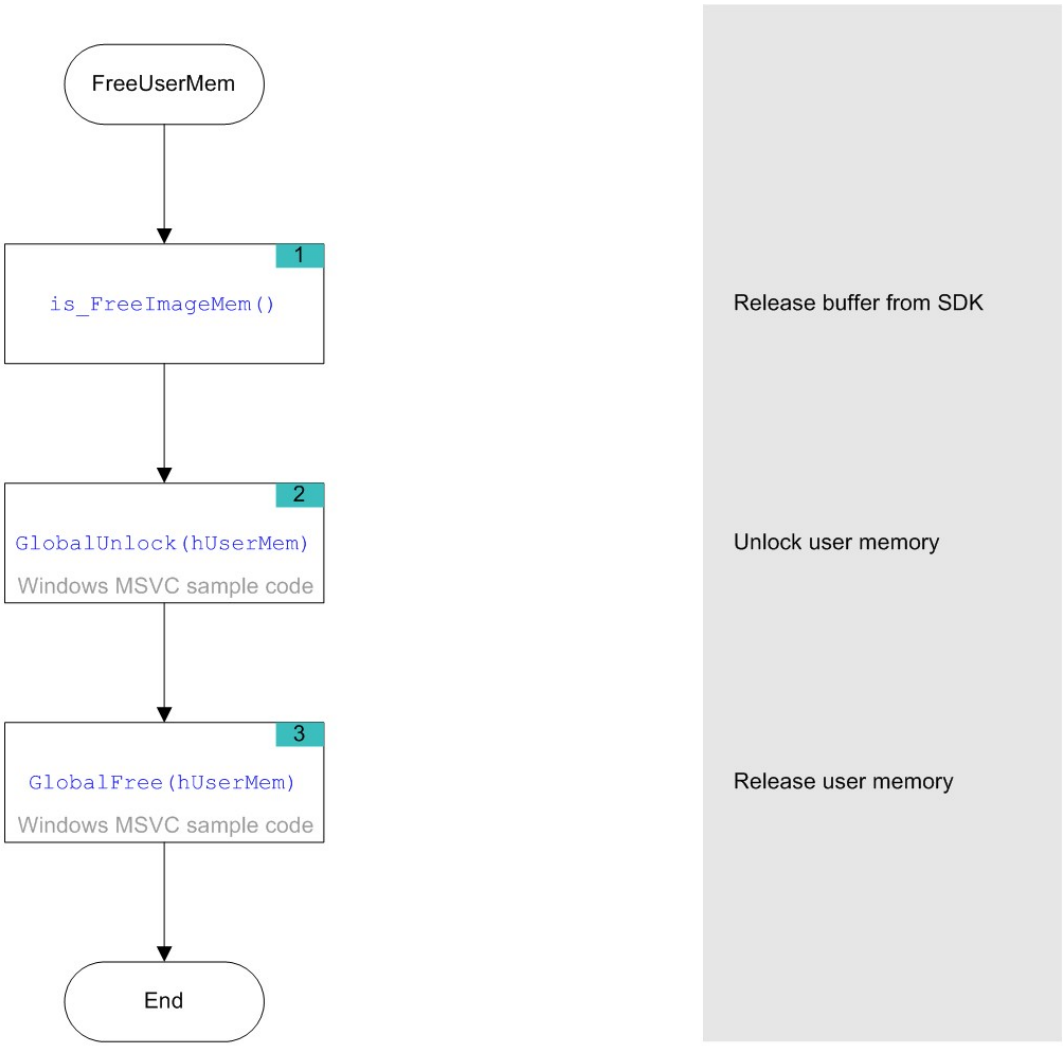
**Note**

Image memory sequences 132 should be used for frame sequence capture.

**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 141 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 130).

| | |
|---|---|
| is_AddToSequence() 156 | Adds image memory to the sequence list. |
| is_ClearSequence() 179 | Deletes the entire sequence list. |
| is_GetActSeqBuf() 226 | Determines the image memory currently used for the sequence. |
| is_SetImageMem() 337 | Makes the indicated image memory the active memory. |
| is_LockSeqBuf() 289 | Protects the sequence image memory from being overwritten. |
| is_UnlockSeqBuf() 354 | Releases the sequence image memory for overwriting. |

## 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 21.

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

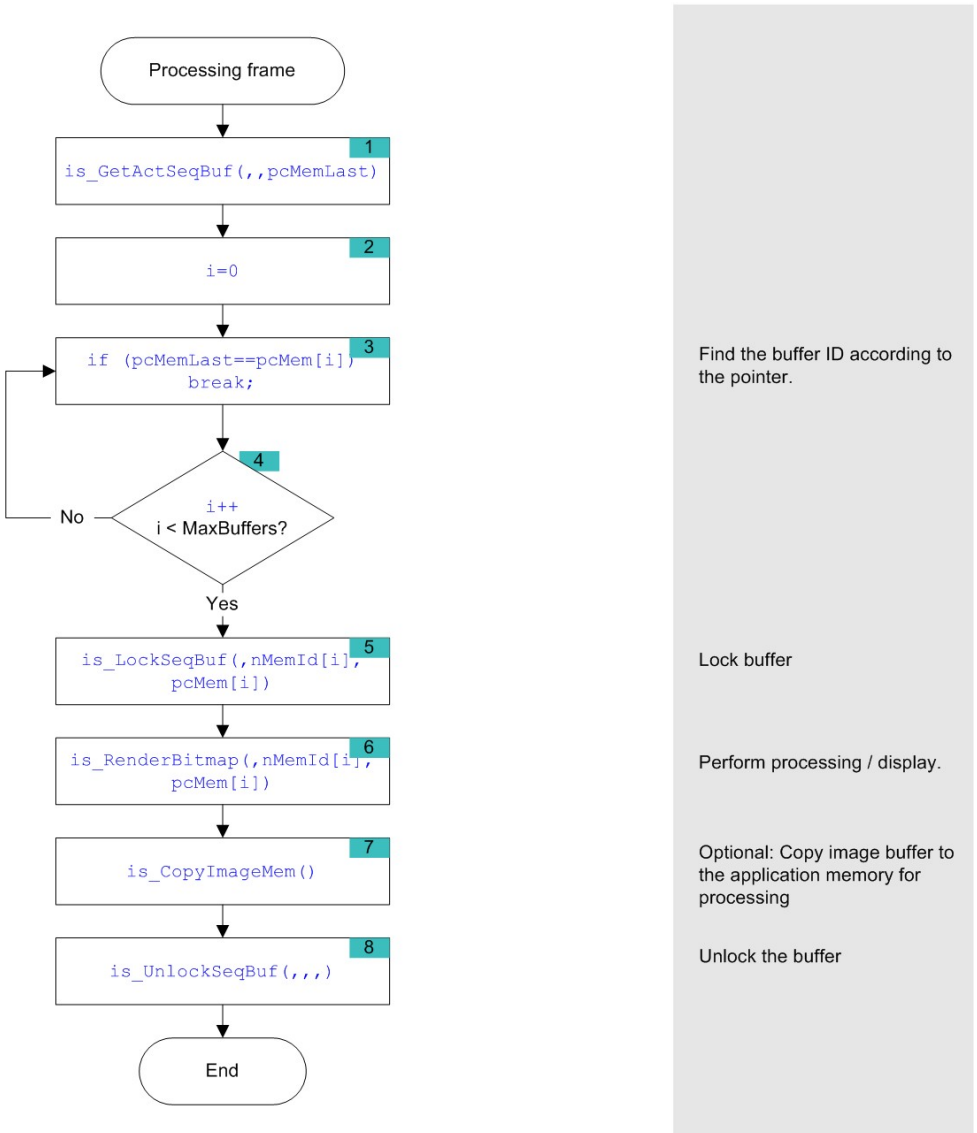
| | |
|---|---|
| is_SetDisplayMode() 322 | Selection of the display mode. |

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

| | |
|---|---|
| is_RenderBitmap() 297 | Outputs the contents of the active image memory to a window. |
| is_SetDisplayPos() 325 | 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 [134] that best meets your application requirements.
- Using events or messages [141], 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. [132]

### 4.2.3.1 Image Capture Modes

For more information on the capture modes of the DCx Cameras see also Camera basics: Freerun [17] and Camera basics: Trigger [19].

**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()` [327] before activating freerun mode.

- Single frame mode (snap mode)
  When `is_FreezeVideo()` [223] 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()` [177] 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 [487] chapter.

The trigger mode is selected using `is_SetExternalTrigger()` [327].

- Software trigger mode
  When this mode is enabled, calling `is_FreezeVideo()` [223] immediately triggers the capture of an image and then transfers the image to the PC. If `is_CaptureVideo()` [177] 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()` [223] [223] 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()` [177], 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 cameras that you are operating in the fast live mode. Not all camera models support this mode (see `is_SetExternalTrigger()` [327]).

**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 327 (OFF)<br>is_CaptureVideo() 177 | X | | Freely selectable |
| | Software | is_SetExternalTrigger 327 (SOFTWARE)<br>is_CaptureVideo() 177 | X | X | Depending on exposure time and trigger delay |
| | Hardware | is_SetExternalTrigger 327 (e.g. HI_LO)<br>is_CaptureVideo() 177 | X | X | Depending on exposure time and trigger delay |
| | Freerun sync. | is_SetExternalTrigger 327 (e.g. HI_LO_SYNC)<br>is_CaptureVideo() 177 | X | | Freely selectable |
| **Single frame** | Off | is_SetExternalTrigger 327 (OFF)<br>is_FreezeVideo() 223 | | | Freely selectable |
| | Software | is_SetExternalTrigger 327 (SOFTWARE)<br>is_FreezeVideo() 223 | X | X | Depending on exposure time and trigger delay |
| | Hardware | is_SetExternalTrigger 327 (e.g. HI_LO)<br>is_FreezeVideo() 223 | X | X | Depending on exposure time and trigger delay |

**Timeout values for image capture**

When you call is_FreezeVideo() 223 or is_CaptureVideo() 177, 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() 326). Information on the error cause can be queried using is_CaptureStatus() 386.

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`<br>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`<br>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()` 352); it is at least 40 ms.

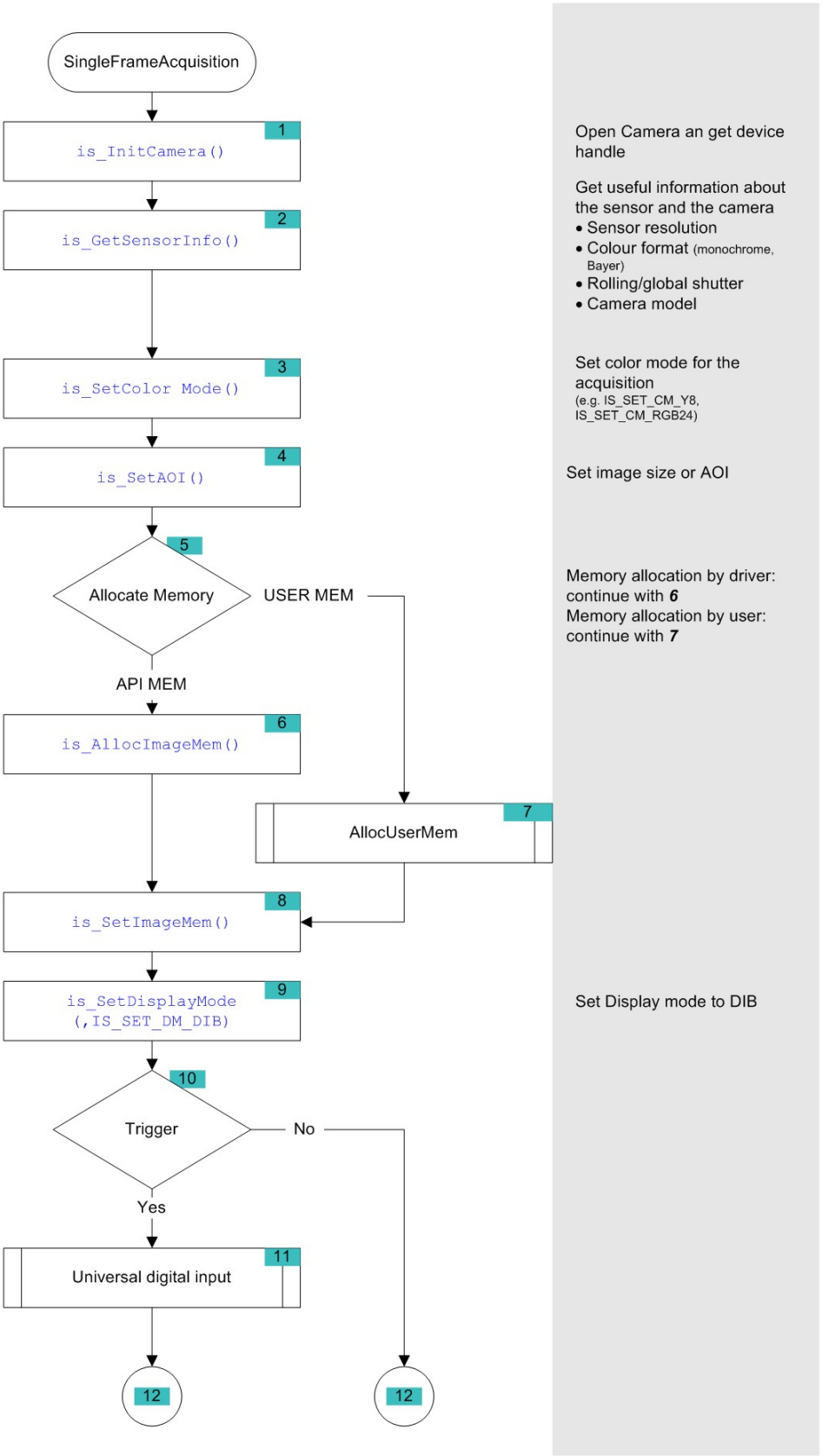
*3 The default value of the uc480 API is 60 s. User-defined values can be set using the `is_SetTimeout()` 350 function.

**Function list**

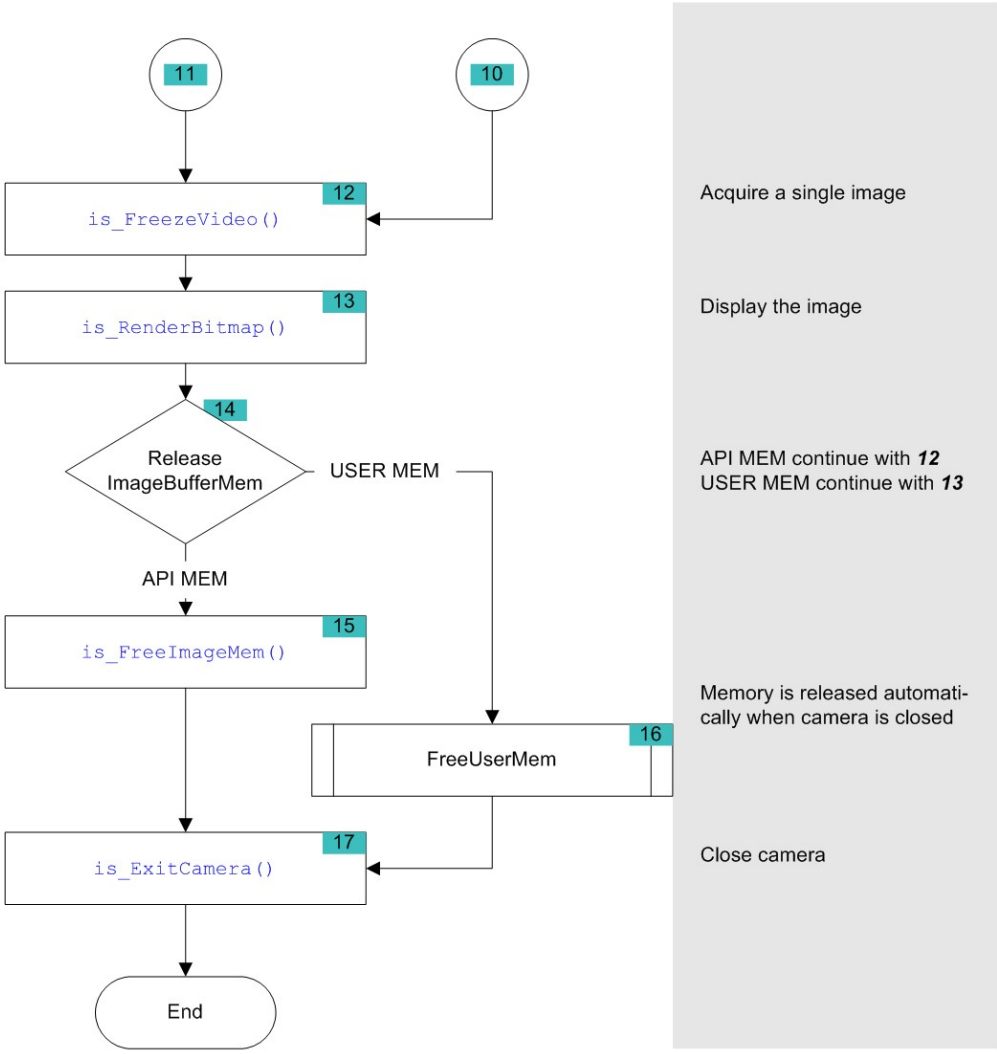
| | |
|---|---|
| `is_CaptureVideo()` 177 | Captures a live video. |
| `is_FreezeVideo()` 223 | Captures an image and writes it to the active image memory. |
| `is_ForceTrigger()` 221 | Forces image capture in hardware trigger mode. |
| `is_HasVideoStarted()` 259 | Returns whether the capture process has been started or not. |
| `is_IsVideoFinish()` 288 | Returns whether the capture process has been terminated or not. |
| `is_SetSensorTestImage()` 345 | Enables test image output from sensor (all cameras). |
| `is_StopLiveVideo()` 353 | Terminates the capturing process (live video or single frame). |

**Flowchart: Single Capture**

Click in the figure to get help on the functions.

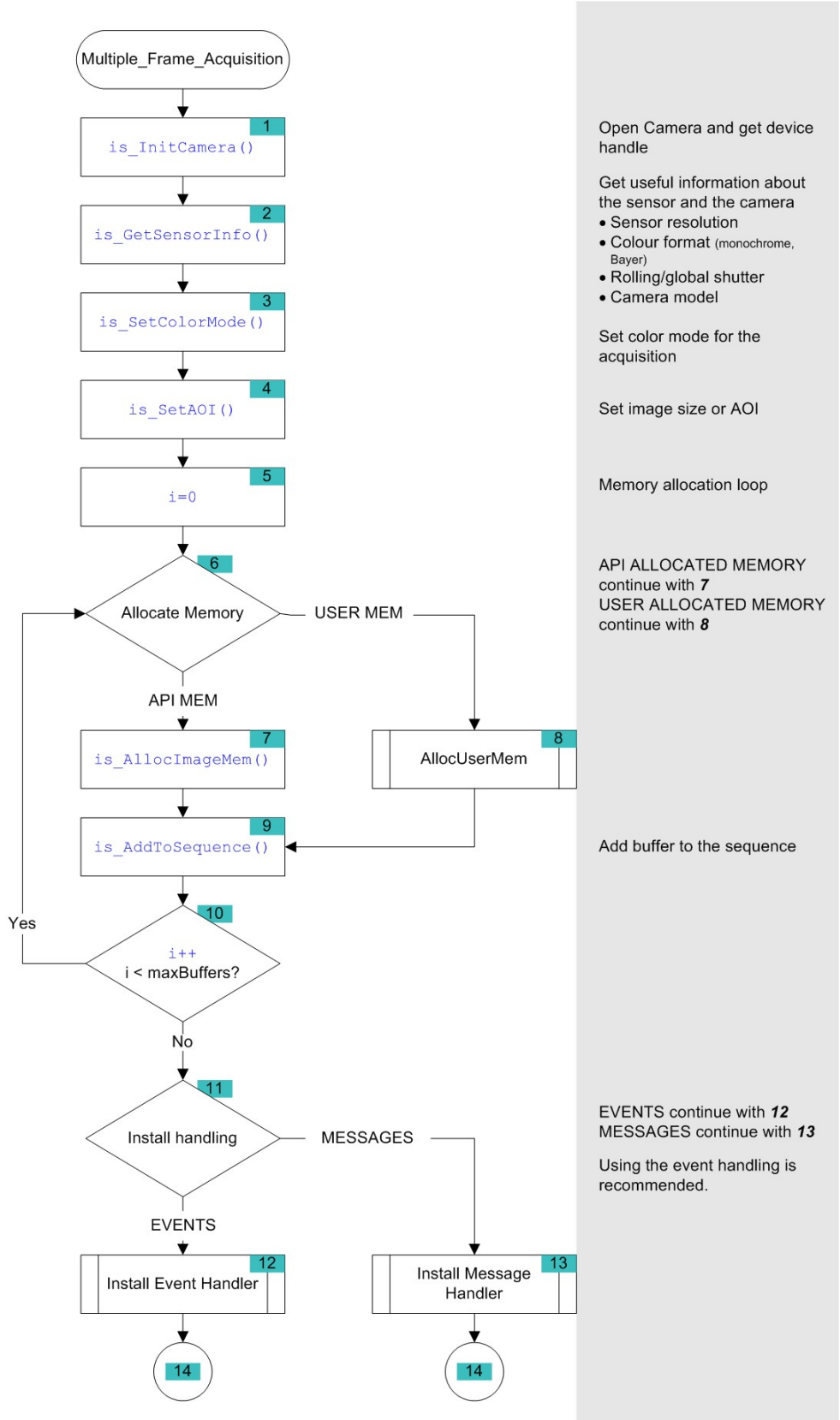



*Flowchart - Single Capture (1 of 2)*

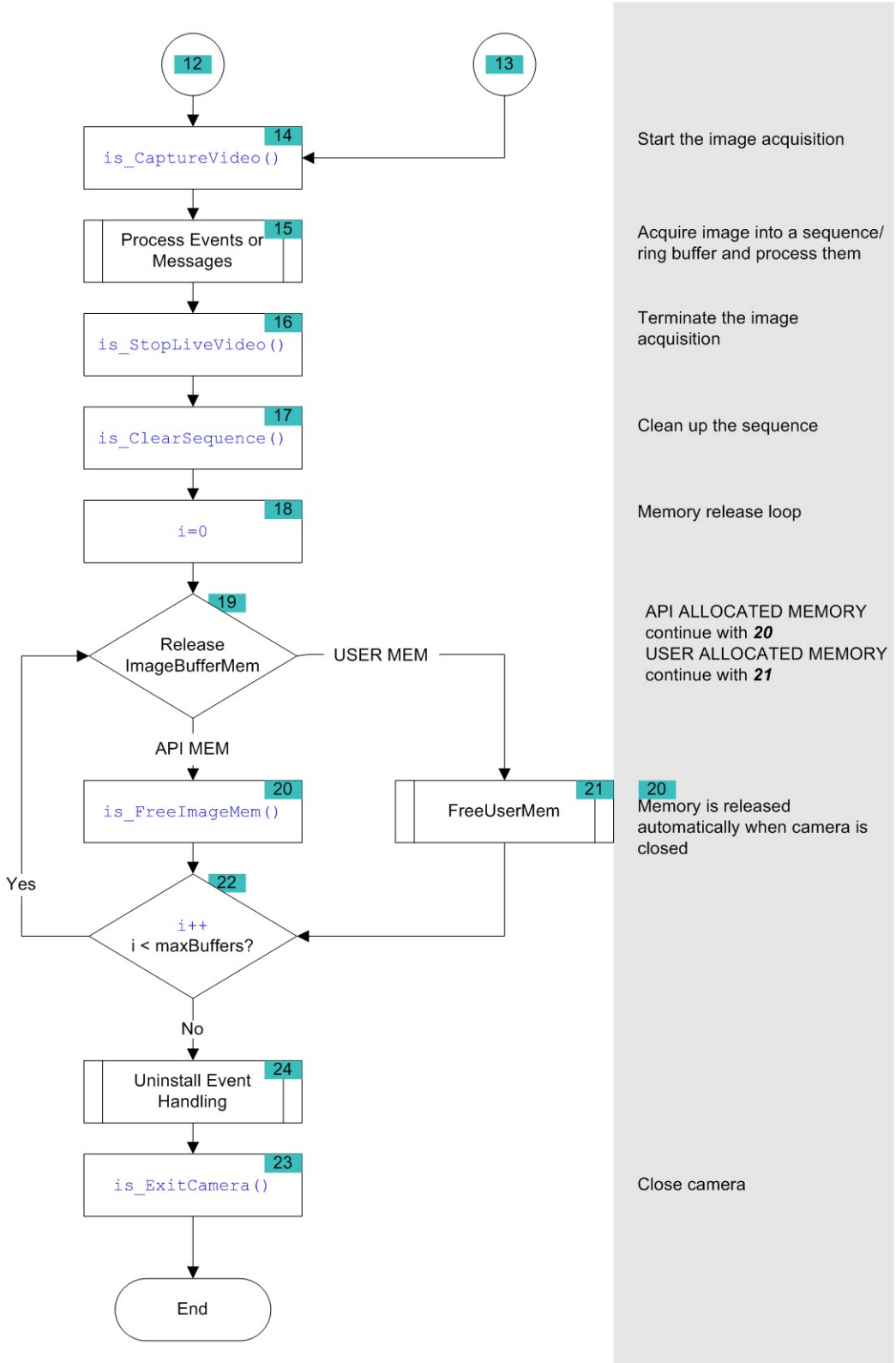



*Flowchart - Single Capture (2 of 2)*

**Flowchart: Sequence Capture**




*Flowchart - Capturing a frame sequence (1 of 2)*




*Flowchart - Capturing a frame sequence (2 of 2)*

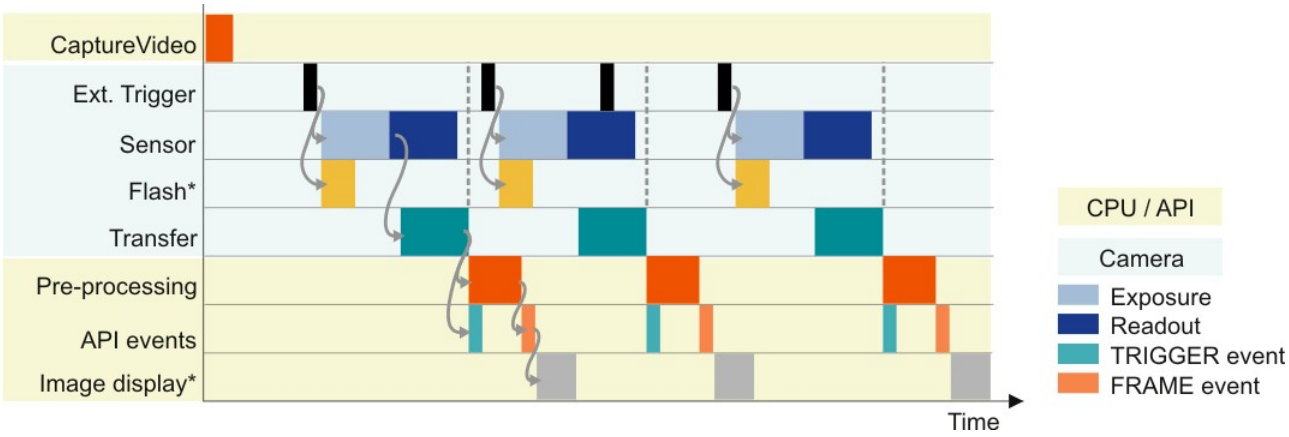
### 4.2.3.2 Event / Message Handling

**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() 327 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**

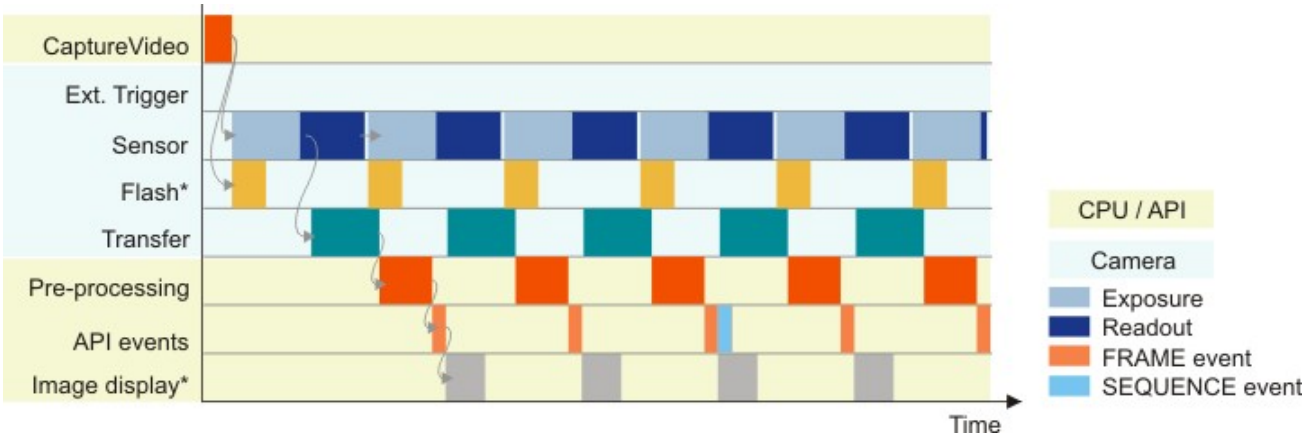
The following illustrations show a schematic view of the image capture sequence. The sensor exposure and readout times and the transmission times depend on the camera settings. The pre-processing time depends on the API functions you are using (e.g. color conversion, edge enhancement).




* Optional function. The start time and duration of the flash signal are defined by the "Flash delay" and "Duration" parameters (see is_IO() 280).

**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() 177 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() 156).



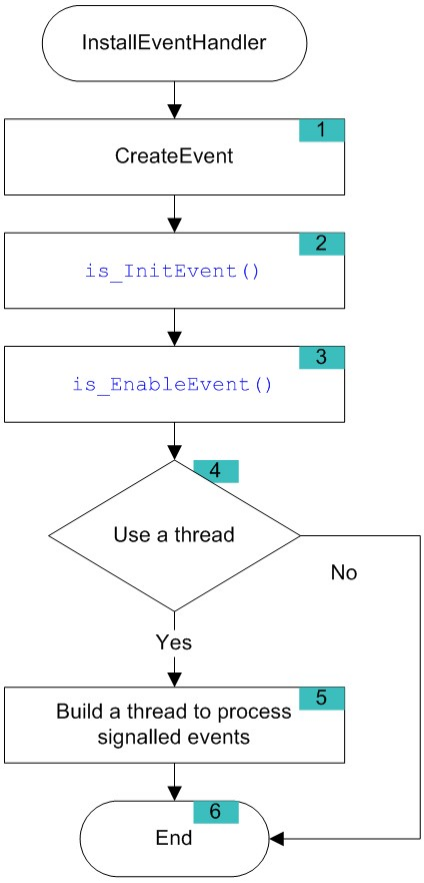

* Optional function. The start time and duration of the flash signal are defined by the "Flash delay" and "Duration" parameters (see is_IO() 280).

### Function List

| | |
|---|---|
| is_DisableEvent() 205 | Disables a single event object. |
| is_EnableEvent() 209 | Enables a single event object. |
| is_EnableMessage() 211 | Turns the Windows messages on/off. |
| is_ExitEvent() 214 | Closes the event handler (Windows only) |
| is_InitEvent() 276 | Initializes the event handler (Windows only) |
| is_EnableAutoExit() 208 | Automatically releases the camera resources when the camera is disconnected from the PC. |
| is_WaitEvent() 355 | 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

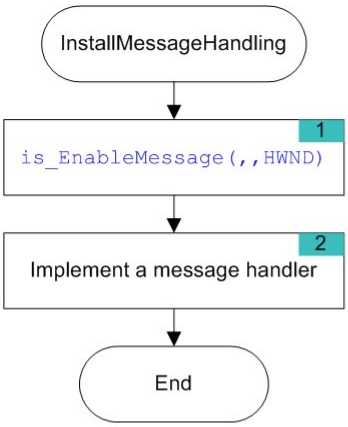
Initialize the uc480 event handling. The event handle from CreateEvent is required.

Enable a specific event.

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

Windows functions:
WaitForSingleObject()
WaitForMultipleObjects()

**Flowchart: Enabling Messages**

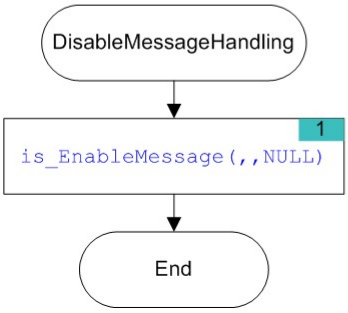



MS Windows only

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 144: Using these functions, you can make settings for the camera and for image capture and preprocessing.
- The DCxCamera's automatic image control 147 features allow automatically adjusting image brightness and image color to changing ambient conditions.
- Image preprocessing 148: These functions specify e.g. how color images are processed after image capture.
- Querying the camera status 148: With these functions, you can query additional useful information on the camera status.
- Using the camera EEPROM 148: 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() 180 | Sets the color temperature |
| is_Exposure() 216 | Returns the adjustable exposure range. |
| is_GetFramesPerSecond() 240 | Returns the current frame rate in live mode. |
| is_GetFrameTimeRange() 241 | Returns the adjustable frame rate range. |
| is_PixelClock() 294 | Returns the adjustable pixel clock range. |
| is_SetAutoParameter() 303 | Enables/disables automatic imaging functions. |
| is_Blacklevel() 170 | Turns black level correction on / off. |
| is_Exposure() 216 | Sets the exposure time. |
| is_SetFrameRate() 329 | Sets the frame rate. |
| is_SetGainBoost() 331 | Sets additional sensor hardware gain boost. |
| is_SetGamma() 332 | Sets the gamma value (digital post-processing). |
| is_SetHardwareGain() 333 | Enables the sensor hardware gain. |
| is_SetHWGainFactor() 335 | Sets the sensor hardware gain factor. |
| is_PixelClock() 294 | Sets the pixel clock frequency. |
| is_ResetToDefault() 299 | 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() 267 | Sets a predefined image size |
| is_AOI() 159 | Sets the size and position of an area of interest (AOI) or of a reference AOI for auto imaging functions. |
| is_SetBinning() 310 | Sets the binning modes. |
| is_SetRopEffect() 340 | Makes real-time geometry changes to an image (Rop = raster operation) |
| is_SetSensorScaler() 343 | Scales the image in the camera |
| is_SetSubSampling() 347 | Sets the subsampling modes. |

**Processing image data**

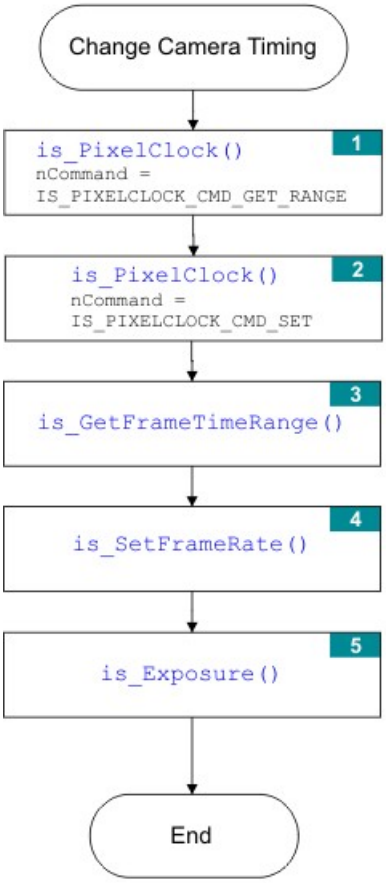
The following set of functions refers to the further processing of image data in the PC:

| | |
|---|---|
| is_GetColorDepth() 237 | Determines the desktop color mode set in the graphics card. |
| is_GetTimeout() 256 | Returns the user-defined timeout values. |

| | |
|---|---|
| is_HotPixel() 260 | Enables and configures the hot pixel correction. |
| is_SetColorConverter() 315 | Selects Bayer conversion mode. |
| is_SetColorCorrection() 317 | Sets color correction. |
| is_SetColorMode() 319 | Selects a color mode. |
| is_Convert() 188 | Conversion parameters for raw Bayer conversion. |
| is_EdgeEnhancement() 206 | Sets edge enhancement. |
| is_SetSaturation() 342 | Sets the image saturation (digital post-processing). |
| is_SetSensorTestImage() 345 | Enables test image output from sensor. |
| is_SetTimeout() 350 | 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.

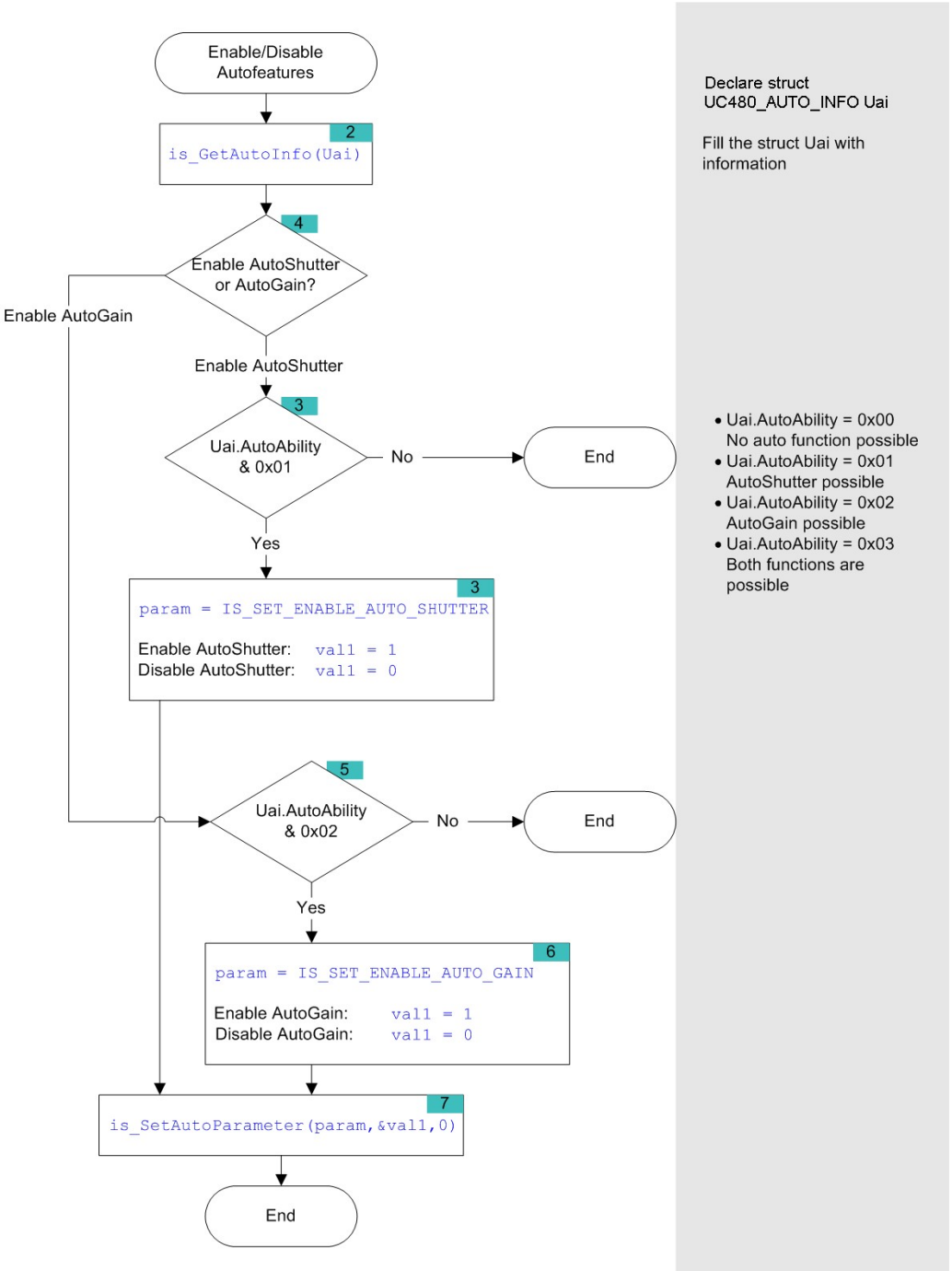
### 4.2.4.2 Automatic Image Control

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()` 303 SDK function.

For more information on the automatic image control see Camera basics: Automatic image control 45.

**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) 26).

| | |
|---|---|
| is_Convert() 188 | Converts a Bayer raw image into the desired output format |
| is_GetColorConverter() 236 | Returns the currently set Bayer conversion mode |
| is_SetBayerConversion() 420 | Sets the algorithm for Bayer conversion |
| is_SetColorConverter() 315 | 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() 235 | Read out current hardware LUT |
| is_GetImageHistogram() 242 | 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() 172 | Returns the event counters and other information.<br>Enables standby mode. |
| is_GetAutoInfo() 227 | Returns status information on the auto features. |
| is_GetCameraList() 233 | Returns information on all connected cameras. |
| is_GetCameraType() 385 | Returns the camera type. |
| is_CaptureStatus() 174 | Displays information on errors that have occurred. |
| is_GetError() 239 | Displays errors that have occurred. |
| is_GetUsedbandwidth() 257 | Returns the bus bandwidth (in Mbyte/s) currently used by all initialized or selected cameras. |
| is_GetVsyncCount() 258 | Returns the VSYNC counter. It will be incremented by 1 each time the sensor starts capturing an image. |
| is_SetErrorReport() 326 | 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() 231 | Returns the factory-set information (e.g. revision information for the individual DCxCamera components). |
| is_GetSensorInfo() 251 | Returns the sensor information. |
| is_ReadEEPROM() 296 | Reads out the writable data area of the EEPROM. |
| is_WriteEEPROM() 358 | Writes user data to the EEPROM. |

## 4.2.5 Saving Images and Videos

Using the uc480 API, you can

- Save and load single frames 149
- Capture an AVI frame sequence 149

### 4.2.5.1 Saving and Loading Single Frames

With the `is_ImageFile()` 264 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()` 369 | Initializes the AVI interface. |
| `isavi_ExitAVI()` 362 | Terminates and closes the AVI interface. |
| `isavi_OpenAVI()` 371 | Opens an AVI file for capturing. |
| `isavi_CloseAVI()` 360 | Closes an AVI file. |
| `isavi_GetAVIFileName()` 364 | Returns the name of the current AVI file. |

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

| | |
|---|---|
| `isavi_SetFrameRate()` 374 | Sets the frame rate of the AVI video. |
| `isavi_SetImageQuality()` 375 | Sets the compression level/image quality of the AVI video. |
| `isavi_SetImageSize()` 376 | 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()` 377 | Starts AVI recording. |
| `isavi_AddFrame()` 359 | Adds a compressed image to the AVI file. |
| `isavi_StopAVI()` 378 | Stops AVI recording. |

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

| | |
|---|---|
| `isavi_GetAVISize()` 366 | Returns the size of the current AVI file. |
| `isavi_GetnCompressedFrames()` 367 | Returns the number of frames in the current AVI file. |
| `isavi_GetnLostFrames()` 368 | Returns the number of frames that have been discarded so far. |
| `isavi_ResetFrameCounters()` 373 | Resets the counters for discarded and saved frames to 0. |

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

| | |
|---|---|
| `isavi_DisableEvent()` 360 | Disables a AVI event. |
| `isavi_EnableEvent()` 361 | Enables a AVI-Event. |
| `isavi_ExitEvent()` 363 | Turns off AVI event handling. |
| `isavi_InitEvent()` 370 | 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:

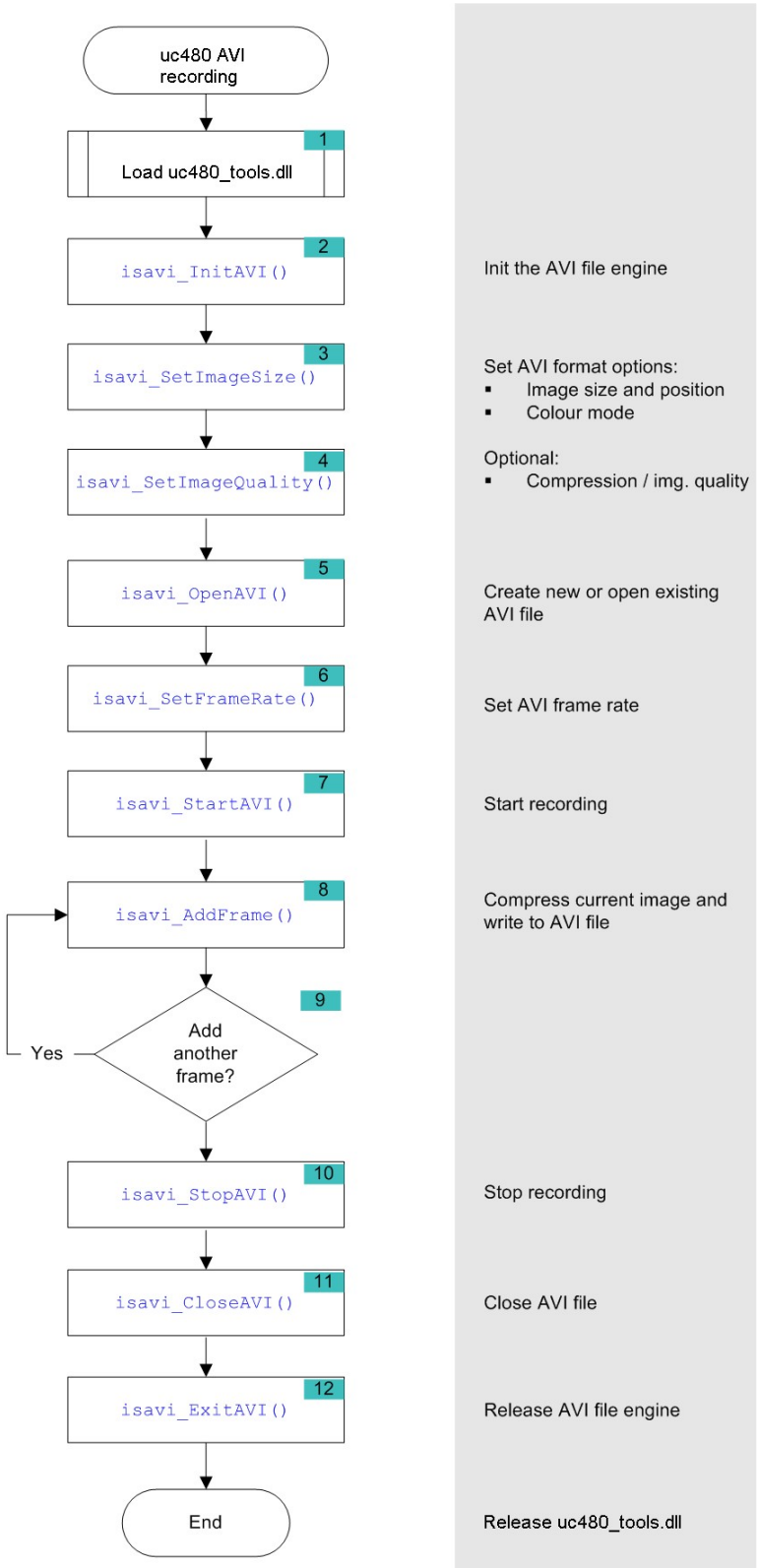
- Open the uc480 installation directory (default: `C:\Program Files\Thorlabs\DCx Cameras\Tools32` or `C:\Program Files\Thorlabs\DCx Cameras\Tools64` ).
- Right-click the `uc480Mjpeg.inf` (`uc480Mjpeg_64.inf`) file.
- Select "Install". The codec is installed automatically.

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

**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 152: 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.

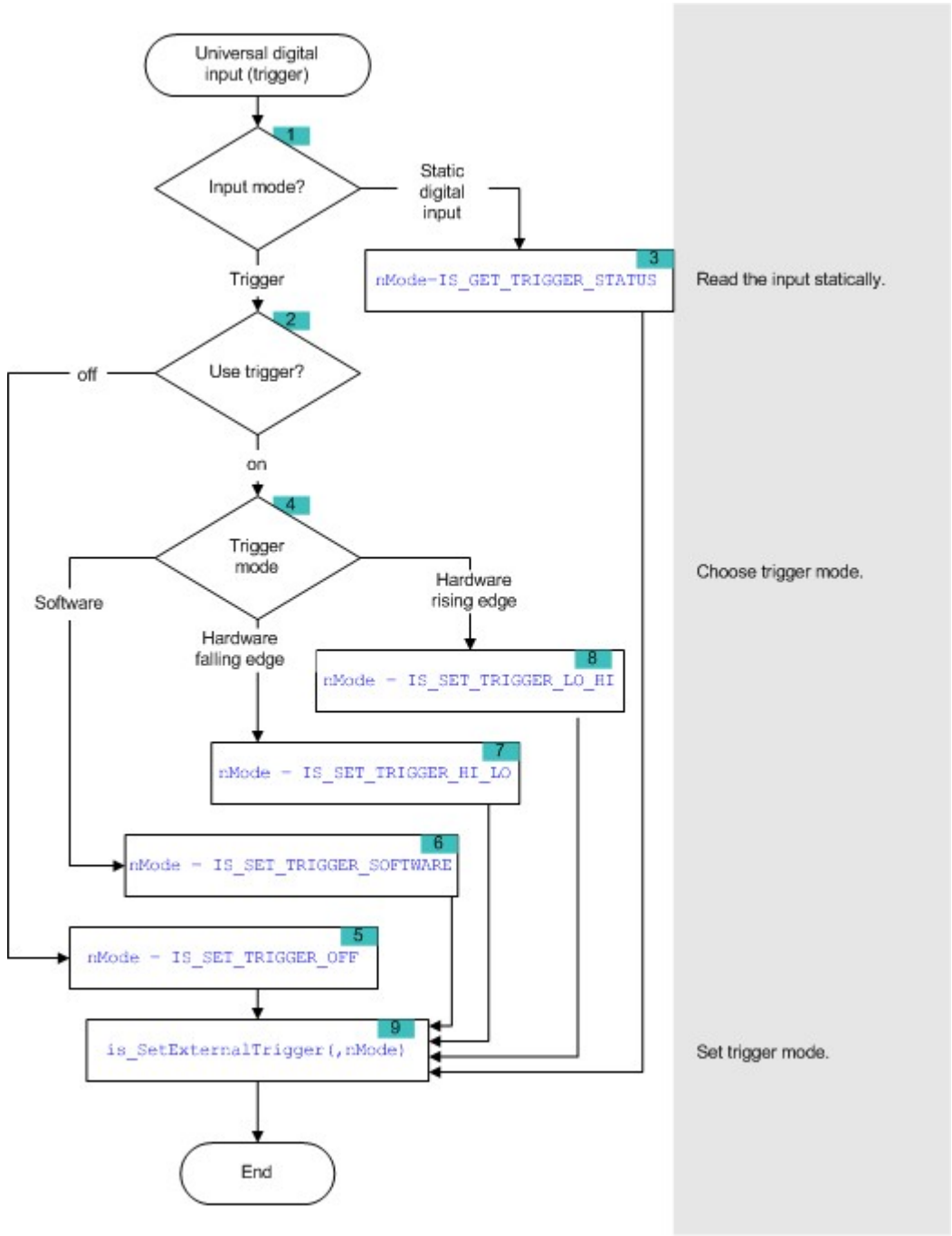
| | |
|---|---|
| is_SetExternalTrigger() 327 | Enables the digital input for trigger operation or returns the applied signal level. |
| is_IO() 280 | Sets the digital output for flash control or a static output level. |
| is_IO() 280 | Sets the delay and power-on time of the flash output. |
| is_SetTriggerDelay() 352 | Sets the trigger signal delay time. |
| is_IO() 280 | Determines the delay and power-on times of the flash output to obtain a global shutter effect when using rolling shutter sensors. |
| is_ForceTrigger() 221 | Simulates a trigger signal in hardware trigger mode. |

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

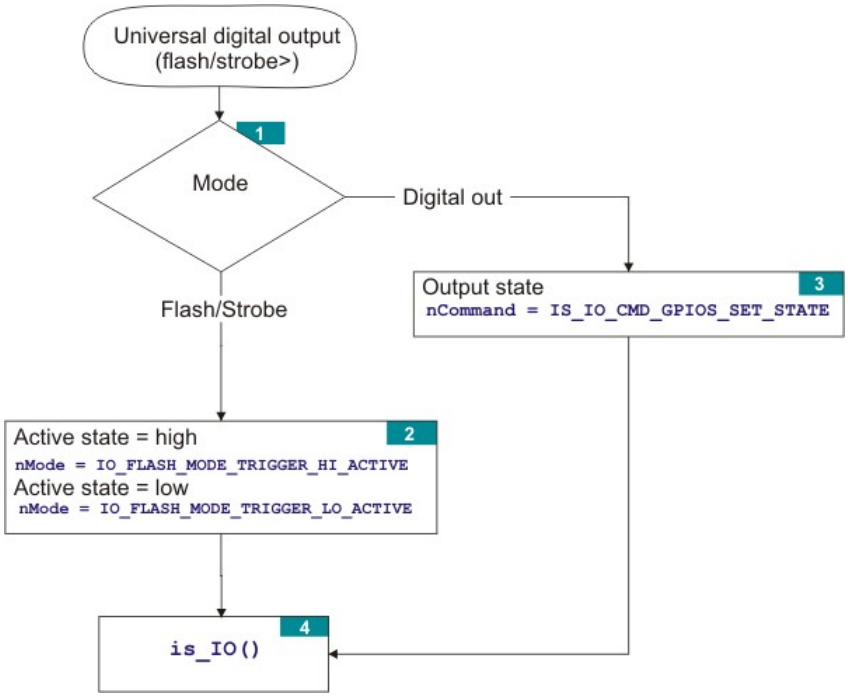
| | |
|---|---|
| is_IO() 280 | Sets the additional digital outputs (GPIO). |
| is_IO() 280 | Defines each port as a digital input or output (GPIO). |
| is_IO() 280 | Toggles the color of the status LED for DCU22x and DCC1240x cameras. |

**Flowchart: Digital input**

Click in the figure to get help on the functions.



**Flowchart: Digital output**



## 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<br>USB 3.0 | USB 2.0<br>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` 451 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()` 239 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<br>USB 3.0 | USB 2.0<br>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()` 157. Using the `is_SetAllocatedImageMem()` 301 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()` 157
- `is_InitImageQueue()` 278
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

### Sample programs

- uc480Sequence (C++)

## 4.3.2 is_AllocImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 248 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()` 337.

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()` 222.

**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 502 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() 222
- is_AddToSequence() 156
- is_SetImageMem() 337
- is_SetAllocatedImageMem() 301
- is_GetColorDepth() 237
- is_GetImgMemPitch() 248

## 4.3.3 is_AOI

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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) 34 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()` 414
- `is_SetImageAOI()` 434
- `is_SetImageSize()` 438
- `is_SetImagePos()` 435

See also Obsolete functions 379

**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()` 260).
- 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()` 157).
- 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:

- o is_SetFrameRate() 329
- o is_Exposure() 216
- o If you are using the flash function: is_IO() 280

**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. 460

**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 112). 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 113). 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 162 structure. In the version 4.20 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 164<br>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 164 |
| IS_AOI_IMAGE_GET_AOI | Returns the AOI in an IS_RECT object.<br>Sample 2 for AOI 164 |
| IS_AOI_IMAGE_SET_POS | Sets the AOI position by using an object of the IS_POINT_2D type.<br>IS_AOI_IMAGE_POS_ABSOLUTE can be ORed here, as well. Sample for setting the AOI position 164 |
| 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 |

| hCam | Camera handle |
|---|---|
| | set for the X position. Sample 3 for AOI 164 |
| 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_SUPPORTED | 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.<br>The axes are passed in an UINT array:<br>• Array[0] - Array[3] = X1…X4<br>• Array[4] - Array[8] = Y1…Y4<br>Sample 1 for multi AOI 165 |
| 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.<br>Sample 2 fo multi AOI 165 |
| 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:<br>• array[0] - array[3] = X1…X4<br>• array[4] - array[8] = Y1…Y4<br>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.<br>Sample 3 for multi AOI 165 |
| IS_AOI_SEQUENCE_GET_SUPPORTED | Returns a bitmask with the supported AOIs (only DCC1240x/DCC3240x camera models)<br>Sample 1 for AOI sequence mode 165 |
| IS_AOI_SEQUENCE_SET_PARAMS | Sets the parameters 162 of AOI 2, 3 or 4 (only DCC1240x/DCC3240x camera models)<br>Sample 2 for sequence AOI mode 166 |
| IS_AOI_SEQUENCE_GET_PARAMS | Returns the parameters 162 of AOI 2, 3 or 4 (only DCC1240x/DCC3240x camera models)<br>Sample 2 for sequence AOI mode 166 |
| IS_AOI_SEQUENCE_SET_ENABLE | Set a bitmask defining which AOIs should be active (only DCC1240x/ DCC3240x camera models).<br>**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.<br>Sample 3 for sequence AOI mode 166 |

| | |
|---|---|
| hCam | Camera handle |
| IS_AOI_SEQUENCE_GET_ENABLE | Returns the bitmask (only DCC1240x/DCC3240x camera models)<br>Sample 3 for sequence AOI mode 166 |
| 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 |

**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 4.20) |
| INT | s32SubsamplingMode | Subsampling mode (not supported in version 4.20) |
| INT | s32DetachImageParameters | • 0 = every change of the exposure time and the master gain is copied from AOI 1 to the additional AOIs (default).<br>As a change of AOI 1 also reset the exposure time, this change is also transferred to AOI 2, 3 and 4.<br>• 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 4.20) |
| 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_api.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. |
| IS_TRIGGER_ACTIVATED | The function cannot be used because the camera is waiting for a trigger signal. |

**Related functions**

- `is_ImageFormat()` 267
- `is_SetBinning()` 310
- `is_SetSubSampling()` 347

### 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;
}
```

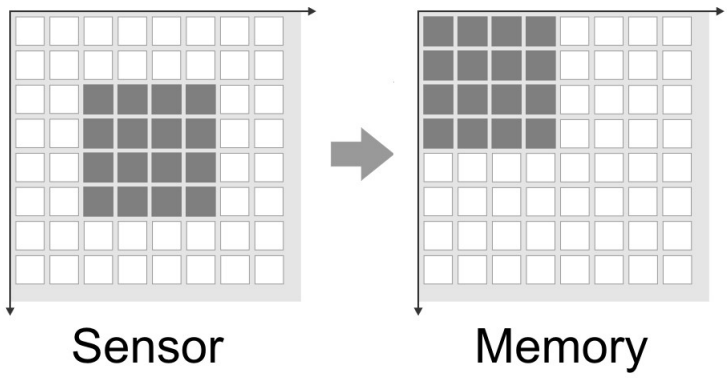
### Sample 3 for AOI

```
// 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
    }
}
```

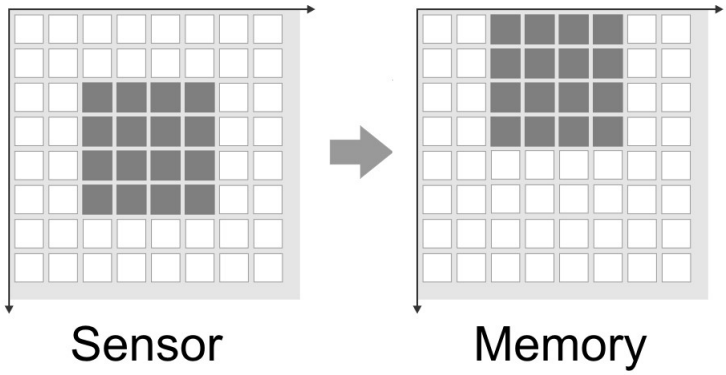
### Examples for setting absolute AOI positions in memory

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



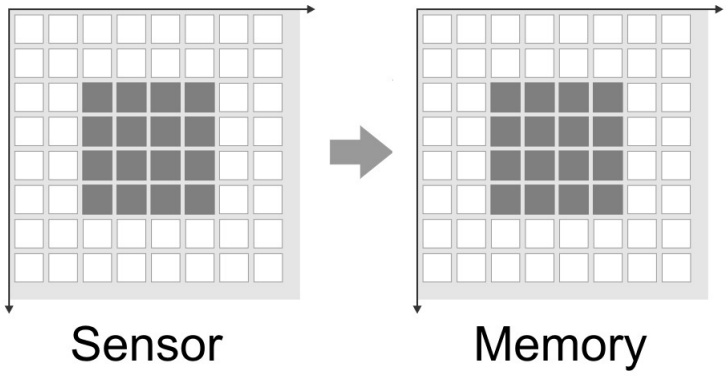

*AOI without absolute memory positioning*

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




*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));
```

## 4.3.4 is_AutoParameter

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 303 function. In version 4.20, 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 1 168)<br>• IS_AWB_GREYWORLD: 0x0001<br>• IS_AWB_COLOR_TEMPERATURE: 0x0002 |
| IS_AWB_CMD_GET_TYPE | Returns the current set type of the auto white balance (Example 2 168)<br>• IS_AWB_GREYWORLD: 0x0001<br>• IS_AWB_COLOR_TEMPERATURE: 0x0002 |
| IS_AWB_CMD_SET_TYPE | Sets the type of the auto white balance (Example 2 168)<br>• IS_AWB_GREYWORLD: 0x0001<br>• IS_AWB_COLOR_TEMPERATURE: 0x0002 |
| IS_AWB_CMD_GET_ENABLE | Returns if the auto white balance is enabled (Example 3 168)<br>• IS_AUTOPARAMETER_DISABLE: 0<br>• IS_AUTOPARAMETER_ENABLE: 1<br>• IS_AUTOPARAMETER_ENABLE_RUNONCE: 2 |
| IS_AWB_CMD_SET_ENABLE | Enables/Disables the auto white balance (Example 3 168)<br>• IS_AUTOPARAMETER_DISABLE: 0<br>• IS_AUTOPARAMETER_ENABLE: 1<br>• IS_AUTOPARAMETER_ENABLE_RUNONCE: 2 |
| IS_AWB_CMD_GET_SUPPORTED_RGB_COLOR_MODELS | Returns the supported color spaces for the auto white balance (Example 4 169)<br>• RGB_COLOR_MODEL_SRGB_D50: 0x0001<br>• RGB_COLOR_MODEL_SRGB_D65: 0x0002<br>• RGB_COLOR_MODEL_CIE_RGB_E: 0x0004<br>• RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008<br>• 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 5 169) |

| | |
|---|---|
| hCam | Camera handle |
| | • RGB_COLOR_MODEL_SRGB_D50: 0x0001<br>• RGB_COLOR_MODEL_SRGB_D65: 0x0002<br>• RGB_COLOR_MODEL_CIE_RGB_E: 0x0004<br>• RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008<br>• RGB_COLOR_MODEL_ADOBE_RGB_D65: 0x0010 |
| IS_AWB_CMD_SET_RGB_COLOR_MODEL | Sets the color space for the auto white balance (Example 5 169)<br>• RGB_COLOR_MODEL_SRGB_D50: 0x0001<br>• RGB_COLOR_MODEL_SRGB_D65: 0x0002<br>• RGB_COLOR_MODEL_CIE_RGB_E: 0x0004<br>• RGB_COLOR_MODEL_ECI_RGB_D50: 0x0008<br>• 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() 303

### 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
    }

    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

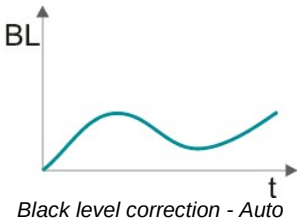
| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

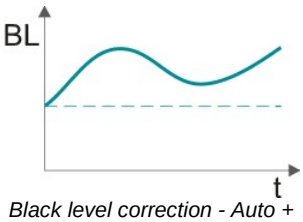
```
INT is_Blacklevel(HIDS hCam, UINT nCommand, void* pParam, UINT cbSizeOfParam)
```

**Beschreibung**

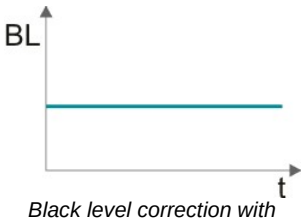
`is_Blacklevel()` controls the black level correction of the camera which might improve the image quality under certain circumstances. By default, the sensor adjusts the black level value for each pixel automatically. If the environment is very bright, it can be necessary to adjust the black level manually.



*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 1 171):<br>• `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.`<br>• `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 171) |
| `IS_BLACKLEVEL_CMD_GET_MODE` | Returns the current black level mode (Example2 171) |
| `IS_BLACKLEVEL_CMD_SET_MODE` | Sets the black level mode (Example 2 171)<br>• `IS_AUTO_BLACKLEVEL_OFF:` The automatic black level mode is switched off.<br>• `IS_AUTO_BLACKLEVEL_ON:` The automatic black level mode is switched on. |
| `IS_BLACKLEVEL_CMD_GET_OFFSET_DEFAULT` | Returns the default offset (Example 3 171) |
| `IS_BLACKLEVEL_CMD_GET_OFFSET_RANGE` | Returns the range of the offset (Example 3 171) |
| `IS_BLACKLEVEL_CMD_GET_OFFSET` | Returns the current offset (Example 3 171) |
| `IS_BLACKLEVEL_CMD_SET_OFFSET` | Sets the offset (Example 3 171) |
| `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));
```

## 4.3.6 is_CameraStatus

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 179, 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()` 174. |
| `IS_PARAMETER_SET_1` | Indicates whether parameter set 1 including camera settings is present on the camera (read-only). See also `is_ParameterSet()` 292.<br>Return values:<br>TRUE Parameter set 1 present<br>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()` 292.<br>Return values:<br>TRUE Parameter set 2 present<br>FALSE Parameter set 2 not present |
| `IS_STANDBY` | Sets the camera to standby mode.<br>Return values:<br>TRUE Camera changes to standby mode<br>FALSE The camera changes to freerun mode |
| `IS_STANDBY_SUPPORTED` | Queries whether the camera supports standby mode (read-only).<br>Return values:<br>TRUE The camera supports standby mode<br>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()` 386. |
| `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_api.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()` 231
- `is_GetError()` 239
- `is_SetErrorReport()` 326
- `is_SetTriggerCounter()` 351

## 4.3.7 is_CaptureStatus

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 379.

**Input parameters**

| | |
|---|---|
| hCam | Camera handle |
| nCommand | |
| `IS_CAPTURE_STATUS_INFO_CMD_GET` | Returns the CaptureStatus information (Example 1 175) |
| `IS_CAPTURE_STATUS_INFO_CMD_RESET` | Resets the CaptureStatus infomation (Example 2 176) |
| `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.<br>Possible cause/remedy<br>Not enough destination memory allocated or all destination buffers locked by the application.<br>• Release locked destination memory<br>• Allocate more destination memory<br>• 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.<br>Possible cause<br>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.<br>Possible cause/remedy<br>All destination buffers locked by the application<br>• Release locked destination memory<br>• Allocate more destination memory<br>• 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.<br>Possible cause/remedy<br>The computer takes too long to process the images in the uc480 API (e.g. color conversion)<br>• Reduce the frame rate so that there is more time to process the filled image memory of the driver<br>• 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.<br>Possible cause<br>The camera has been disconnected or closed. |
| IS_CAP_STATUS_USB_TRANSFER_FAILED | The image was not transferred over the USB bus.<br>Possible cause/remedy<br>Not enough free bandwidth on the USB bus for transferring the image<br>• Reduce the pixel clock frequency<br>• Operate fewer cameras simultaneously on a USB bus<br>• 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.<br>Possible cause/remedy<br>The selected timeout value is too low for image capture<br>• Reduce the exposure time<br>• 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**

| | |
|---|---|
| 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() 239
- is_CameraStatus() 172
- is_SetErrorReport() 326

**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_FAILED];
    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<br>USB 3.0 | USB 2.0<br>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()` 157 and designated as active image memory using `is_SetImageMem()` 337. Using `is_GetImageMem()` 247, 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()` 289 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 134 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 135 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 |
| 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_api.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() 223
- is_StopLiveVideo() 353
- is_SetExternalTrigger() 327
- is_ForceTrigger() 221
- is_SetTimeout() 350
- is_CaptureStatus() 174

**Sample programs**

- SimpleLive (C++)

## 4.3.9 is_ClearSequence

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 156. After a call of `is_ClearSequence()`, there is no more active image memory. To make an image memory the active memory, call `is_SetImageMem()` 337.

**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()` 156
- `is_FreeImageMem()` 222
- `is_SetImageMem()` 337

## 4.3.10 is_ColorTemperature

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 303/`is_AutoParameter()` 167.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| ⊟ `nCommand` | |
| **Setting the color space** | |
| `COLOR_TEMPERATURE_CMD_GET_SUPPORTED_RGB_COLOR_MODELS` | Returns the supported color spaces 182.<br>⊟ More details<br>`pParam:` Pointer to a bit mask of type `UINT`<br>The bit mask returns the supported modes, linked by logical ORs (see Color spaces 182 table).<br>`nSizeOfParam: 4` |
| `COLOR_TEMPERATURE_CMD_SET_RGB_COLOR_MODEL` | Sets a color space 182.<br>⊟ More details<br>`pParam:` Pointer to variable of type `UINT` that passes the value to be set.<br>`nSizeOfParam: 4` |
| `COLOR_TEMPERATURE_CMD_GET_RGB_COLOR_MODEL` | Returns the set color space 182. |

| hCam | Camera handle |
|---|---|
| | More details<br>pParam: Pointer to variable of type UINT returning the current value.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_RGB_COLOR_MODEL_DEFAULT | Returns the default color space.<br>More details<br>pParam: Pointer to variable of type UINT returning the default value.<br>nSizeOfParam: 4 |
| **Setting the color temperature** | |
| COLOR_TEMPERATURE_CMD_SET_TEMPERATURE | Sets a color temperature.<br>More details<br>pParam: Pointer to variable of type UINT that passes the value to be set.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_TEMPERATURE | Returns the set color temperature.<br>More details<br>pParam: Pointer to variable of type UINT returning the current value.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_MIN | Returns the minimum value for the color temperature.<br>More details<br>pParam: Pointer to variable of type UINT returning the minimum value.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_MAX | Returns the maximum value for the color temperature.<br>More details<br>pParam: Pointer to variable of type UINT returning the maximum value.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_INC | Returns the increment for setting the color temperature.<br>More details<br>pParam: Pointer to variable of type UINT returning the increment.<br>nSizeOfParam: 4 |
| COLOR_TEMPERATURE_CMD_GET_TEMPERATURE_DEFAULT | Returns the default value for the color temperature.<br>More details<br>pParam: Pointer to variable of type UINT returning the default value.<br>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() 333
- is_AutoParameter() 167
- is_SetAutoParameter() 303

## 4.3.11 is_Configuration

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 26, 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()` 292 function or via the corresponding function in the uc480 demo 80. 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.<br>⊟ Additional information<br>• `pParam:` Pointer to a `UINT` bitmask. The status flags from `CONFIGURATION_CAPS` 185 are returned in the bitmask.<br>• `nSizeOfParam:` 4<br>Example 1 186 |
| `IS_CONFIG_CPU_IDLE_STATES_CMD_GET_ENABLE` | Returns whether the current settings allow low power consumption operating states (unequal C0).<br>⊟ Additional information<br>• `pParam:` Pointer to a `UINT` bitmask. The status flags from `CONFIGURATION_SEL` 185 are returned in the bitmask.<br>• `nSizeOfParam:` 4<br>Example 2 186 |

| hCam | Camera handle |
|---|---|
| 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.<br>Additional information<br>• pParam: Pointer to a UINT variable, see CONFIGURATION_SEL 185.<br>• nSizeOfParam: 4<br>**Note:** To apply a new setting, you must close all open DCx Cameras and then reopen at least one camera.<br>Example 3 186 |
| 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.<br>Additional information<br>• pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL 185 are returned in the bitmask.<br>• nSizeOfParam: 4<br>Example 3 186 |
| IS_CONFIG_OPEN_MP_CMD_GET_ENABLE | Returns whether OpenMP support is enabled.<br>Additional information<br>• pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL 185 are returned in the bitmask.<br>• nSizeOfParam: 4<br>Example 4 187 |
| IS_CONFIG_OPEN_MP_CMD_SET_ENABLE | Enables OpenMP support.<br>Additional information<br>• pParam: Pointer to a UINT variable,<br>to enable: IS_CONFIG_OPEN_MP_ENABLE<br>to disable: IS_CONFIG_OPEN_MP_DISABLE<br>• nSizeOfParam: 4<br>**Note:** The settings are lost after the application is closed and must be set again the next time the camera is started.<br>Example 4 187 |
| IS_CONFIG_OPEN_MP_CMD_GET_ENABLE_DEFAULT | Returns the default setting for OpenMP support.<br>Additional information<br>• pParam: Pointer to a UINT bitmask. The status flags from CONFIGURATION_SEL 185 are returned in the bitmask.<br>• nSizeOfParam: 4<br>Example 4 187 |
| IS_CONFIG_INITIAL_PARAMETERSET_CMD_SET | Sets the parameter set to read and apply from the camera EEPROM when the camera is opened.<br>Additional information<br>• pParam: Pointer to a UINT variable,<br>IS_CONFIG_INITIAL_PARAMETERSET_NONE<br>IS_CONFIG_INITIAL_PARAMETERSET_1<br>IS_CONFIG_INITIAL_PARAMETERSET_2<br>• nSizeOfParam: 4<br>Example 5 187 |
| IS_CONFIG_INITIAL_PARAMETERSET_CMD_GET | Returns which parameter set will be read and applied from the camera EEPROM when the camera is opened. |

| | |
|---|---|
| hCam | Camera handle |
| | Additional information<br>• pParam: Pointer to a UINT variable,<br>IS_CONFIG_INITIAL_PARAMETERSET_NONE<br>IS_CONFIG_INITIAL_PARAMETERSET_1<br>IS_CONFIG_INITIAL_PARAMETERSET_2<br>• nSizeOfParam: 4<br>Example 5 187 |
| 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_VALUE | Set/recover processor operating states for 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() 292

### 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
        }
    }
}
```

### 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<br>USB 3.0 | USB 2.0<br>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 4.20 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()` 157 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 379

**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. |

**Contents of the BUFFER_CONVERSION_PARAMS structure**

| | | |
|---|---|---|
| char* | pSourceBuffer | Pointer to the raw Bayer buffer which was created with the is_AllocImageMem() 157 function. |
| char* | pDestBuffer | Pointer to the target buffer with the converted data which was created with the is_AllocImageMem() 157 function. |
| INT | nDestPixelFormat | Color mode of the target image; see is_SetColorMode() 319 for the possible modes |
| INT | nDestPixelConverter | Conversion mode of the target image; see is_SetColorConverter() 315 for the possible modes |
| INT | nDestGamma | Sets the gamma correction, see is_SetGamma() 332 |
| INT | nDestEdgeEnhancement | Sets the edge enhancement, see is_EdgeEnhancement() 206 |
| INT | nDestColorCorrectionMode | Sets the color correction, see is_SetColorCorrection() 317 |
| INT | nDestSaturationU | Sets the color saturation (saturation U), see is_SetSaturation() 342 |
| INT | nDestSaturationV | Sets the color saturation (saturation V),see is_SetSaturation() 342 |

**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() 319
- is_SetBayerConversion() 420

**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<br>USB 3.0 | USB 2.0<br>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_api.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()` 157
- `is_SetAllocatedImageMem()` 301

## 4.3.14 is_CopyImageMemLines

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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()` 157
- `is_SetAllocatedImageMem()` 301

## 4.3.15 is_DeviceFeature

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 33 (only monochrome models).
- Toggle between shutter modes, see Basics: Shutter methods. 30
- Control the Log mode, see uc480 Viewer: Shutter. 116

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 193 table (Example 1 194) |
| 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 2 194) |
| IS_DEVICE_FEATURE_CMD_GET_LOG_MODE | Returns the currect Log mode (Example 3 194) |
| 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 4 195) |
| IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE_RANGE | Returns the range of the manual value of the Log mode. |
| IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_VALUE | Returns the current manual value of the Log mode (Example 5 195) |
| IS_DEVICE_FEATURE_CMD_SET_LOG_MODE_MANUAL_VALUE | Sets the manual value of the Log mode |

| | |
|---|---|
| hCam | Camera handle |
| IS_DEVICE_FEATURE_CMD_GET_LOG_MODE_MANUAL_GAIN_DEFAULT | Returns the default settings for the manual gain for the Log mode (Example 6 195) |
| 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 7 195) |
| 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_api.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,
                            (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));
```

## 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 1 197) |
| `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)<br>See `IS_DEVICE_INFO_HEARTBEAT` 196 |
| `IS_DEVICE_INFO_CONTROL` | `infoDevControl` | Camera-related driver data<br>See `IS_DEVICE_INFO_CONTROL` 197 |
| `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<br>Bit 15: algebraic sign<br>Bit 14...11: filled according to algebraic sign<br>Bit 10...4: temperature (places before the decimal point)<br>Bit 3...0: temperature (places after the decimal point)<br>See the Ambient conditions 486 chapter for permissible temperature range. |
| `WORD` | `wLinkSpeed_Mb` | Transfer rate:<br>• `IS_USB_HIGH_SPEED = 480`<br>• `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_api.dll (API) and the driver file (uc480_usb.sys) do not match. |
| IS_SUCCESS | Function executed successfully |

**Related functions**

- is_GetCameraList() 233

**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;
}
```

## 4.3.17 is_DirectRenderer

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 202). When you load a bitmap image (`nMode = DR_LOAD_OVERLAY_FROM_FILE`), `pParam` passes the path to the file (see code samples 203). 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 56 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()` 322 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 supported by the graphics card.<br>More details<br>• IS_SET_DM_DIRECT3D:<br>Tests if Direct3D is suppported.<br>• IS_SET_DM_OPENGL:<br>Tests if OpenGL is supported.<br>Example 201 |
| DR_GET_OVERLAY_DC | Direct3D only: Returns the device context (DC) handle to the overlay area of the graphics card. |

| hCam | Camera handle |
|---|---|
| | More details<br>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.<br>Example 201 |
| DR_RELEASE_OVERLAY_DC | Direct3D only: Releases the device context (DC) handle.<br>More details<br>Using DR_RELEASE_OVERLAY_DC, you can release the DC handle and update the overlay data.<br>Example 201 |
| DR_GET_MAX_OVERLAY_SIZE | Returns the width x and height y of the maximum overlay area supported by the graphics card. Example 202 |
| DR_SET_OVERLAY_SIZE | Defines the size of the overlay area (default: current camera image size). Example 202 |
| 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 202 |
| DR_GET_OVERLAY_KEY_COLOR | Returns the RGB values of the current key color (default: black). Example 202 |
| DR_SET_OVERLAY_KEY_COLOR | Defines the RGB values of the key color.<br>More details<br>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.<br>The key color has no effect in semi-transparent mode!<br>Example 202 |
| DR_SHOW_OVERLAY | Enables overlay display on top of the current camera image. Example 202 |
| DR_HIDE_OVERLAY | Disables overlay display. Example 202 |
| 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 202 |
| 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 202 |
| DR_ENABLE_SEMI_TRANSPARENT_OVERLAY | Enables a semi-transparent display of the overlay area.<br>More details<br>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.<br>The key color has no effect in semi-transparent mode!<br>Example 202 |

| hCam | Camera handle |
|---|---|
| DR_DISABLE_SEMI_TRANSPARENT_OVERLAY | Disables the semi-transparent display of the overlay area. Example 202 |
| 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 203 |
| DR_SET_VSYNC_OFF | Disables image display synchronization. The image is displayed immediately. Example 203 |
| DR_SET_USER_SYNC | Direct3D only: Enables synchronization of the image display with a monitor pixel row specified by the user.<br>More details<br>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.<br>Example 203 |
| DR_GET_USER_SYNC_POSITION_RANGE | Direct3D only: Returns the minimum and maximum row position for DR_SET_USER_SYNC. Example 203 |
| 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 203 |
| DR_CLEAR_OVERLAY | Deletes the data of the overlay area by filling it with black color. Example 203 |
| DR_STEAL_NEXT_FRAME | Copies the next image to the active user memory (Steal function).<br>More details<br>Using the pParam parameter, you specify when the function should return:<br>• IS_WAIT: The function waits until the image save is complete.<br>• IS_DONT_WAIT: The function returns immediately.<br>Example 203 |
| DR_SET_STEAL_FORMAT | Defines the color format for the Steal function.<br>More details<br>For a list of all available color formats, see the function description for is_SetColorMode() 319. The default is IS_CM_BGRA8_PACKED (RGB 32).<br>Example 203 |
| DR_GET_STEAL_FORMAT | Returns the color format setting for the Steal function. Example 203 |
| DR_SET_HWND | Sets a new window handle for image output in Direct3D. Example 203 |
| DR_CHECK_COMPATIBILITY | Returns whether the graphics card supports the uc480 Direct3D functions. Example 203 |
| 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. |

### Return values

| | |
|---|---|
| When used with DR_CHECK_COMPATIBILITY | IS_DR_DEVICE_CAPS_INSUFFICIENT<br>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() 322
- is_SetColorMode() 319
- is_SetImageMem() 337
- is_RenderBitmap() 297

### 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);
```

### 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));
```

### 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

## 4.3.18 is_DisableEvent

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 209. See also `is_InitEvent()` 276.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `which` | ID of the event to be disabled. See also `is_EnableEvent()` 209. |

### 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()` 209
- Windows only: `is_InitEvent()` 276
- Windows only: `is_ExitEvent()` 214
- Linux only: `is_WaitEvent()` 355

## 4.3.19 is_EdgeEnhancement

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 379

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nCommand` | |
| `IS_EDGE_ENHANCEMENT_CMD_GET_RANGE` | Returns the range of the edge enhancement (Example 1 207) |
| `IS_EDGE_ENHANCEMENT_CMD_GET_DEFAULT` | Returns the standard value of the edge enhancement (Example 2 207) |
| `IS_EDGE_ENHANCEMENT_CMD_GET` | Returns the current set edge enhancement (Example 3 207) |
| `IS_EDGE_ENHANCEMENT_CMD_SET` | Sets the edge enhancement (Example 4 207)<br>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 |

**Related functions**

- `is_SetColorMode()` 319
- `is_SetColorConverter()` 315

### 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)
                             );
```

## 4.3.20 is_EnableAutoExit

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 213

## 4.3.21 is_EnableEvent

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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<br>• Event has to be declared by `is_InitEvent()` 276<br>• Event has to be activated by `is_EnableEvent()`<br>• You have to wait for the event in the application program by `WaitForSingleObject` or `WaitForMultipleObject`<br>• Event has to be deactivated by `is_DisableEvent()` 205<br>• Event has to be logged off by `is_ExitEvent()` 214 |
| **Linux** | • Event has to be provided by the uc480 API<br>• Event has to be activated by `is_EnableEvent()`<br>• You have to wait for the event by `is_WaitEvent()` 355<br>• Event has to be deactivated by `is_DisableEvent()` 205 |

**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()` 174.<br>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()` 273 and disconnected afterwards was reconnected. |
| `IS_SET_EVENT_EXTTRIG` | An image which was captured following the arrival of a trigger has been transferred completely.<br>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.<br>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. |

| hCam | Camera handle |
|---|---|
| IS_SET_EVENT_REMOVAL | A camera was removed.<br>This is independent of the device handle (hCam is ignored). |
| IS_SET_EVENT_REMOVE | A camera initialized with is_InitCamera() 273 was disconnected. |
| IS_SET_EVENT_SEQ | The sequence is completed. |
| IS_SET_EVENT_STATUS_CHANGED | **Linux only:**<br>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() 205
- Windows only: is_InitEvent() 276
- Windows only: is_ExitEvent() 214
- Linux only: is_WaitEvent() 355

### 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++)

## 4.3.22 is_EnableMessage

|  |  |
|---|---|
| USB 2.0<br>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()`. 174<br>The parameter IS_CAPTURE_STATUS replaces the previous parameter IS_TRANSFER_FAILED.<br>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.<br>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()` 273 was disconnected. |
| IS_DEVICE_RECONNECTED | A camera initialized with `is_InitCamera()` 273 and disconnected afterwards was reconnected. |
| 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). |

| | |
|---|---|
| hCam | Camera handle |
| 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.<br>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() 276

## 4.3.23 is_ExitCamera

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 157 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()` 273
- `is_SetDisplayMode()` 322
- `is_ExitCamera()` 213

See also Programming: Thread programming 453

**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_api.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()` 273
- `is_EnableAutoExit()` 208

## 4.3.24 is_ExitEvent

| | |
|---|---|
| USB 2.0<br>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()` 209 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `which` | ID of the event to be deleted. See also `is_EnbaleEvent()` 209. |

### 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()` 209
- `is_InitEvent()` 276

### Example

See also `is_ForceTrigger()` 221

## 4.3.25 is_ExitImageQueue

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

### Syntax

```
INT is_ExitImageQueue (HIDS hCam)
```

### Description

`is_ExitImageQueue()` deletes a queue which has been initialized with `is_InitImageQueue()` 278 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()` 179.

### 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()` 278
- `is_WaitForNextImage()` 356

## 4.3.26 is_Exposure

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 294
- `is_SetOptimalCameraTiming()` 338
- `is_SetFrameRate()` 329
- `is_AOI()` 159 (if the image size is changed)
- `is_SetSubSampling()` 347
- `is_SetBinning()` 310

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()` 177), any modification of the exposure time will only become effective when the next image but one is captured. In trigger mode (`is_SetExternalTrigger()` 327), the modification will be applied to the next image. See also the Applying new parameters 46 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 460 chapter.

**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 379

**Input parameters**

| hCam | Camera handle |
|---|---|
| nCommand | |
| IS_EXPOSURE_CMD_GET_CAPS | Returns the supported function modes.<br>More details<br>• pParam: Pointer to bit mask of type UINT<br>In the bit mask, the status flags from EXPOSURE_CAPS 219 are returned.<br>• nSizeOfParam: 4<br>Example 1 220 |
| IS_EXPOSURE_CMD_GET_EXPOSURE_DEFAULT | Returns the default setting for the exposure time.<br>More details<br>• pParam: Pointer to variable of type double returning the default value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_EXPOSURE | Returns the currently set exposure time (in ms).<br>More details<br>• pParam: Pointer to variable of type double returning the current value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MIN | Returns the minimum exposure time.<br>More details<br>• pParam: Pointer to variable of type double returning the minimum value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MAX | Returns the maximum exposure time.<br>More details<br>• pParam: Pointer to variable of type double returning the maximum value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_INC | Returns the exposure time increment.<br>More details<br>• pParam: Pointer to variable of type double returning the increment.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE | Returns the exposure time range.<br>More details<br>• pParam: Pointer to array of type double returning the minimum and maximum values and the increment (in exactly this order).<br>• nSizeOfParam: 24 |
| IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MIN | Returns the minimum exposure time in fine increments for some sensors.<br>More details<br>• pParam: Pointer to variable of type double returning |

| hCam | Camera handle |
|---|---|
| | the minimum value.<br>• nSizeOfParam: 8<br>Example 2 220 |
| IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_MAX | Returns the maximum exposure time in fine increments for some sensors.<br>More details<br>• pParam: Pointer to variable of type double returning the maximum value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE_INC | Returns the exposure time increment in fine increments for some sensors.<br>More details<br>• pParam: Pointer to variable of type double returning the increment.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_FINE_INCREMENT_RANGE | Returns the exposure time range in fine increments for some sensors.<br>More details<br>• pParam: Pointer to array of type double returning the minimum and maximum values and the increment (in exactly this order).<br>• nSizeOfParam: 24<br>Example 3 220 |
| IS_EXPOSURE_CMD_SET_EXPOSURE | Sets the exposure time (in ms).<br>More details<br>• pParam: Pointer to variable of type double that passes the value to be set.<br>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.<br>• nSizeOfParam: 8<br>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.<br>More information<br>• pParam: Pointer to variable of type double returning the minimum value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE_MAX | Returns the maximum long exposure time.<br>More information<br>• pParam: Pointer to variable of type double returning the maximum value.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE_INC | Returns the increments for long exposure.<br>More information<br>• pParam: Pointer to variable of the type double returning the increment.<br>• nSizeOfParam: 8 |
| IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_RANGE | Returns the value range for long exposure.<br>More information<br>• pParam: Pointer to an array of the type double returning the minmum and maximum values and the |

| | |
|---|---|
| hCam | Camera handle |
| | increment.<br>• nSizeOfParam: 24 |
| IS_EXPOSURE_CMD_GET_LONG_EXPOSURE_ENABLE | Returns the current settings for long exposure.<br>⊟ More information<br>pval1: returns the current setting |
| IS_EXPOSURE_CMD_SET_LONG_EXPOSURE_ENABLE | Enables/Disables long exposure.<br>⊟ More information<br>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 |
| 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_api.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() 329
- is_PixelClock() 294
- is_SetOptimalCameraTiming() 338
- is_SetAutoParameter() 303
- is_AutoParameter() 167
- is_SetHardwareGain() 333

### 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];
}
```

## 4.3.27 is_ForceTrigger

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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() 223
- is_CaptureVideo() 177
- is_SetExternalTrigger() 327

### 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);
}
```

## 4.3.28 is_FreeImageMem

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 157 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()` 157 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_api.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()` 157

## 4.3.29 is_FreezeVideo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 327. A hardware triggered image acquisition can be cancelled using `is_StopLiveVideo()` 353 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 134 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 135 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_api.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() 259
- is_IsVideoFinish() 288
- is_SetExternalTrigger() 327
- is_ForceTrigger() 221
- is_CaptureVideo() 177
- is_SetTimeout() 350
- is_CaptureStatus() 174

**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#)

## 4.3.30 is_GetActiveImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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.

### 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_api.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()` 157
- `is_GetImageMem()` 247
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

## 4.3.31 is_GetActSeqBuf

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 157 function, but the running number from the order in which memory was allocated by the `is_AddToSequence()` 156 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pnNum` | Contains the number of the image memory currently used for image capturing.<br>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. Possibly the versions of the `uc480_api.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()` 156
- `is_GetImageMem()` 247

## 4.3.32 is_GetAutoInfo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 45 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()` 303.

**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 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:<br>0 = black<br>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 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_api.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()` 303

## 4.3.33 is_GetBusSpeed

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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" 70 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. |

## 4.3.34 is_GetCameraInfo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 233). 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()` 251 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:<br>• `IS_CAMERA_TYPE_UEYE_USB_SE:` DCU223x, DCU224x and DCC1240x<br>• `IS_CAMERA_TYPE_UEYE_USB_LE:` DCC1545M / DCC1645C<br>• `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()` 172

- is_GetSensorInfo() 251

## 4.3.35 is_GetCameraList

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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.<br>This value can be read out with `is_GetNumberOfCameras()` 249. |
| `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.<br>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_api.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() 249

### 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;
}
```

## 4.3.36 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 |

## 4.3.37 is_GetColorConverter

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 502 section.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ColorMode` | Color mode for which the converter is to be returned<br>For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats 502 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:<br>`IS_CONV_MODE_NONE`<br>`IS_CONV_MODE_SOFTWARE_3X3`<br>`IS_CONV_MODE_SOFTWARE_5X5`<br>`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()` 315
- `is_SetColorMode()` 319

## 4.3.38 is_GetColorDepth

|  |  |
|---|---|
| USB 2.0<br>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()` 319 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`.<br>For a list of all available color formats and the associated input parameters, see the Appendix: color and Memory Formats 502 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()` 319
- `is_AllocImageMem()` 157

## 4.3.39 is_GetDLLVersion

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

### Syntax

```
INT is_GetDLLVersion()
```

### Description

Returns the version of the `uc480_api.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()` 250

### 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);
```

## 4.3.40 is_GetError

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 174
- `is_SetErrorReport()` 326
- `is_CameraStatus()` 172

## 4.3.41 is_GetFramesPerSecond

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

### Syntax

```
INT is_GetFramesPerSecond (HIDS hCam, double* dblFPS)
```

### Description

In live capture mode started by `is_CaptureVideo()` 177, 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()` 241
- `is_SetFrameRate()` 329
- `is_PixelClock()` 294
- `is_Exposure()` 216

## 4.3.42 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. 17

**Note**

The use of the following functions will affect the frame duration:

- `is_PixelClock()` 294
- `is_SetOptimalCameraTiming()` 338
- `is_AOI()` 159 (if the image size is changed)
- `is_SetSubSampling()` 347
- `is_SetBinning()` 310

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()` 240
- `is_SetFrameRate()` 329
- `is_PixelClock()` 294
- `is_Exposure()` 216

## 4.3.43 is_GetImageHistogram

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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<br>For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats 502 section. |
| `pHistoMem` | Pointer to a `DWORD` array<br>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_api.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()` 319

### 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);
```

## 4.3.44 is_GetImageInfo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

### Syntax

```
INT is_GetImageInfo (HIDS hCam, INT nImageBufferID, UEYEIMAGEINFO* pImageInfo, INT nImageInfoSize)
```



### 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()` 289. 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) |
| `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` 245 below |
| `DWORD` | `dwIoStatus` | With DCC3240x Cameras: Returns the states of the digital I/Os at the time of image capture:<br>• Digital input (trigger): Pending signal<br>• GPIO as input: Pending signal<br>• GPIO as output: Set level |

| | | |
|---|---|---|
| | | With all other cameras, dwIoStatus is empty. See dwIOStatus 245 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.<br>This may happen when e.g.:<br>• more memory is allocated than the UC480IMAGEINFO structure needs<br>• nImageBufferID <= 0<br>• pImageInfo == NULL<br>• nImageInfoSize <= 0 |
| IS_NO_SUCCESS | General error message |
| IS_SUCCESS | Function executed successfully |

### Related functions

- is_CaptureStatus() 174
- is_LockSeqBuf() 289
- is_UnlockSeqBuf() 354
- is_SetImageMem() 337
- is_IO() 280

### 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++)

## 4.3.45 is_GetImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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()` 248
- `is_AllocImageMem()` 157
- `is_AddToSequence()` 156
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

### Sample programs

- uc480PixelPeek (C++)

## 4.3.46 is_GetImageMemPitch

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 157. 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_api.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()` 247
- `is_AllocImageMem()` 157
- `is_AddToSequence()` 156
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

## 4.3.47 is_GetNumberOfCameras

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 233
- `is_DeviceInfo()` 196

## 4.3.48 is_GetOsVersion

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 238

## 4.3.49 is_GetSensorInfo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 451 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.<br>`IS_COLORMODE_BAYER`<br>`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.<br>`TRUE` = global shutter<br>`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()` 231
- `is_CameraStatus()` 172

## 4.3.50 is_GetSensorScalerInfo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 343 |
| `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()` 343

## 4.3.51 is_GetSupportedTestImages

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 345.

**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 |

**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() 345
- is_GetTestImageValueRange() 255

## 4.3.52 is_GetTestImageValueRange

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 345.

### 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.<br>In this case, the `TestImageValueMin` and `TestImageValueMax` parameters are equal to 0. |
| `IS_SUCCESS` | Function executed successfully |

### Related functions

- `is_GetSupportedTestImages()` 253
- `is_SetSensorTestImage()` 345

## 4.3.53 is_GetTimeout

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 135 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()` 350
- `is_CaptureVideo()` 177
- `is_FreezeVideo()` 223
- `is_SetExternalTrigger()` 327

### Example

```
// Return user-defined timeout
UINT nTimeout;
INT ret = is_GetTimeout(hCam, IS_TRIGGER_TIMEOUT, &nTimeout);
```

## 4.3.54 is_GetUsedBandwidth

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 294

## 4.3.55 is_GetVsyncCount

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 240

## 4.3.56 is_HasVideoStarted

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 223 function was called with the `IS_DONT_WAIT` parameter.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pbo` | Returns the digitizing status:<br>0 = Image capturing has not started yet.<br>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()` 223
- `is_IsVideoFinish()` 288

## 4.3.57 is_HotPixel

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 28.

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 379

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nCommand` | |
| `IS_HOTPIXEL_DISABLE_CORRECTION` | Disables hot pixel correction (Example 1 262) |
| `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. |
| `IS_HOTPIXEL_GET_CORRECTION_MODE` | Returns the currently set hot pixel correction mode (Example 2 262) |
| `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):<br>• `IS_HOTPIXEL_ENABLE_CAMERA_CORRECTION:` Hot pixel correction is possible via the hot pixel list in the camera EEPROM. |

| | |
|---|---|
| hCam | Camera handle |
| | • IS_HOTPIXEL_ENABLE_SOFTWARE_USER_CORRECTION: Hot pixel correction is possible via the user-defined hot pixel list.<br>• 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 3 262) |
| 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<br>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 4 263) |
| IS_HOTPIXEL_LOAD_SOFTWARE_USER_LIST<br>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 5 263) |
| 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. |
| 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 6 263) |
| 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_api.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;
    }
}
```

### 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;
}
```

## 4.3.58 is_ImageFile

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 223 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 379

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| ⊟ `nCommand` | |
| `IS_IMAGE_FILE_CMD_LOAD` | Loads an image file (bmp, jpg, png) (Example 1 266)<br>The function can be used with UNICODE file names. |
| `IS_IMAGE_FILE_CMD_SAVE` | Saves an image file (bmp, jpg, png) (Example 2 266)<br>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).<br>If NULL is passed, the "Open file"/"Save as" dialog opens. |
| UINT | nFileType | File type to be saved:<br>• IS_IMG_BMP<br>• IS_IMG_JPG<br>• 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:<br>• 100 = maximum quality with minimum compression<br>• If the parameter is set to 0, the the default value of 75 is used.<br>For BMP the parameter is ignored. |
| char | ppcImageMem | **When loading:**<br>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() 222.<br>**When saving:**<br>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() 247
- is_SetImageMem() 337

### 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));
```

## 4.3.59 is_ImageFormat

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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" 269 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.<br>More details<br>You can query the number of entries in the list with `IMGFRMT_CMD_GET_NUM_ENTRIES`.<br>• `pParam`: Pointer to list of type `IMAGE_FORMAT_LIST` 268. The list must be preallocated as specified below.<br>• `nSizeOfParam`: Size of the list<br>Size of (`IMAGE_FORMAT_LIST` + (number of list entries - 1) *<br>Size of (`IMAGE_FORMAT_INFO`)) |
| `IMGFRMT_CMD_GET_NUM_ENTRIES` | Returns the number of entries in the list.<br>More details<br>• `pParam`: Pointer to variable of type `UINT` returning the number of list entries.<br>• `nSizeOfParam`: 4 |
| `IMGFRMT_CMD_SET_FORMAT` | Sets the desired image format.<br>More details<br>• `pParam`: Pointer to variable of type `UINT` passing the format ID 268 of the desired image format.<br>• `nSizeOfParam`: 4 |
| `IMGFRMT_CMD_GET_ARBITRARY_AOI_SUPPORTED` | Returns if the sensor supports a free selection of the area of interest (AOI).<br>More details<br>• `pParam`: Pointer to variable of type `UINT` indicating if free AOI selection is supported:<br>0 = No free AOI supported<br>1 = Free AOI supported<br>• `nSizeOfParam`: 4 |
| `pParam` | Pointer to a function parameter, whose function depends on `nCommand`. |

| | |
|---|---|
| hCam | Camera handle |
| 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.<br>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" 269 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 343). |
| 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 17 chapter.

**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. |
| 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_api.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() 159
- is_SetBinning() 310
- is_SetSubSampling() 347

### 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);
}
```

## 4.3.60 is_InitCamera

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 273
- `is_SetDisplayMode()` 322
- `is_ExitCamera()` 213

See also General: Thread programming 453

**Input parameters**

| | |
|---|---|
| `phCam` | Pointer to the camera handle<br>When you call this function, the pointer value has the following meaning:<br>0: The first available camera will be initialized or selected.<br>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()` 233 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).<br>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 275 below). |
| `hWnd` | Pointer to the window where the Direct3D image will be displayed<br>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 |
| 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_api.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() 213
- is_EnableAutoExit() 208
- is_GetCameraList() 233
- is_SetCameraID() 313
- is_GetCameraInfo() 231

### 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#)

## 4.3.61 is_InitEvent

| | |
|---|---|
| USB 2.0<br>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()` 209) |

**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()` 209
- `is_DisableEvent()` 205
- `is_ExitEvent()` 214

### 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++)

## 4.3.62 is_InitImageQueue

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 156 prior to calling is_InitImageQueue(). With `is_WaitForNextImage()` 356 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()` 226 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()` 215
- `is_WaitForNextImage()` 356
- `is_AddToSequence()` 156

### 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);
```

## 4.3.63 is_InquireImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 157 |
| `NID` | ID of the image memory as allocated by `is_AllocImageMem()` 157 |
| `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_api.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()` 157
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301
- `is_GetColorDepth()` 237

## 4.3.64 is_IO

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 19 and the additional digital outputs (GPIOs) of some DCx Camera models. For information on GPIO wiring, please refer to the Electrical Specifications 487 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 487 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 30, Digital input/output (trigger/flash) 47 and Operating modes 17.

**Attention**

**Accuracy of flash synchronization**

The following parameters have an influence on the camera's internal timing:

- Image geometry 144 (CMOS and CCD sensors)
- Pixel clock 294 (CMOS and CCD sensors)
- Exposure time 216 (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 379

**Input parameters**

| hCam | Camera handle |
|---|---|
| nCommand | |
| **GPIO** | |
| IS_IO_CMD_GPIOS_GET_SUPPORTED | Returns the supported GPIO ports (Example 1 284)<br>• 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 1 284) |
| IS_IO_CMD_GPIOS_GET_SUPPORTED_OUTPUTS | Returns the supported GPIO outputs (Example 1 284) |
| IS_IO_CMD_GPIOS_GET_DIRECTION | Returns the input/output mask of the GPIOs (Example 2 284) |
| IS_IO_CMD_GPIOS_SET_DIRECTION | Set the GPIO on input/output (Example 2 284)<br>• IO_FLASH_MODE_GPIO_1: Sets GPIO 1 as outout.<br>• 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 2 284) |
| IS_IO_CMD_GPIOS_SET_STATE | Sets the state of the GPIOs if they are defined as output (High, Low) (Example 2 284) |
| IS_IO_CMD_GPIOS_GET_CONFIGURATION | Returns the configuration of a GPIO port (Example 10 287) |
| IS_IO_CMD_GPIOS_SET_CONFIGURATION | Sets the configuration of a GPIO port (Example 11 287, Example 12 287) |
| **Flash** | |
| IS_IO_CMD_FLASH_GET_SUPPORTED_GPIOS | Returns the GPIOs which can be used for flash output (Example 6 285) |
| IS_IO_CMD_FLASH_GET_MODE | Returns the current flash mode, see below (Example 7 285) |
| IS_IO_CMD_FLASH_SET_MODE | Sets the flash mode (Example 7 285)<br>• IO_FLASH_MODE_OFF: Disables the digital output.<br>• 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.<br>• 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.<br>• IO_FLASH_MODE_CONSTANT_HIGH: Statically sets the digital output to high level (HIGH).<br>• IO_FLASH_MODE_CONSTANT_LOW: Statically sets the digital output to low level (LOW).<br>• 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.<br>• 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 4 284) |
| IS_IO_CMD_FLASH_APPLY_GLOBAL_PARAMS | Returns the parameters for the global exposure window and sets them as flash parameters (Examplel 4 284) |
| IS_IO_CMD_FLASH_GET_PARAMS | Returns the current values for flash delay and duration |

| hCam | Camera handle |
|---|---|
| | (Example 5 285) |
| IS_IO_CMD_FLASH_SET_PARAMS | Sets the current values for flash delay and duration (Example 5 285) |
| IS_IO_CMD_FLASH_GET_PARAMS_MIN | Returns the minimum possible values for flash delay and duration (Example 5 285) |
| IS_IO_CMD_FLASH_GET_PARAMS_MAX | Returns the maximum possible values for flash delay and duration (Example 5 285) |
| IS_IO_CMD_FLASH_GET_PARAMS_INC | Returns the increments for flash delay and duration (Example 5 285) |
| 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 13 287). |
| IS_IO_CMD_FLASH_SET_GPIO_PARAMS | Sets the flash delay and flash duration and allows the minimum values for GPIOs.<br>**Attention:** For values below 20 µs an unpredictable behavior can occur when flashing is done via the normal flash pin (Example 13 287). |
| **Pulse-width modulation** | |
| IS_IO_CMD_PWM_GET_SUPPORTED_GPIOS | Returns the GPIOs which can be used for pulse-width modulation (PWM) (Example 6 285) |
| IS_IO_CMD_PWM_GET_MODE | Returns the current PWM mode (Example 9 286) |
| IS_IO_CMD_PWM_SET_MODE | Sets the current PWM mode (Example 9 286)<br>• IS_FLASH_MODE_PWM: Sets the flash output as output for PWM mode.<br>• IO_GPIO_1: Sets GPIO 1 as output.<br>• IO_GPIO_2: Sets GPIO 2 as output. |
| IS_IO_CMD_PWM_GET_PARAMS | Returns the current values of the PWM parameters (Example 8 286) |
| IS_IO_CMD_PWM_SET_PARAMS | Sets the current values of the PWM parameters (Example 8 286) |
| IS_IO_CMD_PWM_GET_PARAMS_MIN | Returns the minimum possible values for PWM parameters (Example 8 286) |
| IS_IO_CMD_PWM_GET_PARAMS_MAX | Returns the maximum possible values for PWM parameters (Example 8 286) |
| IS_IO_CMD_PWM_GET_PARAMS_INC | Returns the increments of the PWM parameters (Example 8 286) |
| **LED** | |
| IS_IO_CMD_LED_GET_STATE | Returns the state of the LED (Example 3 284) |
| IS_IO_CMD_LED_SET_STATE | Sets the state of the LED (Example 3 284)<br>• IO_LED_STATE_1: Sets LED to orange.<br>• IO_LED_STATE_2: Sets LED to green. |
| IS_IO_CMD_LED_TOGGLE_STATE | Toggles between the LED states (Example 3 284) |
| 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 IO_FLASH_PARAMS structure**

| | | |
|---|---|---|
| INT | s32Delay | Flash delay (in µs) |
| UINT | u32Duration | Flash duration (in µs)<br>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)<br>1.0…10 000 Hz |
| double | dbl_DutyCycle | Duty cycle of the pulse-width modulation<br>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<br>• When setting the configuration: sets the configuration |
| UINT | u32State | • When reading the configuration: returns the current state of the GPIO (0 = Low, 1 = High).<br>• 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_api.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. |

### 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);
```

### 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));
```

### 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));
```

### 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 work with va
nRet = is_IO(m_hCam, IS_IO_CMD_FLASH_SET_GPIO_PARAMS, (void*)&flashParams,
             sizeof(flashParams));
```

## 4.3.65 is_IsVideoFinish

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 223 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:<br>• `IS_VIDEO_NOT_FINISH` = Digitizing of the image is not completed yet.<br>• `IS_VIDEO_FINISH` = Digitizing of the image is completed.<br>By setting `*pbo == IS_CAPTURE_STATUS` before calling the function, `pbo` contains the following digitizing status:<br>• `IS_VIDEO_NOT_FINISH` = Digitizing of the image is not completed yet.<br>• `IS_VIDEO_FINISH` = Digitizing of the image is completed.<br>• `IS_CAPTURE_STATUS` = Transfer error or conversion problem (e.g. destination memory is invalid)<br>The parameter `IS_CAPTURE_STATUS` replaces the previous parameter `IS_TRANSFER_FAILED.`<br>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()` 223
- `is_HasVideoStarted()` 259

## 4.3.66 is_LockSeqBuf

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 354 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()` 157.

### 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()` 354
- `is_AddToSequence()` 156
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

## 4.3.67 is_Measure

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

### 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 1 291) |
| `IS_MEASURE_CMD_SHARPNESS_AOI_INQUIRE` | Returns information of the AOI, e.g. the sharpness (Example 2 291) |
| `IS_MEASURE_CMD_SHARPNESS_AOI_SET_PRESET` | Sets different predefined AOIs in the image (Example 3 291) |
| `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:<br>• `s32X:` X position<br>• `s32Y:` Y position<br>• `s32Width:` AOI width<br>• `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

```
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));
```

## 4.3.68 is_ParameterSet

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 504 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 379

**Input parameter**

| | |
|---|---|
| `hCam` | Camera handle |
| `nCommand` | |
| `IS_PARAMETERSET_CMD_LOAD_EEPROM` | Loads a camera parameter set from the EEPROM (Example 1 293)<br>The parameter sets in the EEPROM of the camera can be loaded via special file names:<br>• `\\cam\\set1` oder `/cam/set1` |
| `IS_PARAMETERSET_CMD_LOAD_FILE` | Loads a camera parameter set from a file (Example 2 293)<br>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 3 293)<br>The parameter sets in the EEPROM of the camera can be saved with special file names:<br>• `\\cam\\set1` oder `/cam/set1` |
| `IS_PARAMETERSET_CMD_SAVE_FILE` | Saves a camera parameter set in a file (Example 4 293)<br>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 5 293). 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() 172

### 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));
```

## 4.3.69 is_PixelClock

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 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 379

**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 1 295). |
| `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 2 295) |
| `IS_PIXELCLOCK_CMD_GET_DEFAULT` | Returns the default pixel clock (Example 4 295) |
| `IS_PIXELCLOCK_CMD_GET` | Returns the current set pixel clock in MHz (Example 3 295) |
| `IS_PIXELCLOCK_CMD_SET` | Sets the pixel clock in MHz (Example 4 295) |
| `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() 240
- is_GetFrameTimeRange() 241
- is_Exposure() 216
- is_SetOptimalCameraTiming() 338
- is_SetFrameRate() 329
- is_SetAutoParameter() 303
- is_SetBinning() 310
- is_SetSubSampling() 347
- is_AOI() 159

### 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));
```

### 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));
}
```

## 4.3.70 is_ReadEEPROM

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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()` 358

### Example

```
char buffer[64];
is_ReadEEPROM( hCam, 0x00, buffer, 64 );
```

## 4.3.71 is_RenderBitmap

|  |  |
|---|---|
| USB 2.0<br>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()` 157 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()` 198 function (see also Color and memory formats 502).

**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: | |
| `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_api.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() 157
- is_SetColorMode() 319
- is_SetDisplayMode() 322
- is_DirectRenderer() 198

### 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++)

## 4.3.72 is_ResetToDefault

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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_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() 292

## 4.3.73 is_SetAllocatedImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 156 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()` 157. 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()` 222 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 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() 157
- is_FreeImageMem() 222
- is_AddToSequence() 156
- is_SetImageMem() 337
- is_GetColorDepth() 237
- is_GetImgMemPitch() 248

### 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);
```

## 4.3.74 is_SetAutoParameter

|  |  |
|---|---|
| USB 2.0<br>USB 3.0<br>GigE | USB 2.0<br>USB 3.0<br>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 45 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<br>⊟ Control parameters<br>`pval1` = 1 enables control, 0 disables control |
| `IS_GET_ENABLE_AUTO_GAIN` | Returns the current auto gain setting or white level adjustment<br>⊟ Control parameters<br>`pval1:` returns the current setting |
| `IS_SET_ENABLE_AUTO_SENSOR_GAIN` | Enables/disables the internal auto gain control function of the sensor*1<br>⊟ Control parameters<br>`pval1` = 1 enables control, 0 disables control |
| `IS_GET_ENABLE_AUTO_SENSOR_GAIN` | Returns the current auto gain setting of the sensor*1<br>⊟ Control parameters<br>`pval1:` returns the current setting |
| `IS_SET_ENABLE_AUTO_SHUTTER` | Enables/disables the auto exposure shutter function.<br>⊟ Control parameters<br>`pval1` = 1 enables control, 0 disables control |

| hCam | Camera handle |
|---|---|
| IS_GET_ENABLE_AUTO_SHUTTER | Returns the current auto exposure shutter setting.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_ENABLE_AUTO_SENSOR_SHUTTER | Enables/disables the sensor's internal auto exposure shutter function*1<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_ENABLE_AUTO_SENSOR_SHUTTER | Returns the sensor's current auto exposure shutter setting*1<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_ENABLE_AUTO_WHITEBALANCE | Enables/disables the auto white balance function.<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_ENABLE_AUTO_WHITEBALANCE | Returns the current auto white balance setting.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_ENABLE_AUTO_SENSOR_WHITEBALANCE | Enables/disables the sensor's internal auto white balance function*1<br>Control parameters<br>pval1: white balance mode (see is_GetAutoInfo() 227):<br>• WB_MODE_DISABLE<br>• WB_MODE_AUTO<br>• WB_MODE_ALL_PULLIN<br>• WB_MODE_INCANDESCENT_LAMP<br>• WB_MODE_FLUORESCENT_DL<br>• WB_MODE_OUTDOOR_CLEAR_SKY<br>• WB_MODE_OUTDOOR_CLOUDY |
| IS_GET_ENABLE_AUTO_SENSOR_WHITEBALANCE | Returns the sensor's current auto white balance setting*1<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_ENABLE_AUTO_FRAMERATE | Enables/disables the auto frame rate function.<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_ENABLE_AUTO_FRAMERATE | Returns the current auto frame rate setting.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_ENABLE_AUTO_SENSOR_FRAMERATE | Enables/disables the sensor's internal auto frame rate function*1<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_ENABLE_AUTO_SENSOR_FRAMERATE | Returns the sensor's current auto frame rate setting*1<br>Control parameters<br>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.<br>Control parameters<br>pval1: defines the setpoint (average brightness of the |

| hCam | Camera handle |
|---|---|
| | image); the following rule applies independently of the image bit depth:<br>• 0 = black<br>• 128 = 50% gray (default)<br>• 255 = white<br>Note on the sensor's internal control functionality<br>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.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_AUTO_GAIN_MAX | Sets the upper limit for auto gain control.<br>Control parameters<br>pval1: valid value for gain (0...100) |
| IS_GET_AUTO_GAIN_MAX | Returns the upper limit for auto gain control.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_AUTO_SHUTTER_MAX | Sets the upper limit for auto exposure shutter.<br>Control parameters<br>pval1: valid exposure value (0 sets the value continuously to max. exposure) |
| IS_GET_AUTO_SHUTTER_MAX | Returns the upper limit for auto exposure shutter.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_AUTO_BRIGHTNESS_ONCE | Enables/disables the automatic disable for automatic brightness control (gain and exposure time)[*2]<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_AUTO_BRIGHTNESS_ONCE | Returns the automatic disable status[*2]<br>Control parameters<br>pval1: returns the current setting |
| **Speed and hysteresis** | |
| IS_SET_AUTO_SPEED | Sets the speed value for the auto function.<br>Control parameters<br>pval1: defines the control speed (0...100) |
| IS_GET_AUTO_SPEED | Returns the speed value for the auto function.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_AUTO_SKIPFRAMES | Sets the number of frames to be skipped during automatic control.<br>Control parameters<br>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.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_AUTO_SKIPFRAMES_RANGE | Returns the permissible range for the number of frames to be skipped. |

| hCam | Camera handle |
|---|---|
| | Control parameters<br>• pval1: returns the minimum permitted value<br>• pval2: returns the maximum permitted value |
| IS_SET_AUTO_HYSTERESIS | Sets the hysteresis value for auto exposure shutter and auto gain control.<br>Control parameters<br>pval1: defines the hysteresis value (default: 2) |
| IS_GET_AUTO_HYSTERESIS | Returns the hysteresis value for auto exposure shutter and auto gain control.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_AUTO_HYSTERESIS_RANGE | Returns the permissible range for the hysteresis value.<br>Control parameters<br>• pval1: returns the minimum permitted value<br>• 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.<br>Control parameters<br>pval1: defines which fields of view are used for auto exposure shutter (see is_GetAutoInfo() 227):<br>• AS_PM_NONE<br>• AS_PM_SENS_CENTER_WEIGHTED<br>• AS_PM_SENS_CENTER_SPOT<br>• AS_PM_SENS_PORTRAIT<br>• AS_PM_SENS_LANDSCAPE |
| IS_GET_SENS_AUTO_SHUTTER_PHOTOM | Returns the photometry mode for auto exposure shutter.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_SENS_AUTO_SHUTTER_PHOTOM_DEF | Returns the default photometry mode for auto exposure shutter.<br>Control parameters<br>pval1: returns the default |
| IS_SET_SENS_AUTO_GAIN_PHOTOM | Sets the photometry mode for auto gain control.<br>Control parameters<br>pval1: defines which fields of view are used for auto gain control (see is_GetAutoInfo() 227):<br>• AG_PM_NONE<br>• AG_PM_SENS_CENTER_WEIGHTED<br>• AG_PM_SENS_CENTER_SPOT<br>• AG_PM_SENS_PORTRAIT<br>• AG_PM_SENS_LANDSCAPE |
| IS_GET_SENS_AUTO_GAIN_PHOTOM | Returns the photometry mode for auto gain control.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_SENS_AUTO_GAIN_PHOTOM_DEF | Returns the default photometry mode for auto gain control.<br>Control parameters<br>pval1: returns the default |
| Adjusting auto white balance | |
| IS_SET_AUTO_WB_OFFSET | Sets the offset values for the red and blue channels. |

| hCam | Camera handle |
|---|---|
| | Control parameters<br>• pval1: defines the red level offset (-50...50)<br>• pval2: defines the blue level offset (-50...50) |
| IS_GET_AUTO_WB_OFFSET | Returns the offset values for the red and blue channels.<br>Control parameters<br>• pval1: returns the red level offset (-50...50)<br>• pval2: returns the blue level offset (-50...50) |
| IS_SET_AUTO_WB_GAIN_RANGE | Sets the color gain limits for auto white balance.<br>Control parameters<br>• pval1: sets the lowest gain value<br>• pval2: sets the highest gain value |
| IS_GET_AUTO_WB_GAIN_RANGE | Returns the color gain limits for auto white balance.<br>Control parameters<br>• pval1: returns the minimum permitted gain value<br>• pval2: returns the maximum permitted gain value |
| IS_SET_AUTO_WB_ONCE | Sets automatic disabling of auto white balance*2<br>Control parameters<br>pval1 = 1 enables control, 0 disables control |
| IS_GET_AUTO_WB_ONCE | Returns the automatic disable status*2<br>Control parameters<br>pval1: returns the current setting |
| **Speed and hysteresis** | |
| IS_SET_AUTO_WB_SPEED | Sets the speed for auto white balance.<br>Control parameters<br>pval1: defines the control speed (0...100) |
| IS_GET_AUTO_WB_SPEED | Returns the speed for auto white balance.<br>Control parameters<br>pval1: returns the current setting |
| IS_SET_AUTO_WB_HYSTERESIS | Sets the hysteresis value for auto white balance.<br>Control parameters<br>pval1: defines the hysteresis value (default: 2) |
| IS_GET_AUTO_WB_HYSTERESIS | Returns the hysteresis value for auto white balance.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_AUTO_WB_HYSTERESIS_RANGE | Returns the permissible range for the hysteresis value.<br>Control parameters<br>• pval1: returns the minimum permitted value<br>• pval2: returns the maximum permitted value |
| IS_SET_AUTO_WB_SKIPFRAMES | Sets the number of frames to be skipped during automatic control.<br>Control parameters<br>pval1: defines the number (default: 4) |
| IS_GET_AUTO_WB_SKIPFRAMES | Returns the number of frames to be skipped during automatic control.<br>Control parameters<br>pval1: returns the current setting |
| IS_GET_AUTO_WB_SKIPFRAMES_RANGE | Returns the permissible range for the number of frames to be skipped.<br>Control parameters |

| | |
|---|---|
| hCam | Camera handle |
| | • pval1: returns the minimum permitted value<br>• 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_api.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() 292
- is_GetAutoInfo() 227
- is_SetHardwareGain() 333
- is_SetHWGainFactor() 335
- is_Exposure() 216
- is_SetFrameRate() 329
- is_AOI() 159

### 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);
```

## 4.3.75 is_SetBinning

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 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()` 329
- `is_Exposure()` 216
- If you are using the DCx Camera's flash function: `is_IO()` 280

**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 63.

**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. |
| 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. |

| hCam | Camera handle |
|---|---|
| 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 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_api.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() 347
- is_AOI() 159
- is_SetImagePos() 435
- is_PixelClock() 294

## 4.3.76 is_SetCameraID

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 273 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.<br>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_api.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_InitCamera()` 273

- is_GetCameraInfo() 231
- is_CameraStatus() 172

## 4.3.77 is_SetColorConverter

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 26 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()` 353 or set the camera to trigger mode (see `is_SetExternalTrigger()` 327).

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `ColorMode` | Color mode for which the converter is to be set.<br>For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats 502 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 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_api.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() 236
- is_SetColorMode() 319
- is_Convert() 188

## 4.3.78 is_SetColorCorrection

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 303).

**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:<br>• `IS_CCOR_ENABLE_NORMAL`<br>• `IS_CCOR_ENABLE_BG40_ENHANCED`<br>• `IS_CCOR_ENABLE_HQ_ENHANCED`<br>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:<br>• `IS_CCOR_ENABLE_NORMAL`<br>• `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_api.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() 315
- is_SetColorMode() 319
- is_SetAutoParameter() 303

## 4.3.79 is_SetColorMode

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 322 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 502 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 |
| `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 |

| hCam | Camera handle |
|---|---|
| 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_api.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() 322
- is_SetColorConverter() 315
- is_SetColorCorrection() 317
- is_GetColorDepth() 237
- is_AllocImageMem() 157
- is_RenderBitmap() 297

**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 |
| 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 |

## 4.3.80 is_SetDisplayMode

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 133 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()` 273
- `is_SetDisplayMode()` 322
- `is_ExitCamera()` 213

See also Programming: Thread programming 453

**Input parameters**

| hCam | Camera handle |
|---|---|
| Mode | |
| IS_SET_DM_DIB | Captures an image in system memory (RAM). Using `is_RenderBitmap()` 297, 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. |

**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 379). 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 \| | Image display in DirectDraw Overlay Surface mode |

| | |
|---|---|
| IS_SET_DM_ALLOW_OVERLAY | |
| 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_api.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() 297
- is_SetColorMode() 319
- is_DirectRenderer() 198

### 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
```

## 4.3.81 is_SetDisplayPos

|  |  |
|---|---|
| USB 2.0<br>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()` 297. 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()` 159 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()` 159
- `is_RenderBitmap()` 297
- `is_SetDisplayMode()` 322

## 4.3.82 is_SetErrorReport

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 239 function.

**Note**

`is_SetErrorReport()` can be called before calling `is_InitCamera()` 273.
You only need to enable the `is_SetErrorReport()` function once for all cameras in the application.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle<br>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<br>`IS_GET_ERR_REP_MODE` | Current setting |
| `IS_SUCCESS` | Function executed successfully |

### Related functions

- `is_GetError()` 239
- `is_CaptureStatus()` 174
- `is_CameraStatus()` 172

## 4.3.83 is_SetExternalTrigger

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 223 is called, or a continuous triggered capture is started when `is_CaptureVideo()` 177 is called. In hardware trigger mode, you can use the `is_ForceTrigger()` 221 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 134.

**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()` 223 (single frame mode)<br>Call of `is_CaptureVideo()` 177 (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 | |

**Return values**

| | |
|---|---|
| When used with<br>`IS_GET_EXTERNALTRIGGER` | Returns the current setting |
| When used with<br>`IS_GET_TRIGGER_STATUS` | Returns the current signal level at the trigger input |
| When used with<br>`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() 177
- is_FreezeVideo() 223
- is_ForceTrigger() 221
- is_SetTriggerCounter() 351
- is_SetTriggerDelay() 352
- is_IO() 280

### 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++)

## 4.3.84 is_SetFrameRate

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 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()` 216 is recommended in order to keep the defined camera settings.

**Attention**

The use of the following functions will affect the frame rate:

- `is_PixelClock()` 294
- `is_SetOptimalCameraTiming()` 338
- `is_AOI()` 159 (if the image size is changed)
- `is_SetSubSampling()` 347
- `is_SetBinning()` 310

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()` 240. |
| IS_GET_DEFAULT_FRAMERATE | Returns the default frame rate. |
| newFPS | Returns the frame rate actually set. |

**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() 240
- is_GetFrameTimeRange() 241
- is_PixelClock() 294
- is_SetOptimalCameraTiming() 338
- is_Exposure() 216
- is_SetAutoParameter() 303
- is_AOI() 159
- is_SetSubSampling() 347
- is_SetBinning() 310
- is_CaptureVideo() 177

## 4.3.85 is_SetGainBoost

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 333
- `is_SetHWGainFactor()` 335
- `is_SetAutoParameter()` 303

## 4.3.86 is_SetGamma

| | |
|---|---|
| USB 2.0<br>USB 3.0<br>GigE | USB 2.0<br>USB 3.0<br>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<br>(Range: 1…1000.<br>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_api.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 39
- Basics: Color filter (Bayer filter) 26
- Programming: `is_SetColorConverter()` 315

## 4.3.87 is_SetHardwareGain

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 chapter.

You can use the `is_GetSensorInfo()` 251 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()` 331.

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()` 317).

**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. |
| `IS_SET_ENABLE_AUTO_GAIN` | Enables the auto gain functionality (see also `is_SetAutoParameter()` 303). 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. |

| | |
|---|---|
| hCam | Camera handle |
| 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<br>IS_GET_MASTER_GAIN<br>IS_GET_RED_GAIN<br>IS_GET_GREEN_GAIN<br>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_api.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() 335
- is_GetSensorInfo() 251
- is_SetGainBoost() 331
- is_SetAutoParameter() 303

## 4.3.88 is_SetHWGainFactor

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 333 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 460 chapter.

You can use the `is_GetSensorInfo()` 251 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()` 333 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 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<br>`IS_GET_MASTER_GAIN_FACTOR`<br>`IS_GET_RED_GAIN_FACTOR`<br>`IS_GET_GREEN_GAIN_FACTOR` | Current setting |

| | |
|---|---|
| IS_GET_BLUE_GAIN_FACTOR | |
| When used with<br>IS_SET_MASTER_GAIN_FACTOR<br>IS_SET_RED_GAIN_FACTOR<br>IS_SET_GREEN_GAIN_FACTOR<br>IS_SET_BLUE_GAIN_FACTOR | Defined setting |
| When used with<br>IS_GET_DEFAULT_MASTER_GAIN_FACTOR<br>IS_GET_DEFAULT_RED_GAIN_FACTOR<br>IS_GET_DEFAULT_GREEN_GAIN_FACTOR<br>IS_GET_DEFAULT_BLUE_GAIN_FACTOR | Default setting |
| When used with<br>IS_INQUIRE_MASTER_GAIN_FACTOR<br>IS_INQUIRE_RED_GAIN_FACTOR<br>IS_INQUIRE_GREEN_GAIN_FACTOR<br>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() 333
- is_SetGainBoost() 331
- is_SetAutoParameter() 303
- is_GetSensorInfo() 251

**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
```

## 4.3.89 is_SetImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 223, 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()` 157, 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()` 438 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()` 157
- `is_FreeImageMem()` 222
- `is_AddToSequence()` 156
- `is_SetAllocatedImageMem()` 301
- `is_GetColorDepth()` 237
- `is_GetImageMem()` 247
- `is_GetImageMemPitch()` 248

## 4.3.90 is_SetOptimalCameraTiming

| | |
|---|---|
| USB 2.0<br>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()` 177). 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()` 329
- `is_Exposure()` 216
- If you are using the DCx Camera's flash function: `is_IO()` 280

### 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`<br>`[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<br>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() 294
- is_SetFrameRate() 329
- is_SetAutoParameter() 303
- is_CaptureVideo() 177

## 4.3.91 is_SetRopEffect

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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.<br>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.<br>0 = Turn off<br>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_api.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() 310
- is_SetSubSampling() 347
- is_AOI() 159

## 4.3.92 is_SetSaturation

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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.<br>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.<br>Range: `[IS_MIN_SATURATION … IS_MAX_SATURATION]` |
| `IS_GET_SATURATION_V` | Returns the current value for the V saturation. |

**Return values**

| | |
|---|---|
| When used with<br>`IS_GET_SATURATION_U`<br>`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()` 319
- `is_SetColorCorrection()` 317
- `is_SetColorConverter()` 315

## 4.3.93 is_SetSensorScaler

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 461 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_api.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_GetSensorScalerInfo()` 252

### 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);
```

## 4.3.94 is_SetSensorTestImage

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 253 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()` 253. |
| `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_api.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() 253
- is_GetTestImageValueRange() 255

## 4.3.95 is_SetSubSampling

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 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()` 329
- `is_Exposure()` 216
- If you are using the DCx Camera's flash function: `is_IO()` 280

**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. |
| 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 |

| hCam | Camera handle |
|---|---|
| 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<br>IS_GET_SUBSAMPLING | Current setting: Returns an ORing of the defined constants from ueye.h, e.g. IS_SUBSAMPLING_2X_HORIZONTAL |
| When used with<br>IS_GET_SUBSAMPLING_FACTOR_VERTICAL<br>IS_GET_SUBSAMPLING_FACTOR_HORIZONTAL | Current setting: Returns the current factor as integer value (2, 3, 4, 5, 6, 8, 16) |
| When used with<br>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<br>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. |
| 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_api.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() 310
- is_AOI() 159
- is_ImageFormat() 267
- is_PixelClock() 294

## 4.3.96 is_SetTimeout

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 135 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.<br>Value range [0; 4...429496729] (corresponds to 40 ms to approx. 1193 hours)<br>0 = use default value of the uc480 API<br>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_api.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()` 256
- `is_CaptureVideo()` 177
- `is_FreezeVideo()` 223
- `is_SetExternalTrigger()` 327

### Example

```
// Set user-defined timeout to 120 seconds
is_SetTimeout(hCam, IS_TRIGGER_TIMEOUT, 12000);
```

## 4.3.97 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_api.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()` 327
- `is_CameraStatus()` 172

## 4.3.98 is_SetTriggerDelay

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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 460 chapter.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `nTriggerDelay` | Time by which the image capture is delayed (in µs)<br>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_api.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()` 280
- `is_SetExternalTrigger()` 327

## 4.3.99 is_StopLiveVideo

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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() 223
- is_CaptureVideo() 177
- is_SetDisplayMode() 322

## 4.3.100 is_UnlockSeqBuf

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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.<br>When you pass `IS_IGNORE_PARAMETER`, the image memory is only identified by its starting address.<br>`nNum` identifies the position in the sequence list, not the memory ID assigned with `is_AllocImageMem()` 157. |
| `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()` 289
- `is_WaitForNextImage()` 356

## 4.3.101 is_WaitEvent

|  |  |
|---|---|
| - | USB 2.0<br>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()` 209) |
| `nTimeout` | Time (in ms) that the function will wait for an event to occur.<br>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.<br>That means the specified timeout expired without the event having occurred. |

### Related functions

- `is_EnableEvent()` 209
- `is_DisableEvent()` 205

### 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);
```

## 4.3.102 is_WaitForNextImage

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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()` 278). 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()` 354 in order to be re-used in the sequence.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `timeout` | Timeout in ms. Range 0…$2^{32}$-1<br>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_api.dll` (API) and the driver file (`uc480_usb.sys`) do not match. |
| `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.<br>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()` 278
- `is_ExitImageQueue()` 215
- `is_UnlockSeqBuf()` 354

**See also:**

- How to proceed: Image memory sequences 132
- How to proceed: Allocating image memory 130
- Transfer error: uc480 Viewer Image infos 90
- Transfer error: `is_GetImageInfo()` 244

## 4.3.103 is_WriteEEPROM

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>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_api.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_ReadEEPROM()` 296

## 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 149 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<br>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()` 369 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()` 362. |
| `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()` 369

## 4.4.2 isavi_CloseAVI

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | - |

**Syntax**

```
INT isavi_CloseAVI (INT nAviID)
```

**Description**

`isavi_CloseAVI()` closes an AVI file which was opened using `isavi_OpenAVI()` 371.

**Input parameters**

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |

**Related functions**

- `isavi_OpenAVI()` 371
- `isavi_InitAVI()` 369
- `isavi_ExitAVI()` 362

## 4.4.3 isavi_DisableEvent

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |
| `IS_AVI_ERR_PARAMETER` | One of the submitted parameters is outside the valid range. |

**Related functions**

- isavi_EnableEvent() 361

## 4.4.4 isavi_EnableEvent

|  |  |
|---|---|
| USB 2.0<br>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() 369 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() 362. |
| IS_AVI_ERR_PARAMETER | One of the submitted parameters is outside the valid range. |

**Related functions**

- isavi_DisableEvent() 360

## 4.4.5 isavi_ExitAVI

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |
| `IS_AVI_ERR_INVALID_FILE` | The file has no valid AVI format. |

### Related functions

- `isavi_InitAVI()` 369
- `isavi_OpenAVI()` 371
- `isavi_CloseAVI()` 360

## 4.4.6 isavi_ExitEvent

|  |  |
|---|---|
| USB 2.0<br>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()` 370 before it can be enabled and used.

### Input parameters

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |
| `IS_AVI_ERR_PARAMETER` | One of the submitted parameters is outside the valid range. |

### Related functions

- `isavi_InitEvent()` 370
- `isavi_EnableEvent()` 361
- `isavi_DisableEvent()` 360

## 4.4.7 isavi_GetAVIFileName

|  |  |
|---|---|
| USB 2.0<br>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()` 371 function and a Null parameter was specified.

**Note**

The functions `isavi_OpenAVI()` 371 and `isavi_GetAVIFileName()` do not support UNICODE strings. For this purpose you use the functions `isavi_OpenAVIW()` 372 and `isavi_GetAVIFileNameW()` 365.

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()` 369 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()` 362. |

### Related functions

- `isavi_GetAVISize()` 366
- `isavi_OpenAVI()` 371

## 4.4.8 isavi_GetAVIFileNameW

|  |  |
|---|---|
| USB 2.0<br>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()` 372 function and a `NULL` parameter was specified.

### Input parameters

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |

### Related functions

- `isavi_GetAVISize()` 366
- `isavi_OpenAVIW()` 372

## 4.4.9 isavi_GetAVISize

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |

### Related functions

- `isavi_GetAVIFileName()` 364

## 4.4.10 isavi_GetnCompressedFrames

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |

### Related functions

- `isavi_GetnLostFrames()` 368
- `isavi_ResetFrameCounters()` 373

## 4.4.11 isavi_GetnLostFrames

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |

### Related functions

- `isavi_GetnCompressedFrames()` 367
- `isavi_ResetFrameCounters()` 373

## 4.4.12 isavi_InitAVI

|  |  |
|---|---|
| USB 2.0<br>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()` 362
- `isavi_OpenAVI()` 371
- `isavi_CloseAVI()` 360

## 4.4.13 isavi_InitEvent

|  |  |
|---|---|
| USB 2.0<br>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()` 361.

### Input parameters

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |
| `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()` 363
- `isavi_EnableEvent()` 361
- `isavi_DisableEvent()` 360

### 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<br>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()` 371 and `isavi_GetAVIFileName()` do not support UNICODE strings. For this purpose you use the functions `isavi_OpenAVIW()` 372 and `isavi_GetAVIFileNameW()` 365.

**Input parameters**

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |
| `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()` 364
- `isavi_CloseAVI()` 360
- `isavi_InitAVI()` 369
- `isavi_ExitAVI()` 362

## 4.4.15 isavi_OpenAVIW

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |
| `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()` 365
- `isavi_CloseAVI()` 360
- `isavi_InitAVI()` 369
- `isavi_ExitAVI()` 362

## 4.4.16 isavi_ResetFrameCounters

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |

### Related functions

- `isavi_GetnCompressedFrames()` 367
- `isavi_GetnLostFrames()` 368

## 4.4.17 isavi_SetFrameRate

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |
| `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()` 375
- `isavi_SetImageSize()` 376

## 4.4.18 isavi_SetImageQuality

|  |  |
|---|---|
| USB 2.0<br>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()` 369 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()` 362. |
| `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()` 374
- `isavi_SetImageSize()` 376

## 4.4.19 isavi_SetImageSize

|  |  |
|---|---|
| USB 2.0<br>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 502 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()` 369 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.<br>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()` 362. |
| `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()` 374
- `isavi_SetImageQuality()` 375

### 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<br>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()` 369 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()` 362. |
| `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()` 378
- `isavi_InitEvent()` 370
- `isavi_ExitAVI()` 362

## 4.4.21 isavi_StopAVI

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | - |

### Syntax

```
INT isavi_StopAVI (INT nAviID)
```

### Description

`isavi_StopAVI()` stops the image capture thread. Subsequent calls of `isavi_AddFrame()` 359 will be ignored.

### Input parameters

| | |
|---|---|
| `nAviID` | Instance ID set by the `isavi_InitAVI()` 369 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()` 362. |
| `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()` 377
- `isavi_InitEvent()` 370
- `isavi_ExitAVI()` 362

## 4.5 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**

**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 516.

| Obsolete function | Recommended alternative | No longer documented since driver version |
|---|---|---|
| is_ConvertImage() 382 | is_Convert() 188 | |
| is_DisableDDOverlay() 383 | is_DirectRenderer() 198 | |
| is_EnableDDOverlay() 384 | | |
| is_GetCameraType() 385 | is_GetCameraInfo() 231 | |
| is_GetCaptureErrorInfo() 386 | is_CaptureStatus() 174 | |
| is_GetDC() 388 | is_DirectRenderer() 198 | |
| is_GetDDOvlSurface() 389 | | |
| is_GetExposureRange 390 | is_Exposure() 216 | |
| is_GetGlobalFlashDelays() 391 | is_IO() 280 | |
| is_GetLastMemorySequence() | The uc480 memory board is not supported any longer. | 3.30 |
| is_GetMemorySequenceWindow() | | |
| is_GetNumberOfMemoryImages() | | |
| is_GetPixelClockRange() 392 | is_PixelClock() 294 | |
| is_GetRevisionInfo() | is_GetCameraInfo() 231 | 3.20 |
| is_GetHWGain() | is_SetAutoParameter() 303 | 3.31 |
| is_HideDDOverlay() 393 | is_DirectRenderer() 198 | |
| is_IsMemoryBoardConnected() | The uc480 memory board is not supported any longer. | 3.30 |
| is_LoadBadPixelCorrectionTable() 394 | is_HotPixel() 260 | |
| is_LoadImage() 395 | is_ImageFile() 264 | |
| is_LoadImageMem() 396 | | |
| is_LoadParameters() 398 | is_ParameterSet() 292 | |
| is_LockDDMem() 400 | is_DirectRenderer() 198 | |
| is_LockDDOverlayMem() 401 | | |
| is_MemoryFreezeVideo() | The uc480 memory board is not supported any longer. | 3.30 |
| is_ReleaseDC() 403 | is_DirectRenderer() 198 | |
| is_ResetCaptureErrorInfo() 404 | is_CaptureStatus() 174 | |
| is_ResetMemory() | The uc480 memory board is not supported any longer. | 3.30 |
| is_SaveBadPixelCorrectionTable() 405 | is_HotPixel() 260 | |

| Obsolete function | Recommended alternative | No longer documented since driver version |
|---|---|---|
| is_SaveImage() 406 | is_ImageFile() 264 | |
| is_SaveImageEx() 407 | | |
| is_SaveImageMem() 409 | | |
| is_SaveImageMemEx() 410 | | |
| is_SaveParameters() 412 | is_ParameterSet() 292 | |
| is_SetAOI() 414 | is_AOI() 159 | |
| is_SetBadPixelCorrection() 417 | is_HotPixel() 260 | |
| is_SetBadPixelCorrectionTable() 418 | | |
| is_SetBayerConversion() 420 | is_SetColorConverter() 315 | |
| is_SetBlCompensation() 421 | is_Blacklevel() 170 | |
| is_SetBrightness() | is_SetGamma() 332<br>is_Blacklevel() 170 | 3.40 |
| is_SetContrast() | is_SetExposureTime() 427<br>is_SetHardwareGain() 333 | 3.40 |
| is_SetConvertParam() 423 | is_Convert() 188 | |
| is_SetDDUpdateTime() 425 | is_DirectRenderer() 198 | |
| is_SetEdgeEnhancement() 426 | is_EdgeEnhancement() 206 | |
| is_SetExposureTime() 427 | is_Exposure() 216 | |
| is_SetFlashDelay() 429 | is_IO() 280 | |
| is_SetFlashStrobe() 431 | | |
| is_SetHwnd() 433 | is_DirectRenderer() 198 | |
| is_SetImageAOI() 434 | is_AOI() 159 | |
| is_SetImageSize() 438 | | |
| is_SetImagePos() 435 | | |
| is_SetIO() 440 | is_IO() 280 | |
| is_SetIOMask() 441 | | |
| is_SetKeyColor() 442 | is_DirectRenderer() 198 | |
| is_SetLED() 443 | is_IO() 280 | |
| is_SetMemoryMode() | The uc480 memory board is not supported any longer. | 3.30 |
| is_SetPixelClock() 444 | is_PixelClock() 294 | |
| is_SetWhiteBalance() | is_SetAutoParameter() 303 | 3.31 |
| is_SetWhiteBalanceMultipliers() | | |
| is_ShowDDOverlay() 446 | is_DirectRenderer() 198 | |
| is_StealVideo() 447 | | |
| is_TransferImage() | The uc480 memory board is not supported any longer. | 3.30 |
| is_TransferMemorySequence() | | |
| is_UnlockDDMem() 448 | is_DirectRenderer() 198 | |
| is_UnlockDDOverlayMem() 449 | | |
| is_UpdateDisplay() 450 | | |

**Attention**

The `is_SetWhiteBalance()` and `is_SetWhiteBalanceMultipliers()` functions have been completely replaced by the `is_SetAutoParameter()` 303 function and are no longer supported by the uc480 API.

## 4.5.1 is_ConvertImage

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_ConvertImage(HIDS hCam,
                    char* pcSource, INT nIDSource,
                    char** ppcDest, INT* nIDDest,
                    INT* reserved)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_Convert()` 188 function instead (see also Obsolete functions 379).

`is_ConvertImage()` converts a raw Bayer image to the desired format. This conversion is done in the PC. You can use `is_SetConvertParam()` 423 to define the conversion settings.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pcSource` | Pointer to the input image |
| `nIDSource` | Memory ID of the input image |
| `ppcDest` | Pointer to the output image<br>In case a `NULL` value is passed, a new memory is allocated internally. |
| `nIDDest` | Memory ID of the output image |
| `reserved` | Reserved. `NULL` must be passed here. |

### 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_INVALID_BUFFER_SIZE` | The image memory has an inappropriate size to store the image in the desired format. |
| `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_api.dll` (API) and the driver file (`uc480_usb.sys`) do not match. |
| `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_SetConvertParam() 423
- is_SetColorMode() 319
- is_SetBayerConversion() 420

## 4.5.2 is_DisableDDOverlay

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_DisableDDOverlay (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the is_DirectRenderer() 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, is_DisableDDOverlay() disables overlay mode and releases the memory allocated to the overlay. This results in discarding of the overlay data.

### Input parameters

| | |
|---|---|
| hCam | Camera handle |

### Return values

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |

### Related functions

- is_EnableDDOverlay() 384
- is_HideDDOverlay() 393
- is_SetDisplayMode() 322
- is_ShowDDOverlay() 446
- is_GetDDOvlSurface() 389

## 4.5.3 is_EnableDDOverlay

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_EnableDDOverlay (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_EnableDDOverlay()` enables live overlay mode. The overlay cannot be displayed directly, but needs to be visualized before by calling `is_ShowDDOverlay()` 446. The overlay uses black as the so-called key color, so that overlay graphics may not contain any black color.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetDisplayMode()` 322
- `is_DisableDDOverlay()` 383
- `is_ShowDDOverlay()` 446
- `is_HideDDOverlay()` 393
- `is_GetDDOvlSurface()` 389

## 4.5.4 is_GetCameraType

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_GetCameraType (HIDS hCam)
```

**Description**

`is_GetCameraType()` returns the camera type.

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_GetCameraInfo()` 231 function instead (see also Obsolete functions 379).

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |

**Return values**

| | |
|---|---|
| `IS_CAMERA_TYPE_UEYE_USB_SE` | DCU223x, DCU224x and DCC1240xUSB camera |
| `IS_CAMERA_TYPE_UEYE_USB_LE` | DCC1545M / DCC1645CUSB camera |
| `IS_CAMERA_TYPE_UEYE_USB3_CP` | DCC3240x camera |

## 4.5.5 is_GetCaptureErrorInfo

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_GetCaptureErrorInfo (HIDS hCam,
                            UEYE_CAPTURE_ERROR_INFO* CaptureErrorInfo,
                            UINT SizeCaptureErrorInfo)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_CaptureStatus()` 174 function instead (see also Obsolete functions 379).

`is_GetCaptureErrorInfo()` returns detail information on errors that occurred during an image capture process. The function lists all errors that occurred since the last call of the `is_ResetCaptureErrorInfo()` 404 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `CaptureErrorInfo` | Structure of the `UC480_CAPTURE_ERROR_INFO` type that is filled by the uc480 driver. This structure then contains the error list. |
| `SizeCaptureErrorInfo` | Indicates the size of the `CaptureErrorInfo` structure. |

### Contents of the UC480_CAPTURE_ERROR_INFO structure

| | | |
|---|---|---|
| `DWORD` | `dwCapErrCnt_Total` | Returns the total number of errors occurred since the last reset. |
| `BYTE` | `reserved[60]` | Reserved for an internal function |
| `DWORD` | `adwCapErrCnt_Detail[CapErr]` | This array returns the current count for each possible error. The possible errors are listed below. To query the counter of a specific error type, pass its definition in the `CapErr` parameter. |

### Possible CapErr Error Types

| Definition | Description | # |
|---|---|---|
| `IS_CAPERR_API_NO_DEST_MEM` | There is no destination memory for copying the finished image. | 1 |
| `IS_CAPERR_API_CONVERSION_FAILED` | The current image could not be processed correctly. | 2 |
| `IS_CAPERR_API_IMAGE_LOCKED` | The destination buffers are locked and could not be written to. | 3 |
| `IS_CAPERR_DRV_OUT_OF_BUFFERS` | No free internal image memory is available to the driver. The image was discarded. | 4 |
| `IS_CAPERR_DRV_DEVICE_NOT_READY` | The camera is no longer available. It is not possible to access images that have already been transferred. | 5 |
| `IS_CAPERR_USB_TRANSFER_FAILED` | The image was not transferred over the USB bus. | 6 |
| `IS_CAPERR_DEV_TIMEOUT` | The maximum allowable time for image capturing in the camera was exceeded. | 7 |

| # | Possible cause | Remedy |
|---|---|---|
| 1 | Not enough destination memory allocated or all destination buffers locked by the application | • Release locked destination memory<br>• Allocate more destination memory<br>• Reduce the frame rate so that there is more time to process the filled destination memory |
| 2 | Internal error during internal processing of the image | - |
| 3 | All destination buffers locked by the application | • Release locked destination memory<br>• Allocate more destination memory<br>• Reduce the frame rate so that there is more time to process the filled destination memory |
| 4 | 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<br>• Disable resource-intensive API image pre-processing functions (e.g. edge enhancement, color correction, choose smaller filter mask for software color conversion) |
| 5 | The camera has been disconnected or closed | - |
| 6 | Not enough free bandwidth on the USB bus for transferring the image | • Reduce the pixel clock frequency<br>• Operate fewer cameras simultaneously on a USB bus<br>• Check the quality of the USB cabling and components |
| 7 | The selected timeout value is too low for image capture | • Reduce the exposure time<br>• Increase the timeout |
| 8 | The selected data rate of the sensor is too high | • Reduce the pixel clock frequency<br>• Reduce the frame rate<br>• Reduce the image size |
| 9 | The camera's frame rate is too high or the bandwidth on the network is insufficient to transfer the image | • Reduce the frame rate<br>• Increase the value for the receive descriptors in the network card settings |

**Return values**

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |

**Related functions**

- is_ResetCaptureErrorInfo() 404
- is_GetError() 239
- is_CameraStatus() 172
- is_SetErrorReport() 326

**See also:**

- Troubleshooting 499

## 4.5.6 is_GetDC

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_GetDC (HIDS hCam, HDC* phDC)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_GetDC()` returns the device context handle of the overlay buffer. Using this handle, it is possible to access the overlay using the Windows GDI functionality. Thus, all Windows graphics commands such as `Line, Circle, Rectangle, TextOut,` … are available. You should release the device context handle as early as possible using the `is_ReleaseDC()` 403 function. While a `GetDC...ReleaseDC` block is executed, the overlay buffer on the screen will not be updated.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `phDC` | Pointer to the variable that is supposed to contain the device context handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_ReleaseDC()` 403
- `is_ShowDDOverlay()` 446
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_GetDDOvlSurface()` 389
- `is_SetDisplayMode()` 322

## 4.5.7 is_GetDDOvlSurface

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_GetDDOvlSurface (HIDS hCam, void** ppDDSurf)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_GetDDOvlSurface()` returns the pointer to the internal DirectDraw surface. Thus, the functionality provided by the DirectDraw Surface interface can be used.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ppDDSurf` | Contains the pointer to the DirectDraw Surface interface |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetDisplayMode()` 322
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_ShowDDOverlay()` 446
- `is_HideDDOverlay()` 393

## 4.5.8 is_GetExposureRange

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_GetExposureRange (HIDS hCam, double* min,double* max, double* intervall)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_Exposure()` 216 function instead (see also Obsolete functions 379).

Using `is_GetExposureRange()`, you can query the exposure values (in milliseconds) available for the currently selected timing (pixel clock, frame rate). The available time values are comprised between `min` and `max` and can be set in increments defined by the `intervall` parameter.

**Attention**

The increments for setting the exposure time (`intervall`) depend on the sensor's current timing settings (pixel clock, frame rate). The smallest increment always corresponds to the duration of one pixel row, which is the time it takes the sensor to read out one pixel row.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `min` | Returns the minimum available exposure time. |
| `max` | Returns the maximum available exposure time. |
| `intervall` | Returns the increment you can use to change the image exposure time. |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetExposureTime()` 427
- `is_PixelClock()` 294
- `is_GetFrameTimeRange()` 241
- `is_GetFramesPerSecond()` 240
- `is_SetFrameRate()` 329

## 4.5.9 is_GetGlobalFlashDelays

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_GetGlobalFlashDelays (HIDS hCam, ULONG* pulDelay, ULONG* pulDuration)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

- Rolling shutter cameras:
  Using `is_GetGlobalFlashDelays()`, 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, call the `is_SetGlobalShutter()` function before `is_GetGlobalFlashDelays()`. 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_GetGlobalFlashDelays()` 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 Camera basics: Shutter methods 30 chapter.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pulDelay` | Pointer to the variable that returns the flash delay in µs. |
| `pulDuration` | Pointer to the variable that returns the flash duration in µs. |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetFlashStrobe()` 431
- `is_SetFlashDelay()` 429
- `is_SetTriggerDelay()` 352

## 4.5.10 is_GetPixelClockRange

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_GetPixelClockRange (HIDS hCam, INT* pnMin, INT* pnMax)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_PixelClcok()` 294 function instead (see also Obsolete functions 379).

`is_GetPixelClockRange()` returns the adjustable pixel clock range.

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 460 chapter.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pnMin` | Returns the lower limit value. |
| `pnMax` | Returns the upper limit value. |

### Return values

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

### Related functions

- `is_PixelClock()` 294
- `is_GetFramesPerSecond()` 240
- `is_GetFrameTimeRange()` 241
- `is_Exposure()` 216

## 4.5.11 is_HideDDOverlay

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_HideDDOverlay (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_HideDDOverlay()` hides the overlay. Only the contents of the image buffer will be displayed. This way, the frame rate is higher on some systems than with the overlay shown. By hiding the overlay, its data is not lost.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_ShowDDOverlay()` 446
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_GetDDOvlSurface()` 389
- `is_SetDisplayMode()` 322

## 4.5.12 is_LoadBadPixelCorrectionTable

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_LoadBadPixelCorrectionTable (HIDS hCam, const IS_CHAR* File)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_HotPixel()` 260 function instead (see also Obsolete functions 379).

`is_LoadBadPixelCorrectionTable()` loads a list of sensor hot pixel coordinates that was previously saved using the `is_SaveBadPixelCorrectionTable()` 405 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string which contains the name of the file where the coordinates are stored. You can either pass an absolute or a relative path.<br>If NULL is passed, the "Open File" dialog opens. |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SaveBadPixelCorrectionTable()` 405
- `is_SetBadPixelCorrection()` 417
- `is_SetBadPixelCorrectionTable()` 418

## 4.5.13 is_LoadImage

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_LoadImage (HIDS hCam, char* File)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).

`is_LoadImage()` loads an image from a file. The image must have been saved in BMP format. It is loaded into the active image memory.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a filename You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Open File" dialog opens. |

**Return values**

| | |
|---|---|
| `IS_FILE_READ_INVALID_BMP_ID` | The specified file is not a valid bitmap file. |
| `IS_FILE_READ_INVALID_BMP_SIZE` | The bitmap size is not correct (bitmap too large). |
| `IS_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_NOT_SUPPORTED` | The camera model used here does not support this function or setting. |
| `IS_SUCCESS` | Function executed successfully |

**Related functions**

- `Is_ImageFile()` 264
- `is_GetImageMem()` 247
- `is_SetImageMem()` 337

## 4.5.14 is_LoadImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_LoadImageMem (HIDS hCam, char* File, char** ppcImgMem, int* pid)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).`is_LoadImageMem()` loads an image from a file. The image must have been saved in BMP format. The image, together with its color format and color depth properties, is loaded into a newly allocated image memory.

Using the `is_FreeImageMem()` 222 function, you can release the image memory again.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | File name You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Open File" dialog opens. |
| `ppcImgMem` | Pointer to a variable containing the starting address |
| `pid` | Pointer to a variable containing the memory ID |

**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_FILE_READ_INVALID_BMP_ID` | The specified file is not a valid bitmap file. |
| `IS_FILE_READ_INVALID_BMP_SIZE` | The bitmap size is not correct (bitmap too large). |
| `IS_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_api.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_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_ImageFile() 264
- is_GetImageMem() 247
- is_SetImageMem() 337

## 4.5.15 is_LoadParameters

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_LoadParameters (HIDS hCam, char* pFilename)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ParameterSet()` 292 function instead (see also Obsolete functions 379).

`is_LoadParameters()` loads the parameters for a camera from a uc480 ini file or from the camera EEPROM. Using the `is_SaveParameters()` 412 function, you can save camera parameters in an ini file or in the camera.

Only camera-specific ini files can be loaded. The uc480 parameter file 504 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.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pFilename` | Pointer to a filename. You can either pass an absolute or a relative path. For the internal camera parameter sets, these would be `"\\cam\\set1"` or `"/cam/set1"`, or `"\\cam\\set2"` or `"/cam/set2"`, respectively.<br>If `NULL` is passed, the "Open File" dialog is displayed. |

You can load the parameter sets stored in the camera EEPROM using specific filenames:

| − `pFilename` | |
|---|---|
| `"\\cam\\set1" or "/cam/set1"` | Parameter set 1 |
| `"\\cam\\set2" or "/cam/set2"` | Parameter set 2 |

### 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_CRC_ERROR` | A CRC error-correction problem occurred while reading the settings. |
| `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_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_api.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_PARAMETER_SET_NOT_PRESENT | Parameter set is not present. |
| 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() 292

## 4.5.16 is_LockDDMem

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_LockDDMem (HIDS hCam, void** ppMem, INT* pPitch)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

`is_LockDDMem()` enables access to the image memory in DirectDraw mode and returns the pointer to the image memory. In most cases, the image memory is located on the graphics card. Using the pointer, you have direct access to the image memory. Make sure to release the memory as early as possible using the `is_UnlockDDMem()` 448 function.

Calling `is_LockDDMem()` will not interrupt the process of digitizing an image and saving it to the memory area.

While a `LockDDMem` … `UnlockDDMem` block is executed in DirectDraw BackBuffer mode, the contents of the back buffer will not be refreshed on the screen.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ppMem` | Pointer to the variable that holds the address pointer |
| `pPitch` | Pointer to the variable that holds the pitch value |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_UnlockDDMem()` 448
- `is_LockDDOverlayMem()` 401
- `is_UnlockDDOverlayMem()` 449
- `is_LockSeqBuf()` 289
- `is_UnlockSeqBuf()` 354
- `is_UpdateDisplay()` 450

## 4.5.17 is_LockDDOverlayMem

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_LockDDOverlayMem(HIDS hCam, void** ppMem, INT* pPitch)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_LockDDOverlayMem()` enables access to the overlay memory and returns the pointer to the starting address of the overlay buffer. This way, you can write data directly to the overlay buffer without the need to use the Windows GDI functions.

`pPitch` returns the line offset (in bytes) from the beginning of a line to the beginning of the next line. Make sure to release the memory again as early as possible using the `is_UnlockDDOverlayMem()` 449 function.

While a `LockDDOverlayMem … UnlockDDOverlayMem` block is executed, the contents of the overlay buffer will not be refreshed on the screen.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ppMem` | Pointer to the variable that holds the address pointer |
| `pPitch` | Pointer to the variable that holds the pitch value |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_UnlockDDOverlayMem()` 449
- `is_LockDDMem()` 400
- `is_UnlockDDMem()` 448
- `is_LockSeqBuf()` 289
- `is_UnlockSeqBuf()` 354

## 4.5.18 is_PrepareStealVideo

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_PrepareStealVideo (HIDS hCam, INT Mode, ULONG StealColorMode)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

`is_PrepareStealVideo()` enables steal mode during DirectDraw display. Using the `is_StealVideo()` 447 command, you can remove or copy an image from the DirectDraw video data stream. There are two different steal modes:

- Normal steal
  This option redirects a single frame from a DirectDraw video data stream to the active user memory. The frame will not be displayed with DirectDraw.
- Copy steal
  This option displays the frame with DirectDraw and copies it to the currently active image memory.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `Mode` | |
| `IS_SET_STEAL_NORMAL` | Normal mode |
| `IS_SET_STEAL_COPY` | Copy mode |
| `StealColorMode` | reserved |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_StealVideo()` 447
- `is_SetDisplayMode()` 322
- `is_AllocImageMem()` 157
- `is_SetImageMem()` 337

## 4.5.19 is_ReleaseDC

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_ReleaseDC (HIDS hCam, HDC hDC)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_ReleaseDC()` releases the device context handle of the overlay buffer. If overlay display is enabled using the `is_ShowDDOverlay()` 446 function, the contents of the overlay buffer will be updated on the screen after the handle release.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `hDC` | Device context handle returned by `is_GetDC()` 388 |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_GetDC()` 388
- `is_ShowDDOverlay()` 446
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_GetDDOvlSurface()` 389
- `is_SetDisplayMode()` 322

## 4.5.20 is_ResetCaptureErrorInfo

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_ResetCaptureErrorInfo (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_CaptureStatus()` 174 function instead (see also Obsolete functions 379).

`is_ResetCaptureErrorInfo()` deletes the list of errors that occurred while images were being captured. You can retrieve this list using the `is_GetCaptureErrorInfo()` 386 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_GetCaptureErrorInfo()` 386
- `is_GetError()` 239
- `is_CameraStatus()` 172

## 4.5.21 is_SaveBadPixelCorrectionTable

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SaveBadPixelCorrectionTable (HIDS hCam, const IS_CHAR* File)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_HotPixel()` 260 function instead (see also Obsolete functions 379).

`is_SaveBadPixelCorrectionTable()` saves the user-defined hot pixel list to the specified file.

**Note**

You can only save the hot pixel list if you have previously added user-defined pixels with the `is_SetBadPixelCorrectionTable()` 418 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string which contains the name of the file where the coordinates are stored. You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Save as" dialog will be displayed. |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_LoadBadPixelCorrectionTable()` 394
- `is_SetBadPixelCorrection()` 417
- `is_SetBadPixelCorrectionTable()` 418

## 4.5.22 is_SaveImage

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_SaveImage (HIDS hCam, const IS_CHAR* File)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).

`is_SaveImage()` saves an image in bitmap (*.BMP) format to a file. The images are read out from the current image memory. 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).

**Note**

In Direct3D mode, overlay data are not saved.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string containing the BMP filename You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Save as" dialog will be displayed. |

**Return values**

| | |
|---|---|
| `IS_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_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 |

**Related functions**

- `is_ImageFile()` 264
- `is_GetImageMem()` 247
- `is_SetImageMem()` 337

## 4.5.23 is_SaveImageEx

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_SaveImageEx (HIDS hCam, const IS_CHAR* File, INT fileFormat, INT Param)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).

`is_SaveImageEx()` saves an image in bitmap (*.BMP), JPEG (*.JPG), or PNG format to a file. The images are read out from the current image memory. 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).

**Notes**

- 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.
- In Direct3D mode, overlay data are not saved.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string containing the BMP filename You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Save as" dialog will be displayed. |
| `fileFormat` | Specifies the output format of the file. |
| `IS_IMG_BMP` | Bitmap format |
| `IS_IMG_JPG` | JPEG format |
| `IS_IMG_PNG` | PNG format |
| `Param` | When you use `IS_IMG_JPG` or `IS_IMG_PNG` to specify the file format, you can set the quality by specifying a value between 1 and 100 for Param. If `Param=0`, the system uses the default quality (75).<br>If you use `IS_IMG_BMP`, `Param` does not take effect. |

**Return values**

| | |
|---|---|
| `IS_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_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 |

**Related functions**

- is_ImageFile() 264
- is_GetImageMem() 247
- is_SetImageMem() 337

## 4.5.24 is_SaveImageMem

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_SaveImageMem (HIDS hCam, const IS_CHAR* File, char* pcMem, int nID)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).

`is_SaveImageMem()` saves an image in bitmap (*.BMP) format to a file. The images are read out from the specified image memory. 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).

**Note**

In Direct3D mode, overlay data are not saved.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string containing the BMP filename You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Save as" dialog will be displayed. |
| `pcMem` | Pointer to the image memory |
| `nID` | Image memory ID |

**Return values**

| | |
|---|---|
| `IS_FILE_WRITE_OPEN_ERROR` | File cannot be opened for writing or reading. |
| `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_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 |

**Related functions**

- `is_ImageFile()` 264
- `is_GetImageMem()` 247
- `is_SetImageMem()` 337

## 4.5.25 is_SaveImageMemEx

|  |  |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

```
INT is_SaveImageMemEx (HIDS hCam, const IS_CHAR* File, char* pcMem,
                       int nID, INT fileFormat, INT Param)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ImageFile()` 264 function instead (see also Obsolete functions 379).

`is_SaveImageMemEx()` saves an image in bitmap (*.BMP), JPEG (*.JPG), or PNG format to a file. The images are read out from the specified image memory. 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). JPEG files are always saved with a color depth of 8 or 24 bits.

**Notes**

- 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.
- In Direct3D mode, overlay data are not saved.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `File` | Pointer to a string containing the BMP filename. You can either pass an absolute or a relative path.<br>If `NULL` is passed, the "Save as" dialog will be displayed.. |
| `pcMem` | Pointer to the image memory |
| `nID` | Image memory ID |
| `fileFormat` | Specifies the output format of the file. |
| `IS_IMG_BMP` | Bitmap format |
| `IS_IMG_JPG` | JPEG format |
| `IS_IMG_PNG` | PNG format |
| `Param` | When you use `IS_IMG_JPG` or `IS_IMG_PNG` to specify the file format, you can set the quality by specifying a value between 1 and 100 for Param. If `Param=0`, the system uses the default quality (75).<br>If you use `IS_IMG_BMP`, `Param` does not take effect. |

**Return values**

| | |
|---|---|
| IS_FILE_WRITE_OPEN_ERROR | File cannot be opened for writing or reading. |
| 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_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 |

**Related functions**

- is_ImageFile() 264
- is_GetImageMem() 247
- is_SetImageMem() 337

## 4.5.26 is_SaveParameters

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SaveParameters (HIDS hCam, const IS_CHAR* pFilename)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_ParameterSet()` 292 function instead (see also Obsolete functions 379).

`is_SaveParameters()` saves the current camera parameters to an ini file or to the EEPROM of the camera. You can load saved parameters using the `is_LoadParameters()` 398 function. The uc480 parameter file 504 section in the appendix describes the structure of a uc480 ini file.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `pFileName` | Pointer to a filename You can either pass an absolute or a relative path. For internal parameter sets, these are `"\\cam\\set1"` or `"/cam/set1"`, or `"\\cam\\set2"` or `"/cam/set2"`, respectively.<br>If `NULL` is passed, the "Save as" dialog will be displayed.. |

You can save two parameter sets in the non-volatile EEPROM of the camera using specific filenames:

| `pFileName` | |
|---|---|
| `"\\cam\\set1"` or `"/cam/set1"` | Parameter set 1 |
| `"\\cam\\set2"` or `"/cam/set2"` | Parameter set 2 |

### 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_CRC_ERROR` | A CRC error-correction problem occurred while reading the settings. |
| `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_api.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() 292
- is_CameraStatus() 172

## 4.5.27 is_SetAOI

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetAOI (HIDS hCam, INT type, INT* pXPos, INT* pYPos, INT* pWidth, INT* pHeight)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_AOI()` 159 function instead (see also Obsolete functions 379).

`is_SetAOI()` 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 of automatic white balance control

**Attention**

By default, the window size for auto AOIs is always maximum, i.e. it corresponds to the current image AOI.

After a change to the image geometry (by resetting an image AOI, by binning or sub-sampling), the auto AOIs will always be reset to the image AOI value (i.e. to maximum size). This means that it might be necessary to set the AOIs for the auto features again manually.

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing `is_SetAOI()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()` 329
- `is_SetExposureTime()` 427
- If you are using the DCx Camera's flash function: `is_IO()` 280

**Input parameters**

The `pXPos` and `pYPos` parameters represent an offset with respect to the upper left image corner. The cut window is copied to the start position in the memory. If you want the image to be copied to the same offset within the memory, you can link the new position with a logical OR to the `IS_SET_IMAGEPOS_X_ABS` and `IS_SET_IMAGEPOS_Y_ABS` parameters (Example 416).

| | |
|---|---|
| `hCam` | Camera handle |
| ⊟ `type` | |
| `IS_SET_IMAGE_AOI` | Sets an image AOI. |
| `IS_GET_IMAGE_AOI` | Returns the current image AOI. |
| `IS_SET_AUTO_BRIGHT_AOI` | Sets average AOI values for auto gain and auto shutter. |
| `IS_GET_AUTO_BRIGHT_AOI` | Returns the current auto brightness AOI. |
| `IS_SET_AUTO_WB_AOI` | Sets an auto white balance AOI. |
| `IS_GET_AUTO_WB_AOI` | Returns the current auto white balance AOI. |
| `pXPos` | Pointer to the horizontal position of the AOI<br>Returns the current setting when used together with the `IS_GET_...` parameters. |
| `0...XPosMax`<br>`\| IS_SET_IMAGEPOS_X_ABS` | Applies the absolute position to the memory as well. |
| `pYPos` | Pointer to the vertical position of the AOI<br>Returns the current setting when used together with the `IS_GET_...` parameters. |

| | |
|---|---|
| 0...YPosMax<br>\| IS_SET_IMAGEPOS_Y_ABS | Applies the absolute position to the memory as well. |
| pWidth | Pointer to the width of the AOI<br>Returns the current setting when used together with the IS_GET_... parameters. |
| pHeight | Pointer to the height of the AOI<br>Returns the current setting when used together with the IS_GET_.... parameters. |

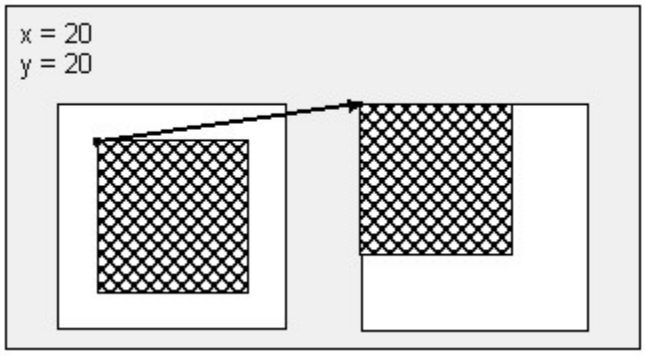
**Return values**

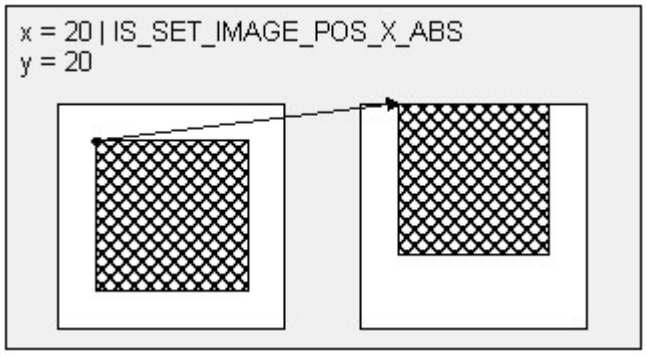
| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |

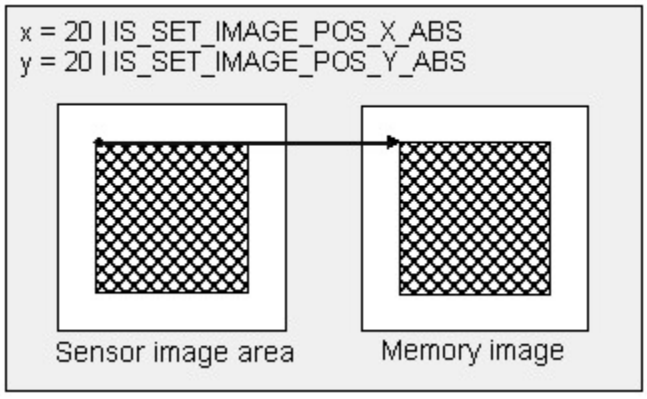
**Related functions**

- is_SetImagePos() 435
- is_SetBinning() 310
- is_SetSubSampling() 347
- is_SetAutoParameter() 303

**Example**



## 4.5.28 is_SetBadPixelCorrection

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetBadPixelCorrection (HIDS hCam, INT nEnable, INT threshold)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_HotPixel()` 260 function instead (see also Obsolete functions 379).

`is_SetBadPixelCorrection()` enables/disables the software correction of sensor hot pixels.

**Attention**

This correction will not work while sub-sampling or binning are enabled or raw Bayer mode is used.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `nEnable` | |
| `IS_BPC_DISABLE` | Disables the correction function. |
| `IS_BPC_ENABLE_SOFTWARE` | Enables software correction based on the hot pixel list stored in the EEPROM. |
| `IS_BPC_ENABLE_USER` | Enables software correction based on user-defined values. First, the `is_SetBadPixelCorrectionTable()` 418 function must be called. |
| `IS_GET_BPC_MODE` | Returns the current mode. |
| `IS_GET_BPC_THRESHOLD` | Returns the current threshold value. |
| `threshold` | Currently not used |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| When used with `IS_GET_BPC_MODE` | Current mode |
| When used with `IS_GET_BPC_THRESHOLD` | Current threshold value |

### Related functions

- `is_LoadBadPixelCorrectionTable()` 394
- `is_SaveBadPixelCorrectionTable()` 405
- `is_SetBadPixelCorrectionTable()` 418

## 4.5.29 is_SetBadPixelCorrectionTable

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetBadPixelCorrectionTable (HIDS hCam, INT nMode, WORD* pList)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_HotPixel()` 260 function instead (see also Obsolete functions 379).

`is_SetBadPixelCorrectionTable()` can be used to set the table containing the hot pixel positions which will be used by the user-defined hot pixel correction function. You can enable hot pixel correction by calling `is_SetBadPixelCorrection()` 417. Each value in the table consists of a 2-byte `WORD` data type. The first value indicates the number of pixel coordinates in the table, the coordinates are listed subsequently (first X, then Y).

A table with 3 hot pixels must be structured as follows:

| 3 | X1 | Y1 | X2 | Y2 | X3 | Y3 |
|---|---|---|---|---|---|---|

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| ⊟ `nMode` | |
| `IS_SET_BADPIXEL_LIST` | Sets a new user-defined list.<br>The `pList` parameter points to a list which has the format described above. |
| `IS_GET_LIST_SIZE` | Returns the number of pixel coordinates included in the user-defined list.<br>If the list contains no coordinates, the return value is -1. |
| `IS_GET_BADPIXEL_LIST` | Copies the user-defined list to the `pList` parameter. Make sure to allocate the memory accordingly. |
| `pList` | Pointer to the starting address of the hot pixel table |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| When used with `IS_GET_LIST_SIZE` | Number of coordinates in the list<br>If the list contains no coordinates, the return value is -1. |

### Related functions

- `is_LoadBadPixelCorrectionTable()` 394
- `is_SaveBadPixelCorrectionTable()` 405
- `is_SetBadPixelCorrection()` 417

### Example

```
WORD *pList = NULL;

// Number of coordinates in the list
INT nCount = is_SetBadPixelCorrectionTable (hCam, IS_GET_LIST_SIZE, NULL);

if (nCount > 0)
{
    // Allocate memory for the entire list
    pList = new WORD[1 + 2 * nCount];

    // Read out list
    is_SetBadPixelCorrectionTable (hCam, IS_GET_BADPIXEL_LIST, pList);

    // Release the list again
    delete [] pList;
}
else
{
    // No hot pixel list programmed
}
```

## 4.5.30 is_SetBayerConversion

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetBayerConversion (HIDS hCam, INT nMode)
```

### Description

`is_SetBayerConversion()` enables you to select one of two algorithms for the Bayer conversion. These algorithms vary in the obtainable quality and in the required computer load.

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_SetColorConverter()` 315 function instead (see also Obsolete functions 379).

**Note**

This function can be used only for the 24 bit, 32 bit and Y8 color formats (color cameras).

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `nMode` | |
| `IS_SET_BAYER_CV_BETTER` | Good quality, minor color artifacts, lower computational load |
| `IS_SET_BAYER_CV_BEST` | Best quality and edge acuity, higher computational load |
| `IS_GET_BAYER_CV_MODE` | Returns the current setting. |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| When used with `IS_GET_BAYER_CV_MODE` | Current setting |

### Related functions

- `is_SetColorConverter()` 315
- `is_SetColorMode()` 319
- `is_SetColorCorrection()` 317

## 4.5.31 is_SetBlCompensation

| | |
|---|---|
| USB 2.0<br>USB 3.0 | USB 2.0<br>USB 3.0 |

**Syntax**

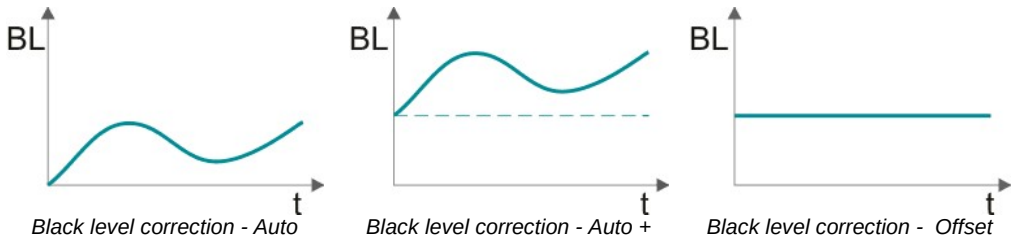
```
INT is_SetBlCompensation (HIDS hCam, INT nEnable, INT offset, INT reserved)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_Blacklevel()` 170 function instead (see also Obsolete functions 379).

`is_SetBlCompensation()` enables the black level correction function 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 by the `offset` parameter.




*Black level correction - Auto* *Black level correction - Auto + offset* *Black level correction - Offset*

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nEnable` | |
| `IS_BL_COMPENSATION_DISABLE` | Disables automatic black level correction. The offset value is used as black level instead. This mode is only supported by sensors of the UI-154x/UI-554x 464 series. |
| `IS_BL_COMPENSATION_ENABLE` | Enables automatic black level correction. The offset value is added to the automatic black level value. |
| `IS_GET_BL_COMPENSATION` | Returns the current mode. |
| `IS_GET_BL_OFFSET` | Returns the currently set value for `offset`. |
| `IS_GET_BL_DEFAULT_MODE` | Returns the default mode. |
| `IS_GET_BL_DEFAULT_OFFSET` | Returns the default value for `offset`. |
| `IS_GET_BL_SUPPORTED_MODE` | Returns the supported modes.<br>Possible values:<br>`IS_BL_COMPENSATION_ENABLE`<br>The sensor supports automatic black level correction.<br>`IS_BL_COMPENSATION_OFFSET`<br>For the sensor used, it is also possible to set the `offset` manual. |
| `IS_IGNORE_PARAMETER` | The `nEnable` parameter is ignored. |
| `offset` | Contains the offset value used for compensation. Valid values are between 0 and 255. |
| `IS_IGNORE_PARAMETER` | The `offset` parameter is ignored. |
| `reserved` | Reserved. 0 must be passed. |

**Return values**

| | |
|---|---|
| When used together with IS_GET_BL_SUPPORTED_MODE | Supported modes |
| When used together with IS_GET_BL_COMPENSATION | Current mode |
| When used together with IS_GET_BL_OFFSET | Current offset |
| 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_api.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. |

## 4.5.32 is_SetConvertParam

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetConvertParam (HIDS hCam,
                        BOOL ColorCorrection,
                        INT BayerConversionMode, INT ColorMode, INT Gamma,
                        double* WhiteBalanceMultipliers)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_Convert()` 188 function instead (see also Obsolete functions 379).

Using `is_SetConvertParam()`, you can set the parameters for converting a raw Bayer image to a color image. To convert the image, use the `is_ConvertImage()` 382 function.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ColorCorrection` | Enables/disables color correction. |
| `BayerConversionMode` | Sets the Bayer conversion mode. |
| `IS_SET_BAYER_CV_BETTER` | Normal quality |
| `IS_SET_BAYER_CV_BEST` | Best quality (higher CPU load) |
| `ColorMode` | Sets the color mode for the output image.<br>For a list of all available color formats and the associated input parameters, see the Appendix: Color and memory formats 502 section. |
| `Gamma` | Gamma value multiplied by 100. Range: [1…1000] |
| `WhiteBalanceMultipliers` | Pointer to an array containing the red, green and blue gain values |

### Return values

| | |
|---|---|
| `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_NO_SUCCESS` | General error message |
| `IS_SUCCESS` | Function executed successfully |

### Related functions

- `is_ConvertImage()` 382
- `is_SetColorMode()` 319
- `is_SetColorConverter()` 315

### Example

Conversion of a raw Bayer image to RGB24. The memory is allocated automatically.

```
INT nRet;
char * pcSource;
INT nIDSource;
INT nX,nY,nBits,nPitch;

// Create raw Bayer test image
is_AllocImageMem (hCam, 256, 256, 8, &pcSource, &nIDSource);
is_InquireImageMem (hCam, pcSource, nIDSource, &nX ,&nY, &nBits, &nPitch);
for (int j = 0; j<nY; j++)
{
    for (int i = 0; i<nX; i++)
    {
        pcSource[i + j * nPitch] = i;
    }
}

// Define conversion parameters (example)
INT Gamma = 120;
double rgbGains[3];
rgbGains[0] = 1.0 ; // Red channel gain
rgbGains[1] = 3.0 ; // Green channel gain
rgbGains[2] = 1.0 ; // Blue channel gain

char* pcDest;  // Pointer to the newly allocated image memory
INT nIDDest;  // ID of the newly allocated image memory


// Set conversion parameters
nRet = is_SetConvertParam(hCam, TRUE, IS_SET_BAYER_CV_BETTER, IS_CM_RGB8_PACKED, Gamma, rgbGains);

// Convert image
if (nRet == IS_SUCCESS)
{
    pcDest = NULL;
    is_ConvertImage(hCam, pcSource, nIDSource, &pcDest, &nIDDest, 0);
}

// Release allocated image memory
is_FreeImageMem (hCam, pcSource, nIDSource);
is_FreeImageMem (hCam, pcDest, nIDDest);
```

## 4.5.33 is_SetDDUpdateTime

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_SetDDUpdateTime (HIDS hCam, INT ms)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

`is_SetDDUpdateTime()` sets the timer interval used for the video image update cycle in DirectDraw BackBuffer mode.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `ms` | Time in milliseconds. Valid range: 20...2000 ms |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetDisplayMode()` 322
- `is_UpdateDisplay()` 450

## 4.5.34 is_SetEdgeEnhancement

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetEdgeEnhancement (HIDS hCam, INT nEnable)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_EdgeEnhancement()` 206 function instead (see also Obsolete functions 379).

`is_SetEdgeEnhancement()` enables 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.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| ⊟ `nEnable` | |
| `IS_EDGE_EN_DISABLE` | Disables the edge filter. |
| `IS_EDGE_EN_STRONG` | Enables strong edge enhancement. |
| `IS_EDGE_EN_WEAK` | Enables weaker edge enhancement. |
| `IS_GET_EDGE_ENHANCEMENT` | Returns the current setting. |

### Return values

| | |
|---|---|
| When used together with `IS_GET_EDGE_ENHANCEMENT` | 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()` 319
- `is_SetColorConverter()` 315

## 4.5.35 is_SetExposureTime

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetExposureTime (HIDS hCam, double EXP, double* newEXP)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_Exposure()` 216 function instead (see also Obsolete functions 379).

Using `is_SetExposureTime()`, you can set the exposure time (in milliseconds). Since this value depends on the sensor timing, the exposure time actually used may slightly deviate from the value set here. The actual exposure time is returned by the newEXP parameter.

In free-running mode (`is_CaptureVideo()` 177), any modification of the exposure time will only become effective when the next image but one is captured. In trigger mode (`is_SetExternalTrigger()` 327), the modification will be applied to the next image.

For minimum and maximum exposure times as well as other sensor-based dependencies, please refer to Camera and sensor data 460 chapter.

**Notes**

**1. New Driver Versions**

Newer driver versions sometimes allow an extended value range for the exposure time setting. We recommend to query the value range every time and set the exposure time explicitly.

**2. Settings with Influence on Exposure Time**

The use of the following functions will affect the exposure time:

- `is_PixelClock()` 294
- `is_SetOptimalCameraTiming()` 338
- `is_SetFrameRate()` 329 (if the new image duration is shorter that the exposure time)
- `is_AOI()` 159 (if the image size is changed)
- `is_SetSubSampling()` 347
- `is_SetBinning()` 310

Changes made to the window size, the frame rate or the read-out timing (pixel clock frequency) also affect the defined exposure time. For this reason, you need to call `is_SetExposureTime()` again after such changes.

**3. Exposure Time Increment**

The increments for setting the exposure time (`IS_GET_EXPOSURE_INCREMENT`) depend on the sensor's current timing settings (pixel clock, frame rate). The smallest increment always corresponds to the duration of one pixel row, which is the time it takes the sensor to read out one pixel row.

Some sensors allow setting the exposure time with a finer increment (`IS_GET_EXPOSURE_FINE_INCREMENT`). This function is currently only supported by the sensors of the DCC1240x / DCC3240x 461 models.

**4. Rounding Errors**

When calculating a new exposure time based on the `..._INCREMENT` parameter, please note that calculations with floating point values in the PC will always be subject to rounding errors. Therefore, an addition or subtraction of an `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 or subtracting the value `INCREMENT/2.f` (half intervall) when calculating with `n*INCREMENT`. This ensures that the desired value will be set even after rounding.

### Input parameters

| | |
|---|---|
| hCam | Camera handle |
| EXP | New desired exposure time<br>For EXP=0.0, the exposure time is 1/frame rate. |
| IS_GET_EXPOSURE_TIME | Returns the current exposure time in the newEXP parameter. |
| IS_GET_DEFAULT_EXPOSURE | Returns the default exposure time. |
| IS_SET_ENABLE_AUTO_SHUTTER | Enables the auto exposure function (see also is_SetAutoParameter() 303). |
| IS_GET_EXPOSURE_MIN_VALUE | Returns the minimum available exposure time. |
| IS_GET_EXPOSURE_MAX_VALUE | Returns the maximum available exposure time. |
| IS_GET_EXPOSURE_INCREMENT | Returns the increment you can use to change the image exposure time. |
| IS_GET_EXPOSURE_FINE_INCREMENT | Returns a finer exposure time increment for some sensors. |
| newEXP | When setting a new exposure time via EXP:<br>Returns the exposure time actually set (may deviate from desired exposure time)<br>When passing other parameters via EXP:<br>Returns the queried value (e. g. current exposure time) |

### Return values

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| IS_NOT_SUPPORTED | The camera model used here does not support this function or setting. |

### Related functions

- is_SetFrameRate() 329
- is_Exposure() 216
- is_PixelClock() 294
- is_SetOptimalCameraTiming() 338
- is_SetAutoParameter() 303

## 4.5.36 is_SetFlashDelay

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetFlashDelay (HIDS hCam, ULONG ulDelay, ULONG ulDuration)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

`is_SetFlashDelay()` sets a delay for driving the flash output. In addition, you can specify the flash duration.

This allows the implementation of a global flash functionality which exposes all rows of a rolling shutter sensor. 31 In addition, it is possible, for a camera with global shutter sensors, 30 to set the flash start in free-run mode to the start time of the exposure window.

For further information, please refer to the Digital in-/output (trigger/flash) 47 and Shutter methods 30 chapters.

**Note**

**Accuracy of Flash Synchronization**

The following parameters have an influence on the camera's internal timing:

- Image geometry 144 (CMOS and CCD sensors)
- Pixel clock 294 (CMOS and CCD sensors)
- Exposure time 216 (CCD sensors)

If you change any of these parameters, you will have to set the flash duration and flash delay parameters once again.

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `ulDelay` | Time by which the flash start is delayed (in µs)<br>0: No delay<br>For `0 < ulDelay < 40` the driver automatically sets `ulDelay = 40` |
| `IS_GET_FLASH_DELAY` | Returns the currently set delay time. |
| `IS_GET_FLASH_DURATION` | Returns the currently set flash duration. |
| `IS_GET_MIN_FLASH_DELAY` | Returns the minimum value for the delay. |
| `IS_GET_MIN_FLASH_DURATION` | Returns the minimum value for the flash duration. |
| `IS_GET_MAX_FLASH_DELAY` | Returns the maximum value for the delay. |
| `IS_GET_MAX_FLASH_DURATION` | Returns the maximum value for the flash duration. |
| `IS_GET_FLASH_DELAY_GRANULARITY` | Returns the increment of the adjustable delay time. |
| `IS_GET_FLASH_DURATION_GRANULARITY` | Returns the increment of the adjustable flash duration. |
| `ulDuration` | Time during which the flash is on (in µs).<br>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 last sensor row |

### Return values

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| When used with<br>IS_GET_FLASH_DELAY<br>IS_GET_FLASH_DURATION | Current setting |

### Related functions

- is_IO() 280
- is_SetExternalTrigger() 327
- is_SetTriggerDelay() 352

## 4.5.37 is_SetFlashStrobe

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetFlashStrobe (HIDS hCam, INT nMode, INT nLine)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

`is_SetFlashStrobe()` controls the DCx Camera's digital outputs and defines the flash strobe behavior. 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.

For details on flash timing, see the Digital input/output (trigger/flash) 47 and Operating modes 17 chapters.

**Notes**

**Duration and Trigger Delay**

You can set the flash duration and flash delay using the `is_SetFlashDelay()` 429 function. The `is_GetGlobalFlashDelays()` 391 returns the values required for precisely synchronizing the flash signal to the current timing settings.

**Accuracy of Flash Synchronization**

The following parameters have an influence on the camera's internal timing:

- Image geometry 144 (CMOS and CCD sensors)
- Pixel clock 294 (CMOS and CCD sensors)
- Exposure time 216 (CCD sensors)

If you change any of these parameters, you will have to set the flash duration and flash delay parameters once again.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `nMode` (Flash mode) | |
| **Flash strobe with exposure time synchronization** | |
| `IS_SET_FLASH_OFF` | Disables the digital output. |
| `IS_SET_FLASH_LO_ACTIVE` | Enables the flash strobe in trigger mode.<br>`LO_ACTIVE`: The digital output is set to low level for the flash duration. |
| `IS_SET_FLASH_HI_ACTIVE` | Enables the flash strobe in trigger mode.<br>`HI_ACTIVE`: The digital output is set to high level for the flash duration. |
| `IS_SET_FLASH_LO_ACTIVE_FREERUN` | Enables the flash strobe in freerun mode.<br>`LO_ACTIVE`: The digital output is set to low level for the flash duration. |
| `IS_SET_FLASH_HI_ACTIVE_FREERUN` | Enables the flash strobe in freerun mode.<br>`HI_ACTIVE`: The digital output is set to high level for the flash duration. |
| `IS_GET_FLASHSTROBE_MODE` | Returns the current mode. |
| `IS_SET_FLASH_IO_1` | Additionally enables the flash strobe via the first general purpose I/O (GPIO 1) if supported by the camera model. |
| `IS_SET_FLASH_IO_2` | Additionally enables the flash strobe via the second general purpose I/O (GPIO 2) if supported by the |

| | |
|---|---|
| hCam | Camera handle |
| | camera model. |
| IS_GET_SUPPORTED_FLASH_IO_PORTS | Returns which of the general purpose I/Os (GPIO) can be used for the flash strobe |
| **Statically setting the output level** | |
| IS_SET_FLASH_HIGH | Statically sets the digital output to high level (HIGH). |
| IS_SET_FLASH_LOW | Statically sets the digital output to low level (LOW). |
| nLine | Currently not used |

**Return values**

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| When used with IS_GET_FLASHSTROBE_MODE | Current setting |
| When used with IS_GET_SUPPORTED_FLASH_IO_PORTS | Returns the supported ports linked by a logical OR.<br>IS_SET_FLASH_IO_1<br>IS_SET_FLASH_IO_2 |

**Related functions**

- is_SetFlashDelay() 429
- is_IO() 280
- is_SetExternalTrigger() 327
- is_SetTriggerDelay() 352
- is_CaptureVideo() 177

**Example**

```
//Software trigger mode, HI_ACTIVE flash mode
is_SetExternalTrigger (hCam, IS_SET_TRIGGER_SOFTWARE);
is_SetFlashStrobe (hCam, IS_SET_FLASH_HI_ACTIVE, 0);
is_FreezeVideo (hCam, IS_WAIT);

// Flash additionally on second GPIO output
is_SetFlashStrobe (hCam, IS_SET_FLASH_HI_ACTIVE | IS_SET_FLASH_IO_2, 0);

// Flash only on normal flash output
is_SetFlashStrobe (hCam, IS_SET_FLASH_HI_ACTIVE, 0);
```

**Sample programs**

- uc480 Flash Strobe (C++)
- uc480 IO (C++)

## 4.5.38 is_SetHwnd

|  |  |
|---|---|
| USB 2.0 | |

### Syntax

```
INT is_SetHwnd (HIDS hCam, HWND hwnd)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

`is_SetHwnd()` sets a new window handle for image output in DirectDraw mode. The new handle and the image output will only be effective when `is_SetDisplayMode()` 322 is called for the next time.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `hwnd` | Window handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetDisplayMode()` 322

## 4.5.39 is_SetImageAOI

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

### Syntax

```
INT is_SetImageAOI (HIDS hCam,
                    INT xPos, INT yPos,
                    INT width, INT height)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the is_AOI() 159 function instead (see also Obsolete Functions 379).

### Input parameters

| | |
|---|---|
| hCam | Camera handle |
| xPos | X position of the upper left corner. |
| yPos | Y position of the upper left corner. |
| width | Image width |
| height | Image height |

### Return values

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| IS_INVALID_MODE | Camera is in standby mode, function not allowed |

### Related functions

- is_AOI() 159
- is_SetBinning() 310
- is_SetSubSampling() 347

## 4.5.40 is_SetImagePos

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetImagePos (HIDS hCam, INT x, INT y)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_AOI()` 159 function instead (see also Obsolete functions 379).

`is_SetImagePos()` determines the position of an area of interest (AOI) in the display window. When used together with the `is_SetAOI()` 414 function, you can cut out an area of interest of the full video image.

To avoid a positional mismatch between the display area and the image area, make sure to call the functions in the correct order. Starting from the original image, it is mandatory to keep to the following order:

1. `is_SetAOI()` 414
2. `is_SetImagePos()`

**Attention**

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing `is_SetImagePos()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()` 329
- `is_SetExposureTime()` 427
- If you are using the DCx Camera's flash function: `is_SetFlashStrobe()` 431

**Input parameters**

The x and y parameters represent an offset with respect to the upper left image corner. The cut window is copied to the start position in the memory. If you want the image to be copied to the same offset within the memory, you can link the new position with a logical OR to the `IS_SET_IMAGE_POS_X_ABS` and `IS_SET_IMAGE_POS_Y_ABS` parameters.

| hCam | Camera handle |
|---|---|
| x | |
| 0...xMax | Sets the horizontal position |
| 0...xMax<br>\| IS_SET_IMAGE_POS_X_ABS | Applies the absolute position to the memory as well. |
| IS_GET_IMAGE_POS_X | Returns the current x position. |
| IS_GET_IMAGE_POS_X_MIN | Returns the minimum value for the horizontal AOI position. |
| IS_GET_IMAGE_POS_X_MAX | Returns the maximum value for the horizontal AOI position. |
| IS_GET_IMAGE_POS_X_INC | Returns the increment for the horizontal AOI position. |
| IS_GET_IMAGE_POS_X_ABS | Returns the absolute horizontal position in the memory. |
| IS_GET_IMAGE_POS_Y | Returns the current Y position. |
| IS_GET_IMAGE_POS_Y_MIN | Returns the minimum value for the vertical AOI position. |
| IS_GET_IMAGE_POS_Y_MAX | Returns the maximum value for the vertical AOI position. |
| IS_GET_IMAGE_POS_Y_INC | Returns the increment for the vertical AOI position. |
| IS_GET_IMAGE_POS_Y_ABS | Returns the absolute vertical position in the memory. |

| y | |
|---|---|
| 0...yMax | Sets the vertical position |
| 0...yMax<br>\| IS_SET_IMAGE_POS_Y_ABS | Applies the absolute position to the memory as well. |
| 0 | When returning settings via parameter x (s. above) |

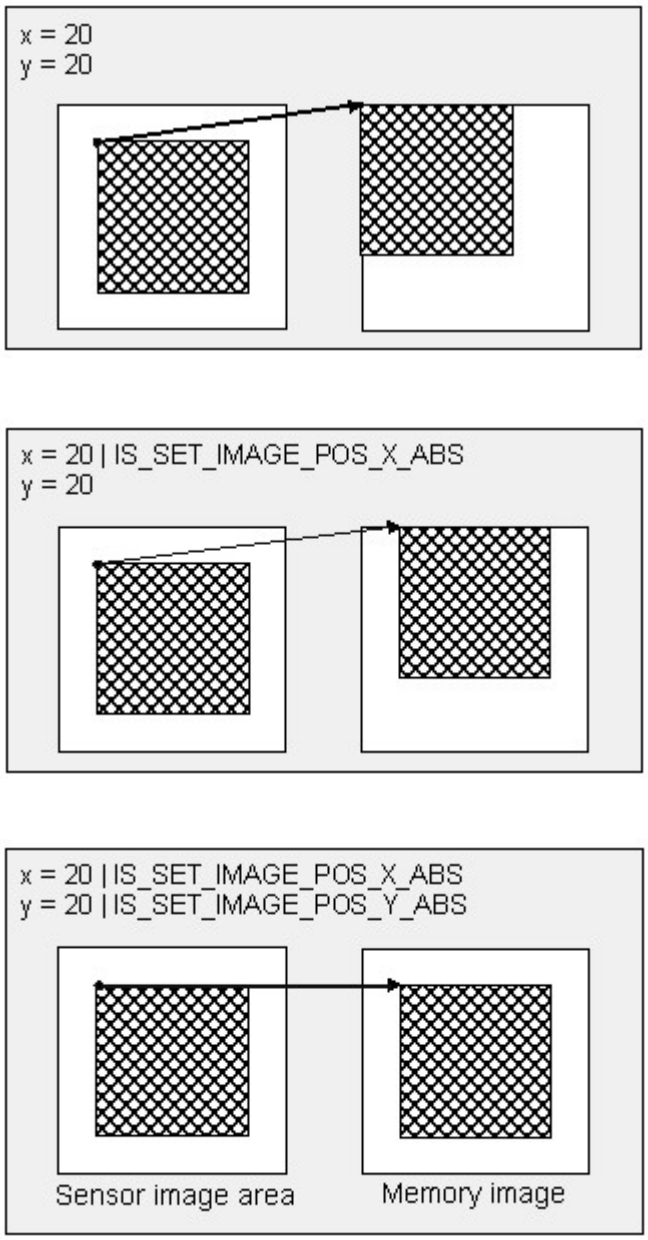
**Return values**

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| 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. |
| When used with IS_GET_IMAGE_POS parameters | Current setting |
| IS_INVALID_MODE | Camera is in standby mode, function not allowed |

**Related functions**

- is_AOI() 159
- is_ImageFormat() 267

**Example**




*Examples for is_SetImagePos*

## 4.5.41 is_SetImageSize

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetImageSize (HIDS hCam, INT x, INT y)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the is_AOI() 159 function instead (see also Obsolete functions 379).

In conjunction with the is_SetImagePos() 435 settings, is_SetImageSize() determines the size of the captured area of interest (AOI).

To avoid a positional mismatch between the display area and the image area, make sure to call the functions in the correct order. Starting from the original image, it is mandatory to keep to the following order:

1.is_SetImageSize()

2.is_SetImagePos() 435

**Hint**

is_SetAOI() 414 combines both functions. With is_SetAOI(), you can set the position and size of an area of interest using a single function call.

**Note**

Changes to the image geometry or pixel clock affect the value ranges of the frame rate and exposure time. After executing is_SetImageSize(), calling the following functions is recommended in order to keep the defined camera settings:

- is_SetFrameRate() 329
- is_Exposure() 216
- If you are using the DCx Camera's flash function: is_IO() 280

**Input parameters**

| hCam | Camera handle |
|---|---|
| x | |
| 1...xMax | Sets the image width |
| IS_GET_IMAGE_SIZE_X | Returns the current image width. |
| IS_GET_IMAGE_SIZE_X_MIN | Returns the minimum AOI image width. |
| IS_GET_IMAGE_SIZE_X_MAX | Returns the maximum AOI image width. |
| IS_GET_IMAGE_SIZE_X_INC | Returns the increment for the AOI image width. |
| IS_GET_IMAGE_SIZE_Y | Returns the current image height. |
| IS_GET_IMAGE_SIZE_Y_MIN | Returns the minimum AOI image height |
| IS_GET_IMAGE_SIZE_Y_MAX | Returns the maximum AOI image height |
| IS_GET_IMAGE_SIZE_Y_INC | Returns the increment for the AOI image height |
| y | |
| 1...yMax | Sets the image height |
| 0 | Return settings |

**Return values**

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |
| When used with IS_GET_IMAGE_SIZE parameters | Current setting |
| IS_INVALID_MODE | Camera is in standby mode, function not allowed |

**Related functions**

- is_AOI() 159

## 4.5.42 is_SetIO

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetIO (HIDS hCam, INT nIO)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

`is_SetIO()` sets the additional digital outputs (GPIOs) of the DCx Camera or returns their current states. Using `is_SetIOMask()` 441, you can define each GPIO as a digital input or output.

**Attention**

To be able to set the Status of a GPIO you must first configure the GPIO as output using `is_SetIOMask()` 441. If only one GPIO is configured as output the command `is_SetIO()` has no effect on the other GPIO.

The GPIOs are not provided with optocouplers and use TTL voltages. For information on GPIO wiring, please refer to the Electrical Specifications 487 chapter.

**Note**

To connect and control a flash (strobe) unit for the DCx Cameras, it is recommended to use the flash output provided (see `is_SetFlashStrobe()` 431).

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nIO` | Bit mask for outputs |
| `0x00 (00)` | Sets both outputs to 0. |
| `0x01 (01)` | Sets the first output to 1, the second one to 0. |
| `0x02 (10)` | Sets the first output to 0, the second one to 1. |
| `0x03 (11)` | Sets both outputs to 1. |
| `IS_GET_IO` | Reads the states of the GPIOs. If a GPIO is configured as input this reads the signal applied to the GPIO. |

**Return values**

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| When used with `IS_GET_IO` | Current setting |

**Related functions**

- `is_SetIOMask()` 441
- `is_GetImageInfo()` 244
- `is_SetFlashStrobe()` 431
- `is_SetExternalTrigger()` 327

## 4.5.43 is_SetIOMask

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetIOMask (HIDS hCam, INT nMask)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

Using `is_SetIOMask()`, you can configure the general purpose I/Os (GPIO) of some DCx Camera models as an input or output. The `is_SetIO()` 440 function sets or returns the current GPIO states.

The GPIOs are not provided with optocouplers and use TTL voltages. For information on GPIO wiring, please refer to the Electrical Specifications 487 chapter.

**Note**

To use hardware triggering with the DCx Cameras, we suggest that you use the trigger input provided for this purpose (see `is_SetExternalTrigger()` 327). To connect and control a flash (strobe) unit for the DCx Cameras, it is recommended to use the flash output provided (see `is_SetFlashStrobe()` 431).**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nMask` | Bit mask for inputs/outputs. |
| `0x00 (00)` | Use both GPIOs as inputs. |
| `0x01 (01)` | Use the first GPIO as output, the second one as input. |
| `0x02 (10)` | Use the first GPIO as input, the second one as output. |
| `0x03 (11)` | Use both GPIOs as outputs. |
| `IS_GET_IO_MASK` | Returns the current bit mask. |
| `IS_GET_INPUT_MASK` | Returns the IOs to be used as inputs. |
| `IS_GET_OUTPUT_MASK` | Returns the IOs to be used as outputs. |

**Return values**

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| Wen used with `IS_GET_IO` | Current setting |
| When used with `IS_GET_INPUT_MASK` `IS_GET_OUTPUT_MASK` | Bit masks of the IOs to be used. |

**Related functions**

- `is_SetIO()` 440
- `is_GetImageInfo()` 244
- `is_SetFlashStrobe()` 431
- `is_SetExternalTrigger()` 327

## 4.5.44 is_SetKeyColor

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_SetKeyColor (HIDS hCam, INT r, INT g, INT b)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

Using `is_SetKeyColor()`, you define the key color for DirectDraw Overlay Surface mode.

This function can also be used to return the key color. Use the `r` parameter to specify the color value to be returned. Depending on the call type, the function returns either a value reflecting the color content (0...255) or the corresponding RGB value (0 ... 16777215).

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |
| `r` | Red content of the key color (0...255). |
| `IS_GET_KC_RED` | The function returns the red content value. |
| `IS_GET_KC_GREEN` | The function returns the green content value. |
| `IS_GET_KC_BLUE` | The function returns the blue content value. |
| `IS_GET_KC_RGB` | The function returns the RGB color. |
| `g` | Green content of the key color (0...255). |
| `b` | Blue content of the key color (0...255). |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |
| When used with<br>`IS_GET_KC_RGB`<br>`IS_GET_KC_RED`<br>`IS_GET_KC_GREEN`<br>`IS_GET_KC_BLUE` | Color value |

### Related functions

- `is_SetDisplayMode()` 322
- `is_ShowDDOverlay()` 446
- `is_HideDDOverlay()` 393
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_GetDDOvlSurface()` 389
- `is_SetColorMode()` 319

## 4.5.45 is_SetLED

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetLED (HIDS hCam, INT nValue)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_IO()` 280 function instead (see also Obsolete functions 379).

Using `is_SetLED()`, you can toggle the color of the LED on the back of the USB DCx camera housing.

**Note**

The `is_SetLED()` function is only supported by DCU22x and DCC1240x

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `nValue` | |
| `IS_SET_LED_OFF` | Switches LED to red. |
| `IS_SET_LED_ON` | Switches LED to green. |
| `IS_SET_LED_TOGGLE` | Toggles between red and green. |

**Return values**

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

**Related functions**

- `is_IO()` 280
- `is_SetExternalTrigger()` 327

## 4.5.46 is_SetPixelClock

|  |  |
|---|---|
| USB 2.0 | USB 2.0 |

**Syntax**

```
INT is_SetPixelClock (HIDS hCam, INT Clock)
```

**Description**

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_PixelClock()` 294 function instead (see also Obsolete functions 379).

`is_SetPixelClock()` sets the frequency used to read out image data from the sensor (pixel clock frequency). 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.

**Note**

Some sensors allow a higher pixel clock setting when if 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_SetPixelClock()`, calling the following functions is recommended in order to keep the defined camera settings:

- `is_SetFrameRate()` 329
- `is_Exposure()` 216
- If you are using the DCx Camera's flash function: `is_IO()` 280

**Input parameters**

| | |
|---|---|
| `hCam` | Camera handle |
| `Clock` | Pixel clock frequency to be set (in MHz) |
| `IS_GET_PIXEL_CLOCK` | Returns the current pixel clock |
| `IS_GET_DEFAULT_PIXEL_CLK` | Returns the default pixel clock |
| `IS_GET_PIXEL_CLOCK_INC` | Returns the pixel clock increment |

**Return values**

| | |
|---|---|
| When used with `IS_GET_PIXEL_CLOCK` | Current setting |
| When used with `IS_GET_DEFAULT_PIXEL_CLK` | Default 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_api.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_SetOptimalCameraTiming() 338
- is_SetFrameRate() 329
- is_Exposure() 216
- is_SetAutoParameter() 303
- is_SetBinning() 310
- is_SetSubSampling() 347
- is_AOI() 159

## 4.5.47 is_ShowDDOverlay

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_ShowDDOverlay (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

In DirectDraw BackBuffer mode, `is_ShowDDOverlay()` displays the overlay, i.e. the most recent data stored in the overlay buffer. With some graphics cards, overlay display may reduce the frame rate.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_HideDDOverlay()` 393
- `is_DisableDDOverlay()` 383
- `is_EnableDDOverlay()` 384
- `is_GetDDOvlSurface()` 389
- `is_SetDisplayMode()` 322

## 4.5.48 is_StealVideo

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

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

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

Using `is_StealVideo()`, you can initiate the extraction of an image from the DirectDraw image data stream. The extracted image is written to the active image memory. The data includes the color format set with the `is_SetColorMode()` 319 function.

You can use the `is_PrepareStealVideo()` 402 function to specify to remove the image from the DirectDraw data stream or to copy it. If you set the copy option, the image will both be displayed using DirectDraw and copied to the currently active image memory.

See also the Events in Live Mode figure in the Event handling 141 section.

**Attention**

This function is not supported in Direct3D mode.

### Input parameters

| | |
|---|---|
| hCam | Camera handle |
| Wait | |
| IS_WAIT | The function waits until the image save is complete. |
| IS_DONT_WAIT | The function returns immediately. |

### Return values

| | |
|---|---|
| IS_SUCCESS | Function executed successfully |
| IS_NO_SUCCESS | General error message |

### Related functions

- `is_PrepareStealVideo()` 402
- `is_SetDisplayMode()` 322
- `is_SetColorMode()` 319
- `is_AllocImageMem()` 157
- `is_SetImageMem()` 337
- `is_SetAllocatedImageMem()` 301

## 4.5.49 is_UnlockDDMem

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_UnlockDDMem (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

Using `is_UnlockDDMem()`, you can unlock the image memory in the DirectDraw modes. This results in a refresh of the BackBuffer contents on the screen.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_LockDDMem()` 400
- `is_LockDDOverlayMem()` 401
- `is_UnlockDDOverlayMem()` 449
- `is_LockSeqBuf()` 289
- `is_UnlockSeqBuf()` 354
- `is_UpdateDisplay()` 450

## 4.5.50 is_UnlockDDOverlayMem

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_UnlockDDOverlayMem (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

Using `is_UnlockDDOverlayMem()`, you can unlock the overlay buffer in DirectDraw BackBuffer mode. This results in an overlay buffer refresh on the screen, provided that the overlay display was enabled using `is_ShowDDOverlay()` 446.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_LockDDOverlayMem()` 401
- `is_UnlockDDOverlayMem()` 449
- `is_LockDDMem()` 400
- `is_UnlockDDMem()` 448
- `is_LockSeqBuf()` 289
- `is_UnlockSeqBuf()` 354
- `is_UpdateDisplay()` 450

## 4.5.51 is_UpdateDisplay

|  |  |
|---|---|
| USB 2.0 | - |

### Syntax

```
INT is_UpdateDisplay (HIDS hCam)
```

### Description

**Attention**

This function is obsolete and should not be used anymore. We recommend to use the `is_DirectRenderer()` 198 function instead (see also Obsolete functions 379).

Using `is_UpdateDisplay()`, you can manually update the screen output in the DirectDraw modes. In normal operation, the driver performs updates automatically. There may be cases, however, where a manual update of the screen output is necessary.

### Input parameters

| | |
|---|---|
| `hCam` | Camera handle |

### Return values

| | |
|---|---|
| `IS_SUCCESS` | Function executed successfully |
| `IS_NO_SUCCESS` | General error message |

### Related functions

- `is_SetDisplayMode()` 322
- `is_LockDDOverlayMem()` 401
- `is_UnlockDDOverlayMem()` 449
- `is_LockDDMem()` 400
- `is_UnlockDDMem()` 448

## 4.6 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++ 451
- Programming in C# 452
- Programming in VB.NET 452
- Programming in Delphi 452
- Programming with ActiveX 453
- Thread programming 453


### 4.6.1 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\.`

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_api.lib
- Function library (DLL): `uc480_Api.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_api.so`

## 4.6.2 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_Api.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.

## 4.6.3 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 453 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.

## 4.6.4 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) 453 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.

### 4.6.5 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.

### 4.6.6 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**

**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()` 273
- `is_SetDisplayMode()` 322
- `is_ExitCamera()` 213

**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.

# 4.7 Lists



## 4.7.1 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<br>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_api.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. |
| 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 |

| No | Error | Description |
|---|---|---|
| 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. |
| 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. |

| No | Error | Description |
|---|---|---|
| 176 | IS_INVALID_I2C_DEVICE_ADDRESS | Invalid $I^2C$ 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. |
| 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. |

## 4.7.2 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() 362. |
| 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. |

## 4.7.3 Linux: Not Supported Functions

The following uc480 API functions are not supported by the Linux driver version 4.20 and older:

- is_DirectRenderer() 198
- is_GetColorDepth() 237
- is_GetDC() 388
- is_ReleaseDC() 403
- is_RenderBitmap() 297
- is_SetDisplayMode() 322
- is_SetDisplayPos() 325
- is_SetHwnd() 433
- is_SetOptimalCameraTiming() 338
- is_UpdateDisplay() 450

# 5 Specifications

This chapter lists the specifications of the available DCx camera models.

## 5.1 Model Comparison

The following table outlines the key features of DCx camera series for direct comparison (see also the DCx Camera Family 14 chapter).

| | DCC1545M<br>DCC1645C | DCU223x<br>DCU224x | DC1240x | DC3240x |
|---|---|---|---|---|
| Sensor Type | CMOS | CCD | CMOS | CMOS |
| Objective Mount | CS (C*) | C | C | C |
| EMC compliance | CE B<br>FCC B | CE A<br>FCC A | CE B<br>FCC A | CE A<br>FCC A |
| WxHxD [mm]<br>(CCD size) | 44 x 44 x 25.6 | 32 x 34 x 30.2<br>(37.2) | 32 x 34 x 30.2 | 29 x 29 x 29 |
| Mounting holes<br>bottom<br>top<br>side | <br>1<br>0<br>0 | <br>2<br>3<br>3 | <br>2<br>3<br>3 | <br>2<br>3<br>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 509 | 30 | 30 | 30 | 30 |
| Interface | USB 2.0 | USB 2.0 | USB 2.0 | USB 3.0<br>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 |
| Max. pixel clock (MHz) at full resolution | 43 | 43 | 43 | 480 |
| Max. data rate (MBytes/s) | 38 | 38 | 38 | 300 |

*) with supplied adapter

## 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<br>15/16<br>22<br>32 | CMOS Global Shutter<br>CMOS Rolling Shutter<br>CCD Progressive Scan<br>CMOS Global Shutter, USB3 | | | | | | |
| **Resolution** | | | | | | | |
| 3<br>4, 40, 45 | XGA<br>SXGA | 1024 x 768 (0.78 MPixel)<br>1280 x 1024 (1.30 MPixel) | | | | | |
| | **Color / Mono** | | | | | | |
| M<br>C<br>N | Monochrome sensor<br>Color sensor<br>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 461
- DCC1545M 464
- DCC1645C 466

**Cameras with CCD sensors**

- DCU223x 468
- DCU224x 470

**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

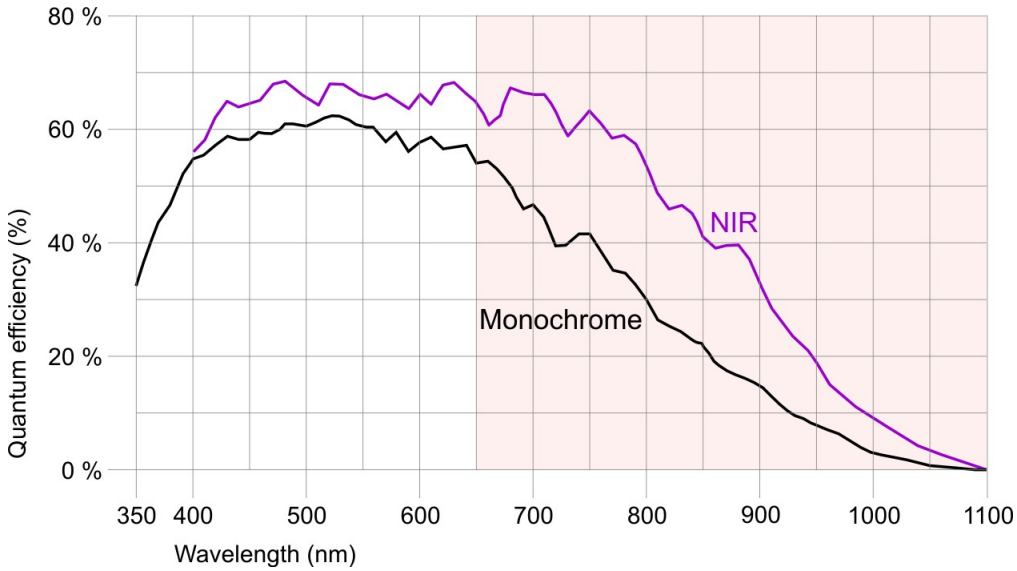
| **Sensor specification** | | | |
|---|---|---|---|
| Sensor type | CMOS | | |
| Shutter system | Electronic global and rolling shutter | | |
| Characteristic | Linear | | |
| Readout mode | Progressive scan | | |
| Resolution class | SXGA | | |
| Resolution | 1280 x 1024 pixels (1.3 Megapixel) | | |
| Aspect ratio | 5:4 | | |
| Bit depth 41 | 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 25 | 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()` 260<br>• Multi AOI with 2 or 4 AOI, see Camera basics: AOI 35<br>• Sequence AOI<br>• Sensor internal image scaler, downscaling by factor 1...4, see `is_SetSensorScaler()` 343 and uc480 Viewer: Size 97)<br>• Allows to switch between global and rolling shutter readout, see `is_DeviceFeature()` 192 | | |
| **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 | |
| 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,<br>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 |

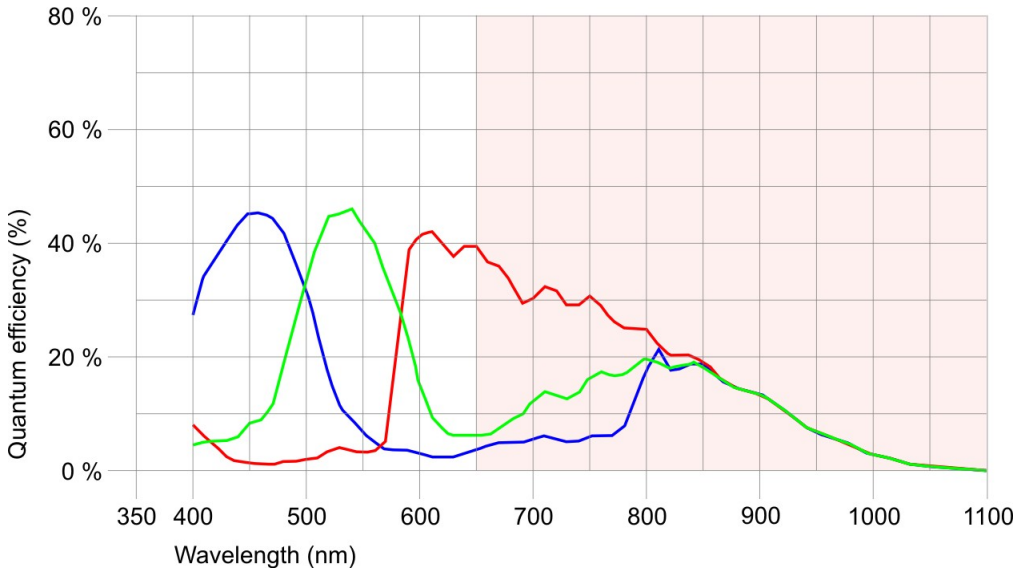
** Not yet defined.
*1 The maximum possible pixel clock frequency depends on the PC hardware used.
*2 Requires maximum pixel clock frequency.
*3 Requires minimum pixel clock frequency.
*4 Use of this function increases the frame rate.
*5 The power consumption depends on the sensor model and the pixel 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 63 chapter.

**Relative sensor sensitivity**




*Sensor sensitivity of the DCC1240M (monochrome) / DCC3240x (monochrome and NIR)*




*Sensor sensitivity of the DCC1240C / DCC3240C*

## 5.3.2 DCC1545M

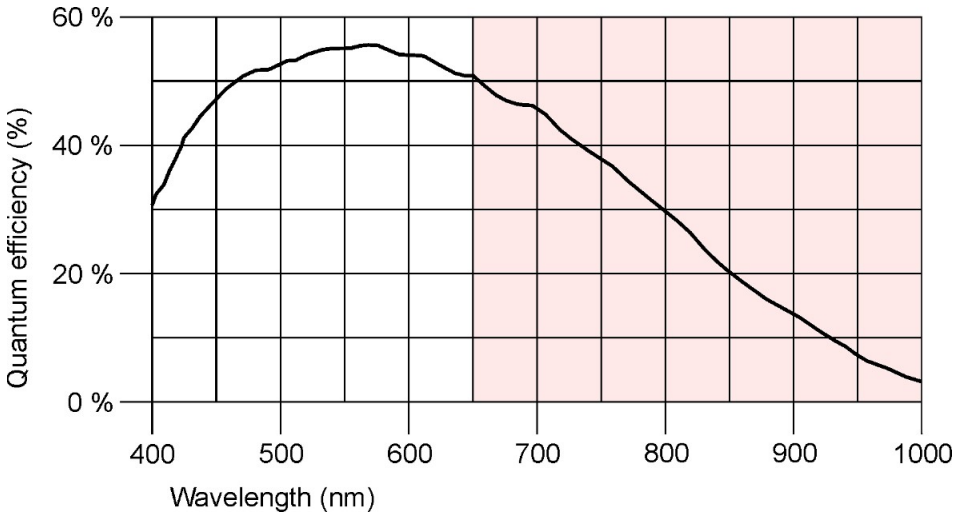
| | | |
|---|---|---|
| **Sensor specification** | | |
| Sensor type | | CMOS |
| Shutter system | | Electronic rolling shutter |
| Readout mode | | Progressive scan |
| Resolution class | | SXGA |
| Resolution | | 1280 x 1024 pixels (1.31 Megapixel) |
| Aspect ratio | | 5:4 |
| Bit depth 41 | | 10 bits |
| 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, 640 x 480 pixels | fps | 94 |
| 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 66 chapter.

**Relative sensor sensitivity**




*Sensor sensitivity of the DCC1545M (monochrome)*

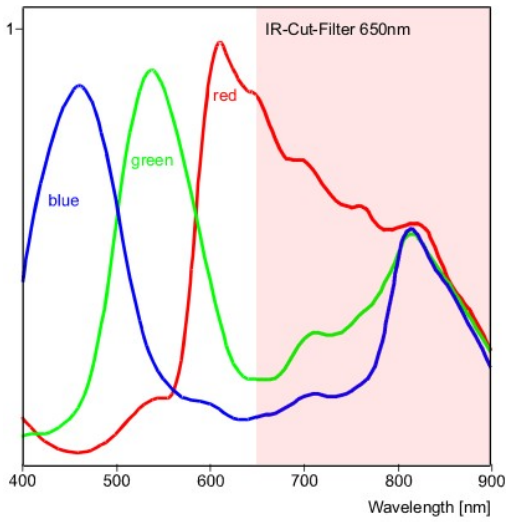
### 5.3.3 DCC1645C

| | | |
|---|---|---|
| **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 41 | | 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 | | 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 66 chapter.

**Relative sensor sensitivity**




*Sensor sensitivity of the DCC1645C*

## 5.3.4 DCU223x

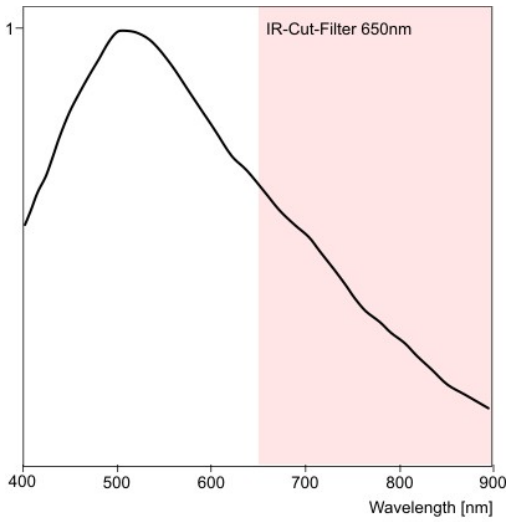
| **Sensor specification** | | |
|---|---|---|
| Sensor type | | CCD |
| Shutter system | | Electronic global shutter |
| Characteristic | | Linear |
| Readout mode | | Progressive scan |
| Resolution class | | XGA |
| Resolution | | 1024 x 768 pixels (0.79 Megapixel) |
| Aspect ratio | | 4:3 |
| Bit depth 41 | | 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 |

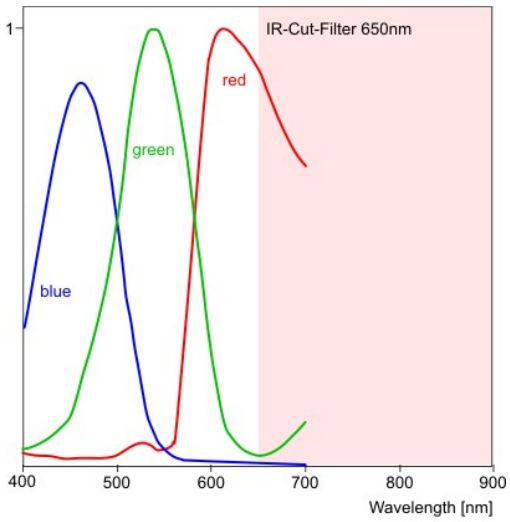
[*1] The maximum possible pixel clock frequency depends on the PC hardware used.
[*2] Requires maximum pixel clock frequency.
[*3] Requires minimum pixel clock frequency.
[*4] Use of this function increases the frame rate.
[*5] The power consumption depends on the sensor model and the pixel clock setting.

Please see also the DCU223x Application Notes 66 chapter.

**Relative sensor sensitivity**




*Sensor sensitivity of the DCU223M (monochrome)*




*Sensor sensitivity of the DCU223C (Color)*

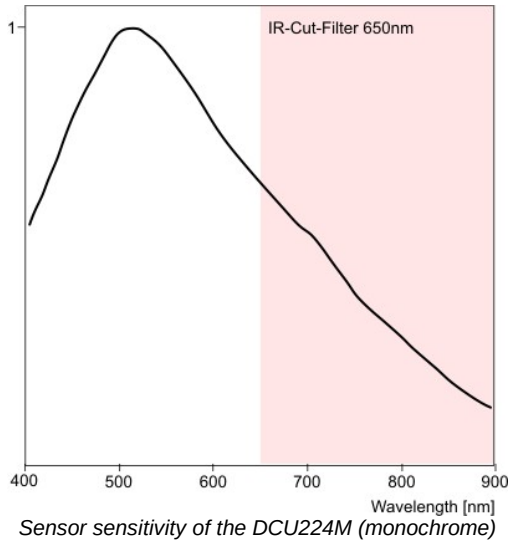
## 5.3.5 DCU224x

| | | |
|---|---|---|
| **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 41 | | 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, vertical | Pixels | 2, 2 |
| 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 |

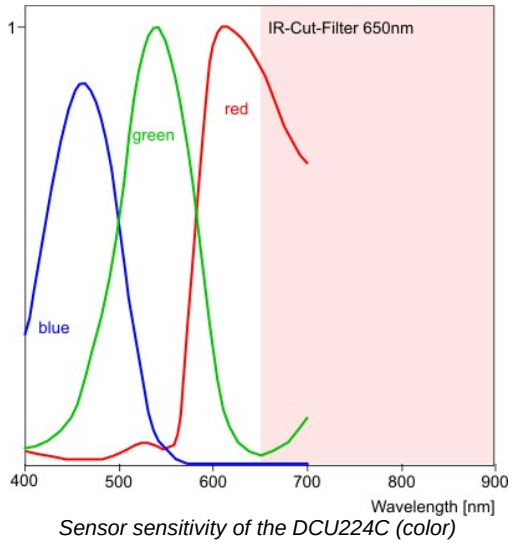
| | | |
|---|---|---|
| Factor | | 2x, 3x, 4x |
| Frame rate with 2x binning, 1280 x 512 pixels | fps | 23 |
| Frame rate with 3x binning, 1280 x 340 pixels | fps | 28 |
| Frame rate with 4x binning, 1280 x 256 pixels | fps | 31 |
| **Subsampling** | | |
| Mode | | Vertical*4 |
| Method | | V: Color 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 66 chapter.

**Relative sensor sensitivity**



*Sensor sensitivity of the DCU224M (monochrome)*



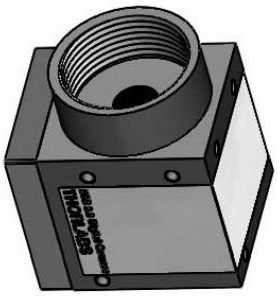
*Sensor sensitivity of the DCU224C (color)*

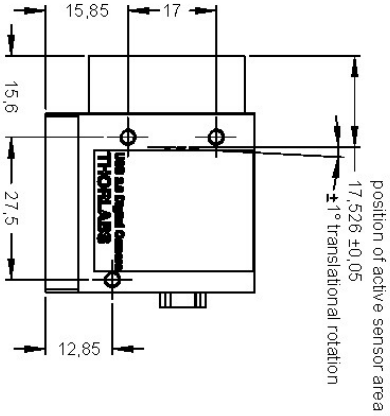
## 5.4 Mechanical Specifications

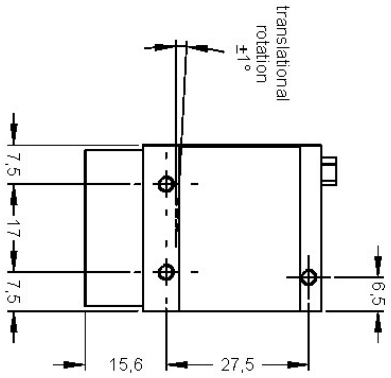


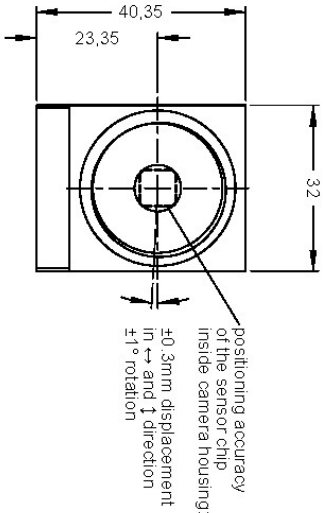
**General**

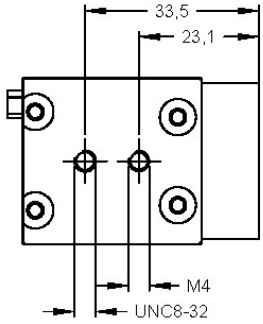
### 5.4.1 DCU223x, DCU224x



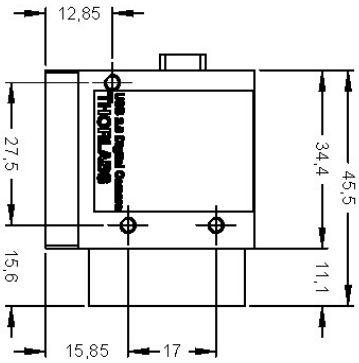


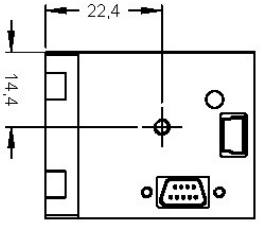



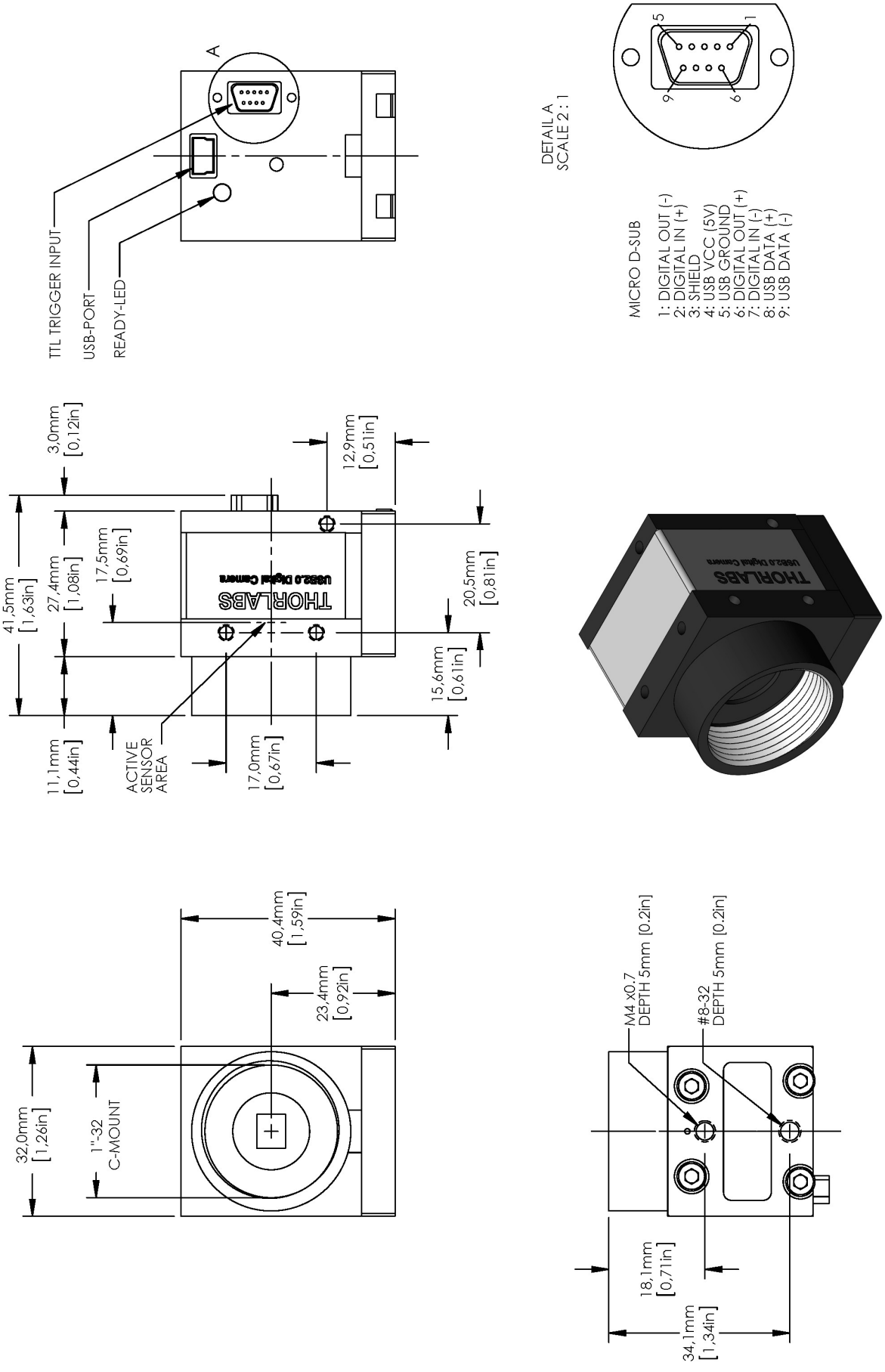




all dimensions are mm
all threads M3, unless otherwise noted

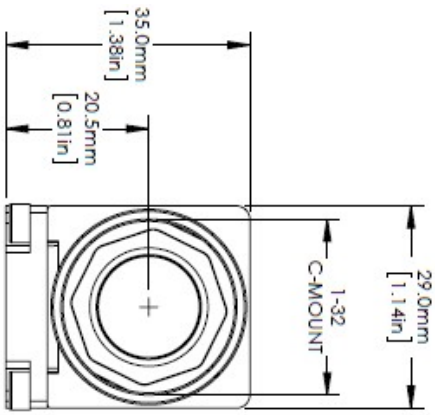


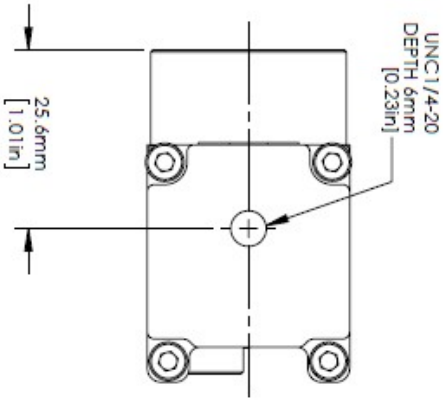



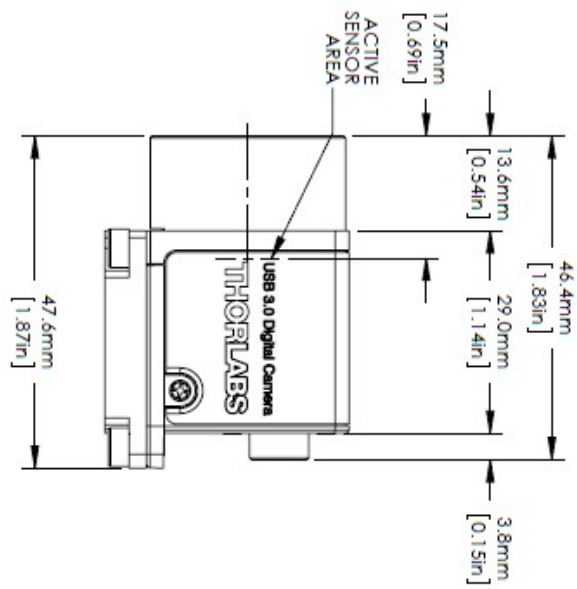
## 5.4.2 DCC1240x



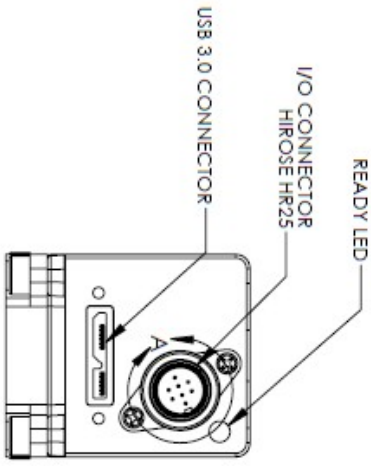
### 5.4.3 DCC3240x

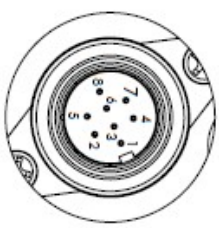


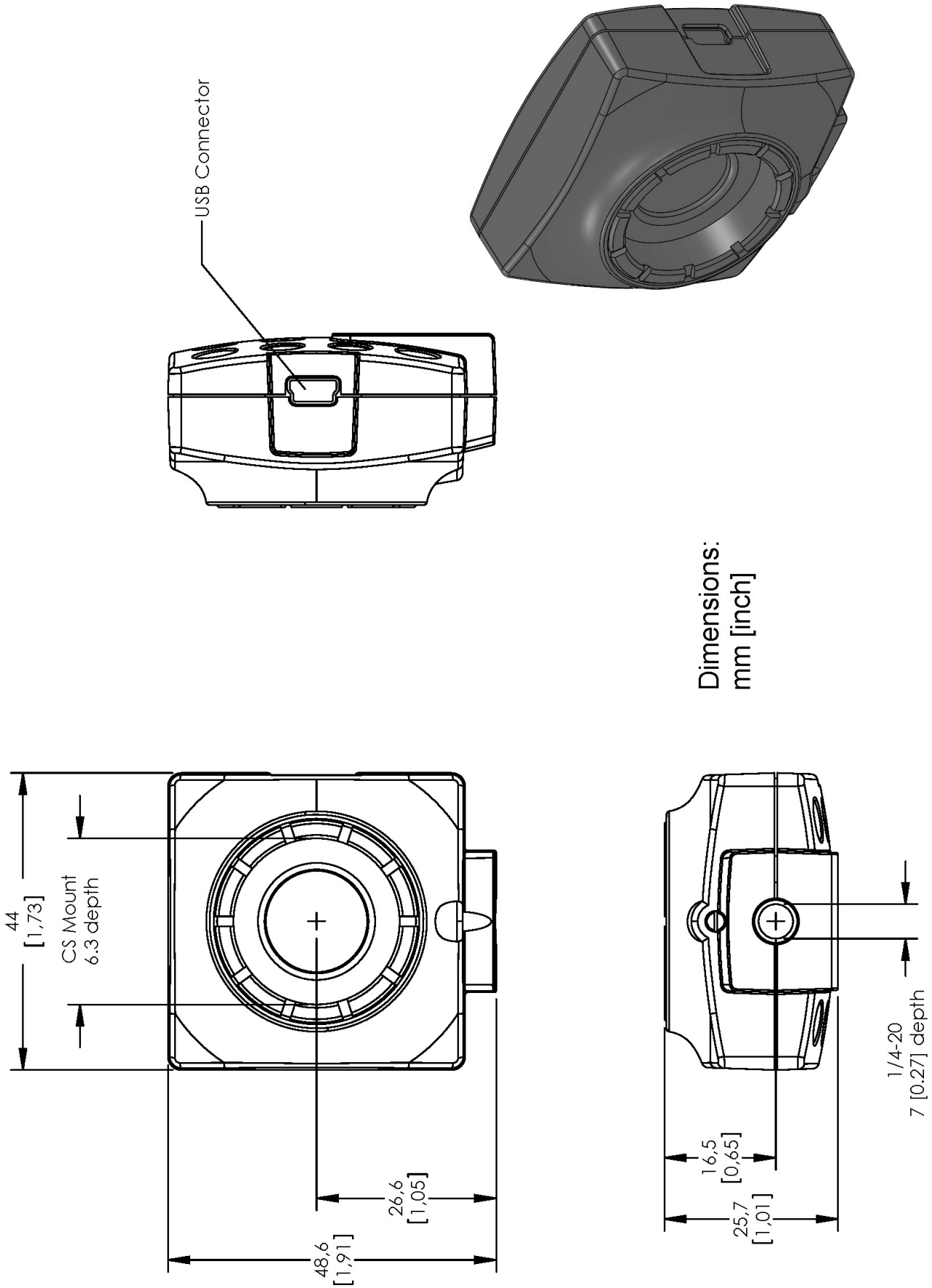




| PIN | DESCRIPTION |
|---|---|
| 1 | GROUND |
| 2 | FLASH OUTPUT, OPTO-DECOUPLED (-) |
| 3 | GPIO 1, 3.3V LVCMOS |
| 4 | TRIGGER INPUT,OPTO-DECOUPLED (-) |
| 5 | FLASH OUTPUT, OPTO-DECOUPLED (+) |
| 6 | GPIO 2, 3.3V LVCMOS |
| 7 | TRIGGER INPUT,OPTO-DECOUPLED (+) |
| 8 | OUTPUT SUPPLY VOLTAGE, 5V (100mA) |



### 5.4.4 DCC1545M, DCC1645C



## 5.4.5 Flange Back Distance

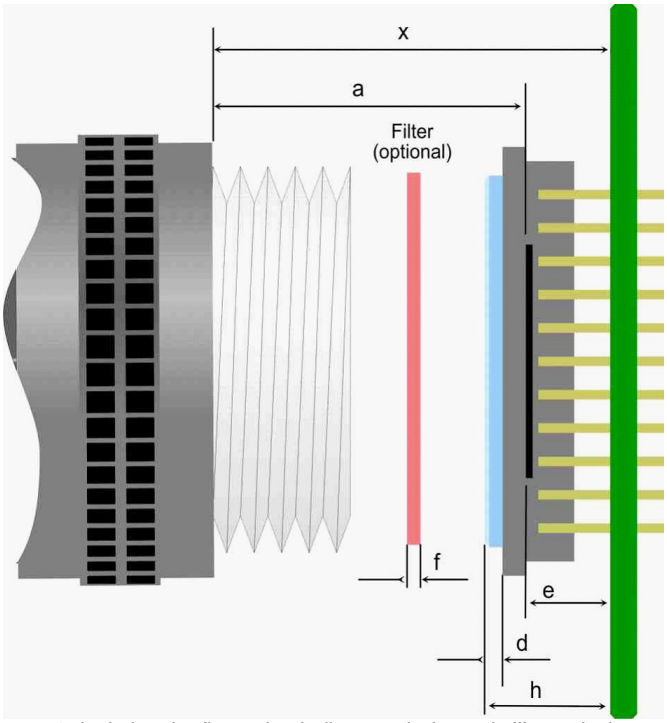


### 5.4.5.1 Calculating the Flange Back Distance

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 480 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)<br>17.526 mm*1 for C-mount<br>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)

**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 480 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 461 | 0.550 | 1.130 | 18.840 | 19.190 | 2.530 |
| **CCD** | | | | | |
| DCU223x 468 | 0.750 | 2.810 | 20.590 | 20.930 | 4.780 |
| DCU224x 470 | 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 480 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 464 | 0.525 | 1.270 | 13.23 | 15.95 | 17.50 | 2.480 |
| DCC1645C 466 | 0.550 | 1.400 | 13.10 | 16.08 | 17.63 | 2.500 |

### 5.4.5.2 Maximum Immersion Depth for Lenses

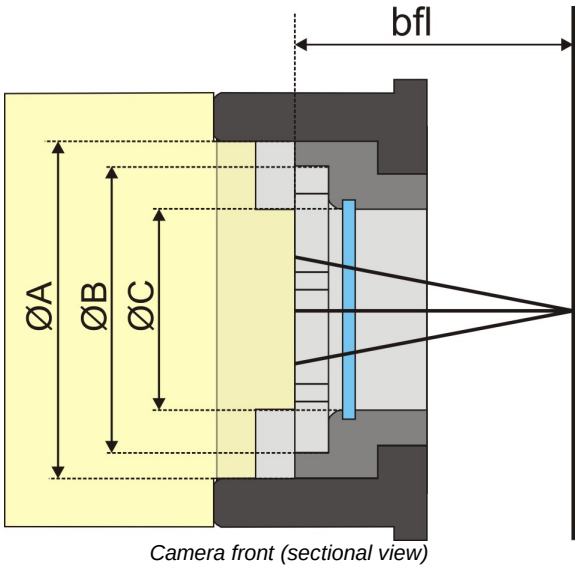
Some C-mount lenses reach deep into the camera flange. This may cause the lens to push against the back of the filter glass inside the camera or even make it impossible to screw in the lens.

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*<br>*DCU224X*<br>*DCC1240X* | 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)*<br>*DCU224X (V2)* | C-mount | 5 mm | 24.0 | - | 8.4 | - | 7.8 |
| | | | 17.1 | - | 9.2 | | |
| | | | 14.1 | - | 9.7 | | |
| *DCC1545M*<br>*DCC1645C* | 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 | - | | |
| *DCC3240x* | 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

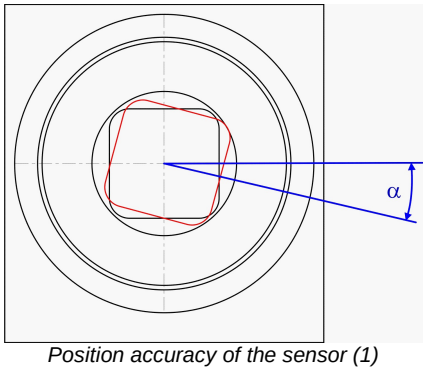
**Note**

The data given in the table include the following tolerances as a safety clearance:

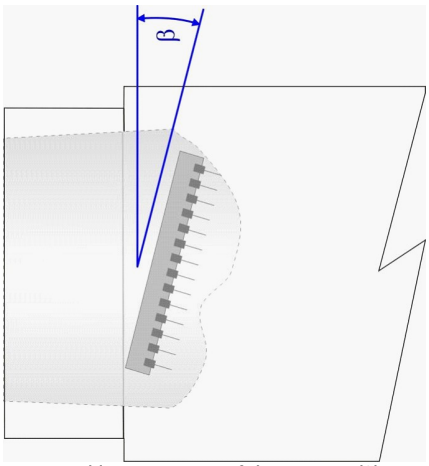
- Immersion depth: 0.2 mm
- Diameter: 0.2 mm

## 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 480
- Mounting the filter 484
- Cleaning the filter glasses 485

### 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).

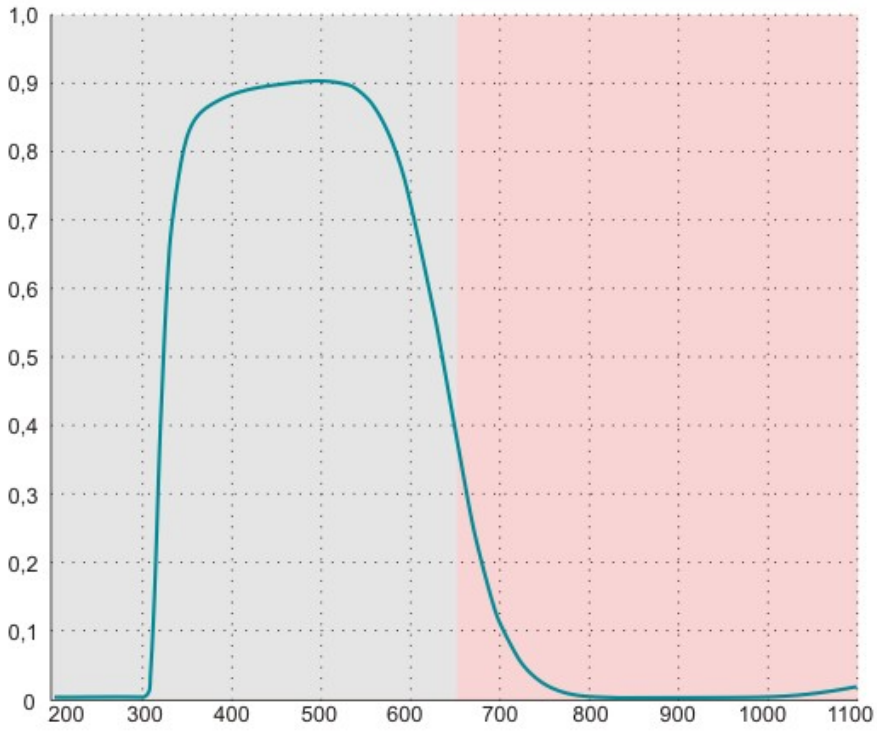
The following table shows an overview of the different optical filters used in the DCx Cameras:

| 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 |

## Notes

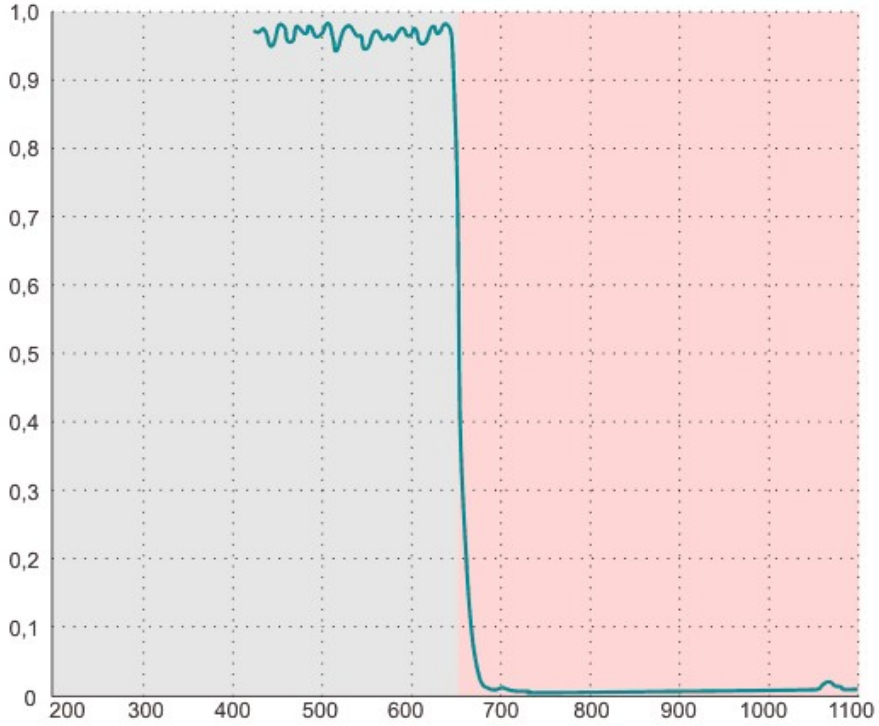
- All sensors have a D263 type cover glass. This glass is opaque to wavelengths below 330 nm.
- You can tell the filter type from visually by its coloration:
  - Reddish glass: HQ filter
  - Bluish glass: BG filter
  - Opaque glass: DL filter
  - Plain glass: GL filter
- New DCx color cameras use an 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 102 is selected automatically.

**Infrared cut filter (type BG)**



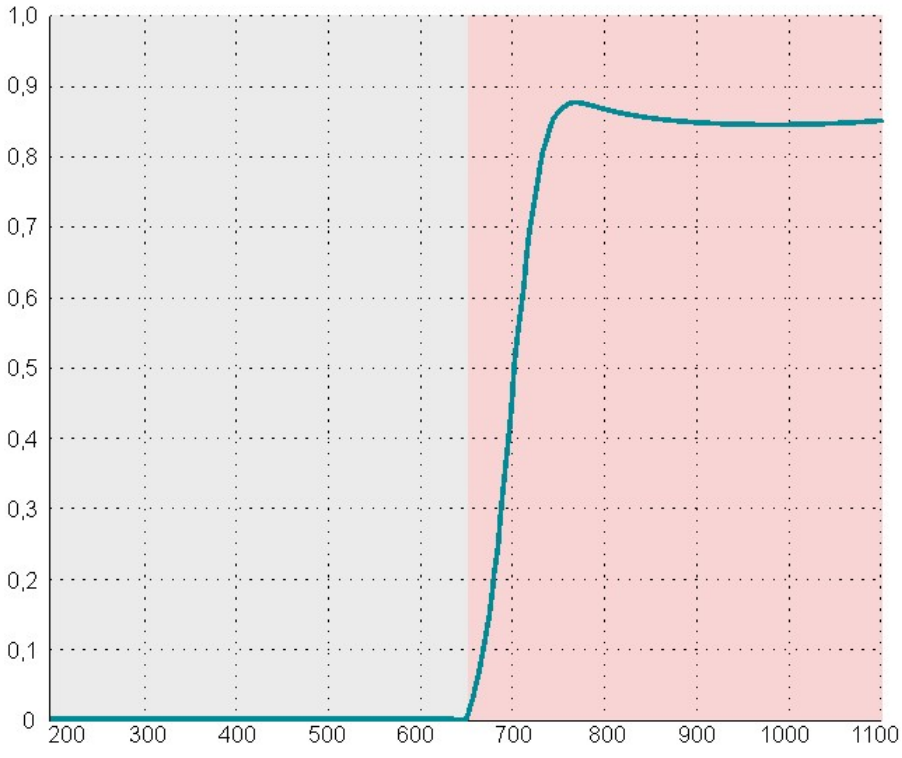

*BG filter*

**Infrared cut filter (type HQ)**



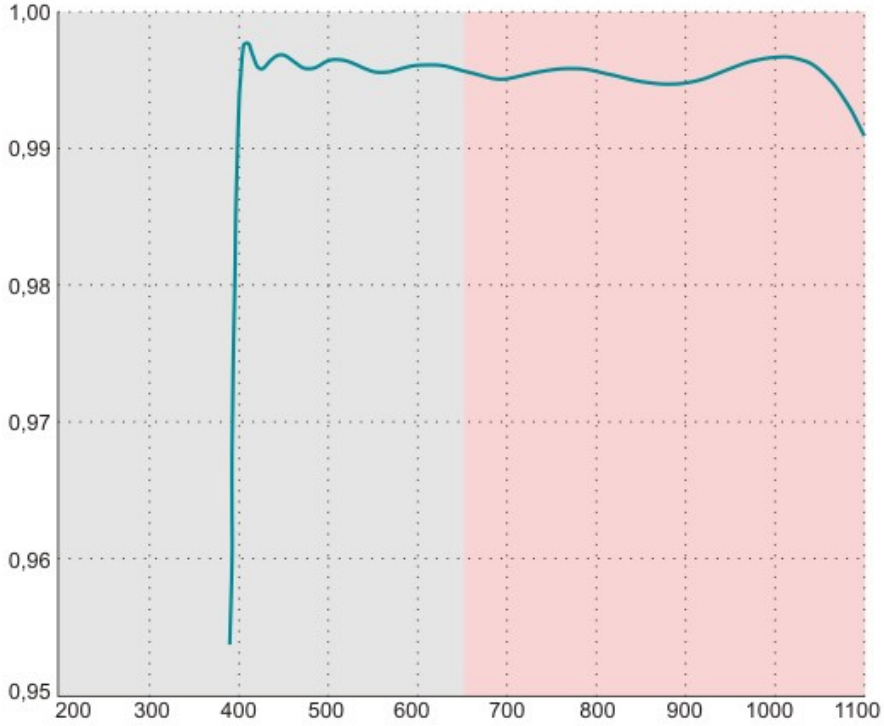

*HQ filter*

**Daylight cut filter (type DL)**




*DL cut filter*

**Plain glass filter (type GL)**




*Glass filter*

**Note**

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.

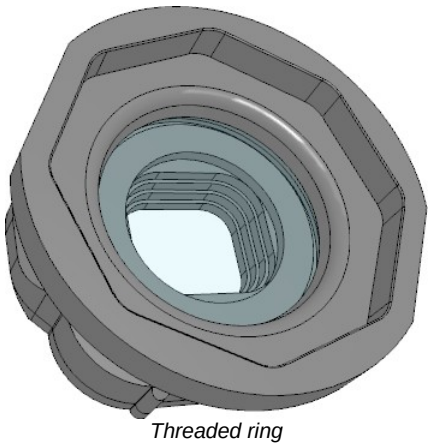
### 5.4.7.2 Mounting the Filter

**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.

### 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**

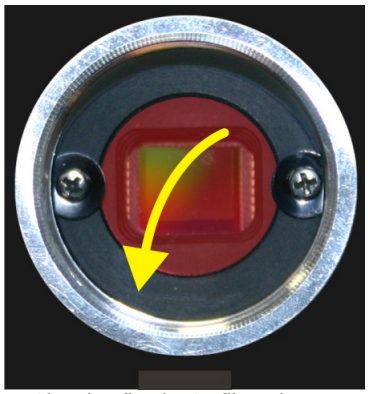
- The filter glasses can be cleaned only from outside. If remove the filter glasses, the sensor might become soiled. Thorlabs is not liable for any damage to the sensor resulting from removal of the filter glasses.
- First, remove dirt particles on the glass using compressed air. Do not use compressed 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*

## 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<br>23 | 50<br>122 | °C<br>°F |
| Storage temperature | -20<br>-4 | 60<br>140 | °C<br>°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 460 section.
This section contains information on all camera models sorted by sensor type.



## 5.5.1 DCU223x, DCU224x, DCC1240x

In this section the additional digital input / output of these cameras is described in detail:



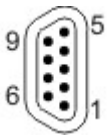
### 5.5.1.1 I/O Connector - Pin Assignment

**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 (-) |
| 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 (-) |

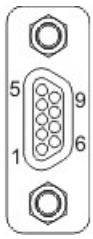



DCU22xX / DCC1240X

Micro D-Sub socket male, camera rear view

**Pin assignment of the CAB-DCU-Tx cable for USB 2.0, trigger and flash**

| Pin | Description | Cable color |
|---|---|---|
| 1 | Digital output (-) | green |
| 2 | Digital input (+) | white |
| 6 | Digital output (+) | yellow |
| 7 | Digital input (-) | brown |




*DCU22xX / DCC1240X*

*Micro D-Sub connector female, connecting side view*

For a comprehensive list of all cables and connectors available for DCU22xX / DCC1240X cameras, please refer to the DCU22xX / DCC1240X Accessories 497 section.

### 5.5.1.2 Digital Input (Trigger) Circuit

**Digital input specifications**

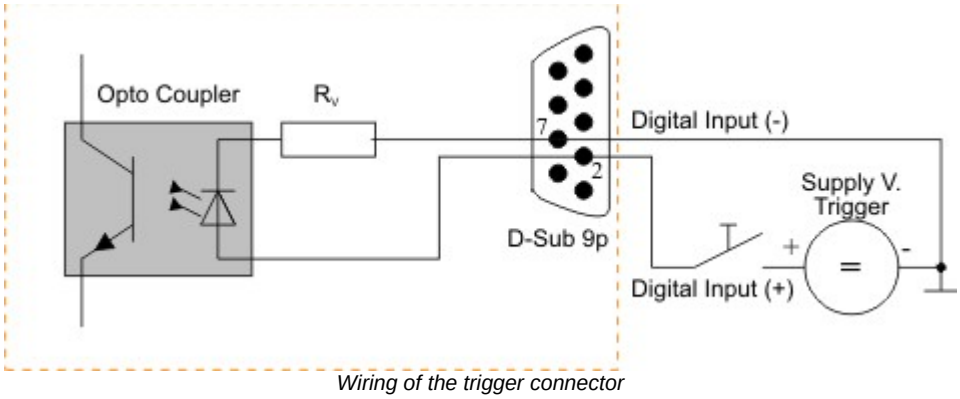
| USB board revision *) | 1.2 | | 2.0 or higher | | |
|---|---|---|---|---|---|
| | Min. | Max. | Min. | Max. | |
| Level low | 0 | 2 | 0 | 2 | V |
| Level high | 9 | 24 | 5 | 24 | V |
| Voltage range | 0 | 30 | 0 | 30 | V |
| Trigger pulse width (edge) | 100 | - | 100 | - | µs |
| Trigger edge steepness | 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 58 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*

### 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**

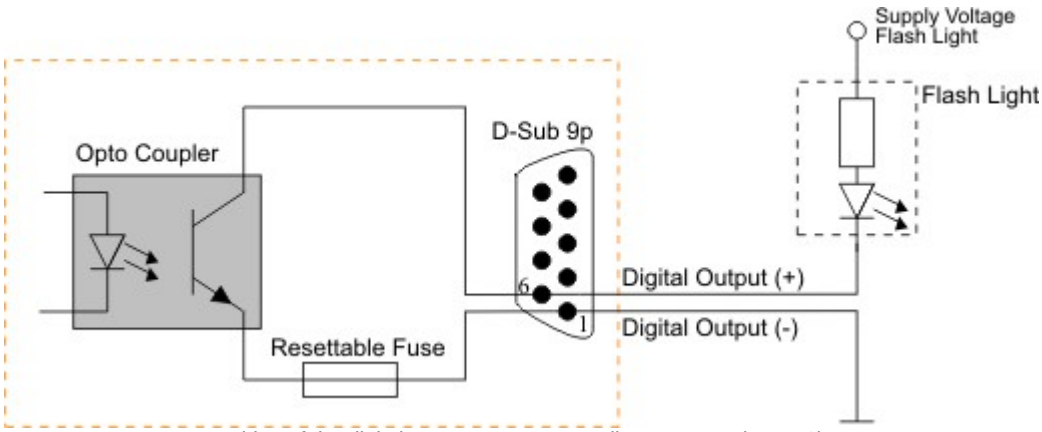
*) For information on how to determine the USB board revision, please refer to the DCx Driver Compatibility 58 chapter.

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.

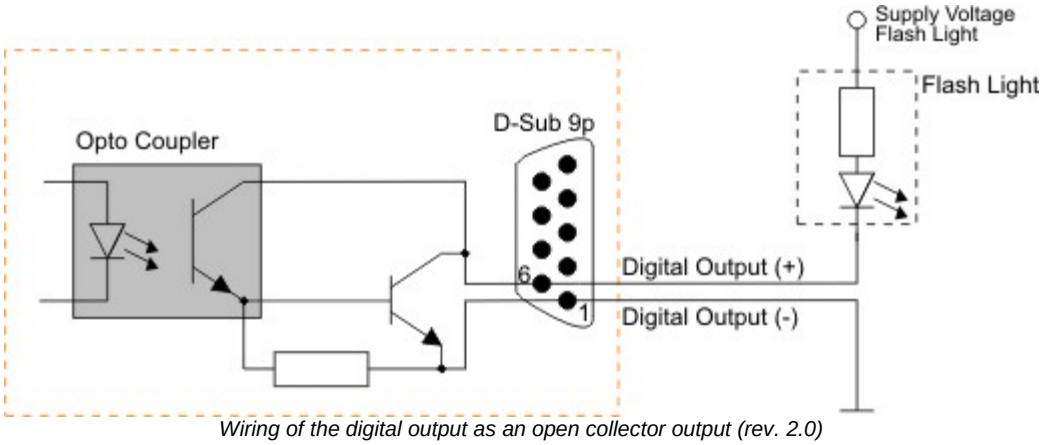
The output of the opto coupler can be used as an open collector or open emitter output. This means that the output signal can be connected to ground or to the supply voltage. The output signal is active if the collector-emitter switch is closed (software setting: Flash high active, see also the Camera Properties: Input/Output 104 section).

**Digital output wiring**

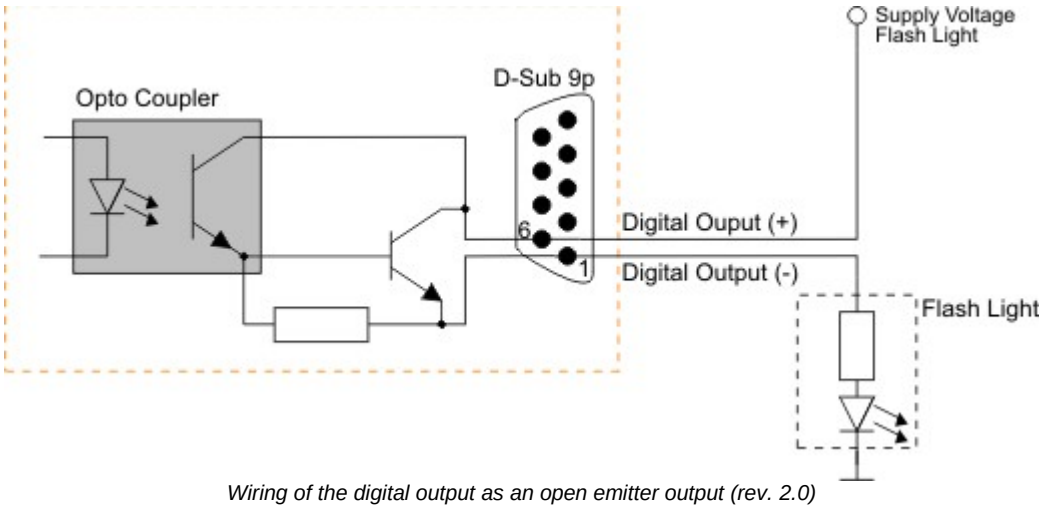
The following figures show examples of how the digital output is wired.




*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)*

## 5.5.2 DCC3240x



### 5.5.2.1 I/O Connector Pin Assignment

**Attention**

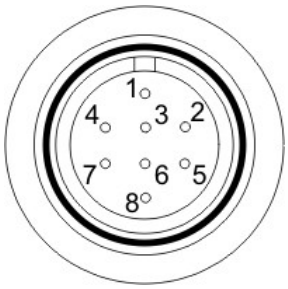
The General Purpose IO are not potential-free and have no protective circuit. Faulty wiring (overvoltage, undervoltage or inverting the wiring when used as serial interface) can result in a damage in the electronics.

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 498 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**

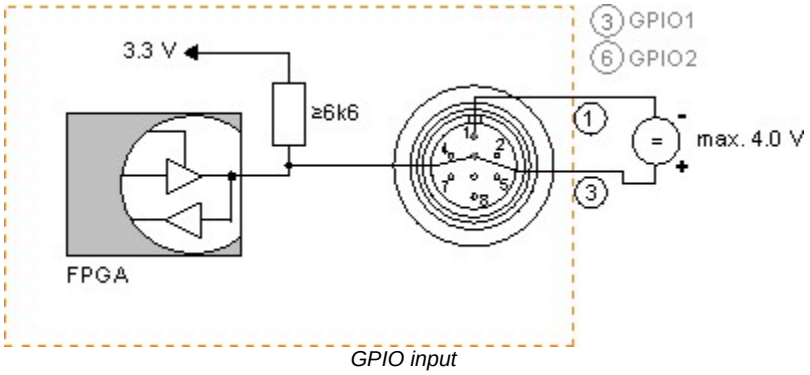
The General Purpose IO are not potential-free and have no protective circuit. Faulty wiring (overvoltage, undervoltage or inverting the wiring when used as serial interface) can result in a damage in the electronics.

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

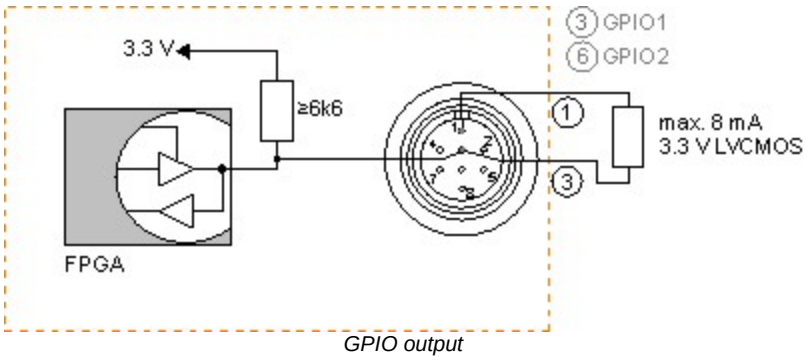
The following figures illustrate GPIO wiring examples.




*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 |
| Signal level "High" | 2.4 | 3.3 | V |
| Output current | 0 | 8.0 | mA |

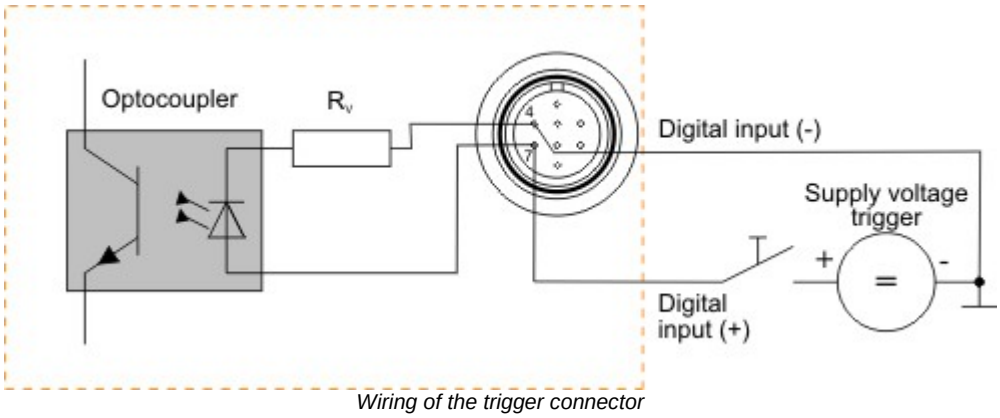
### 5.5.2.3 Digital Input (Trigger) Circuit

**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*

### 5.5.2.4 Digital Output (Flash) Circuit

**Digital output specifications**

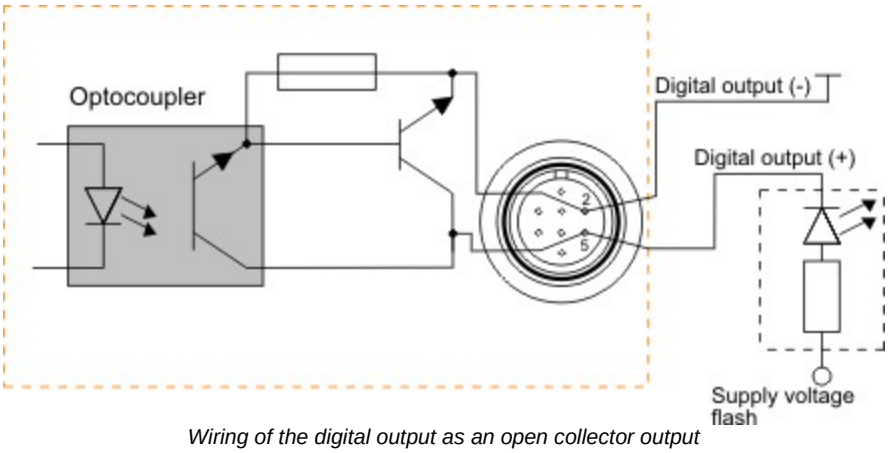
| | Max. | |
|---|---|---|
| Output current (short-time) | 500 | mA |
| Output current (permanent) | 150 | mA |
| Output voltage | 30 | V |
| Breakdown voltage | 50 | V |
| Collector power dissipation | 125 | mW |

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.

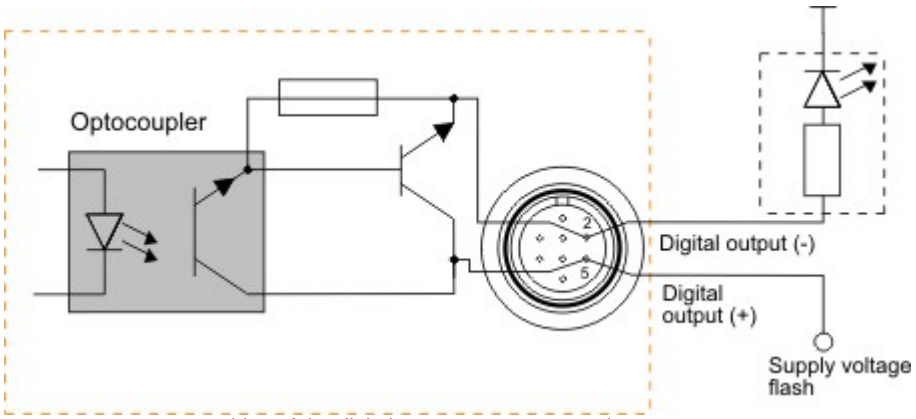
The output of the optocoupler can be used as an open collector or open emitter output. This means that the output signal can be connected to ground or to the supply voltage. The output signal is active if the collector-emitter switch is closed (software setting: Flash high active, see also the Camera properties: Input/output 104 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**

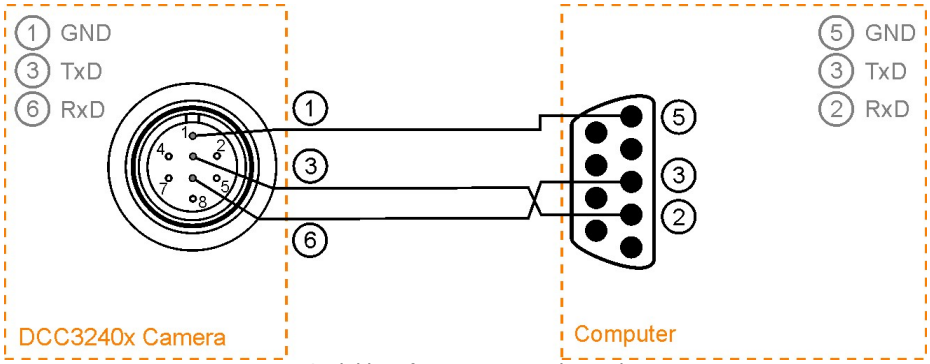
| | | | | |
|---|---|---|---|---|
| Minimum output voltage | | **Min.** | **Max.** | |
| | Signal level "Low" | 0 | 0.8 | V |
| | Signal level "High" | 2.0 | 4.0 | V |
| Maximum input voltage | | **Min.** | **Max.** | |
| | Signal level "Low" | 0 | 0.8 | V |
| | Signal level "High" | 2.0 | 4.0 | V |
| Supported Baud rates | 9.600<br>19.200<br>38.400<br>57.600<br>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.

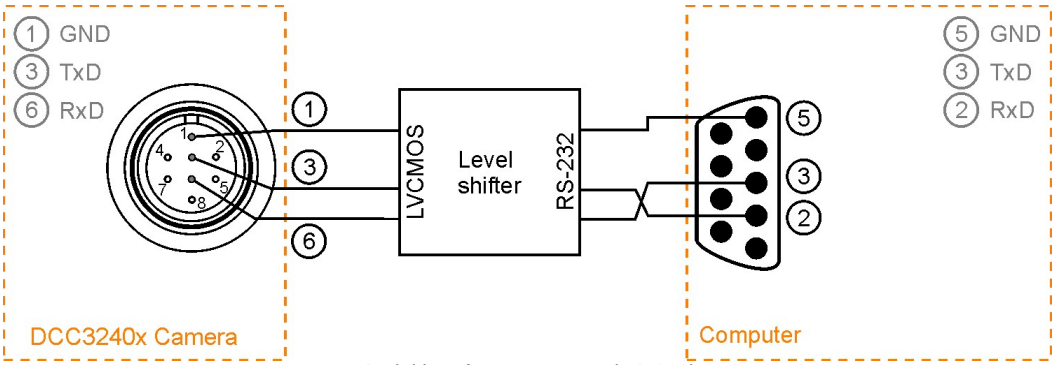
**Serial interface wiring (UART)**

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() 287).




*Serial interface connector (UART)*

**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 518 for technical support and a detailed quote tailored to your needs.

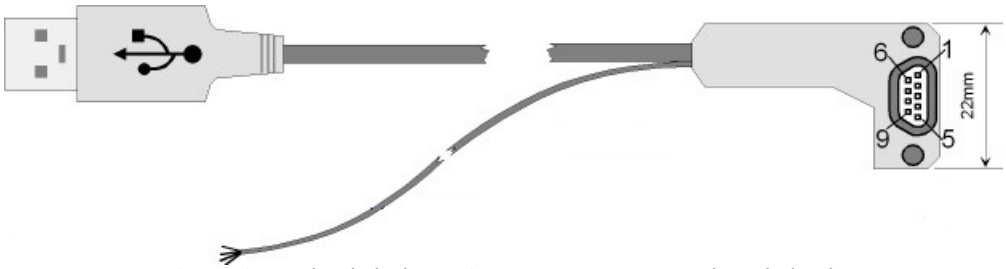
**USB cables**

All Thorlabs DCx cameras are shipped with a 1.5m USB2.0, A to Mini B, cable.

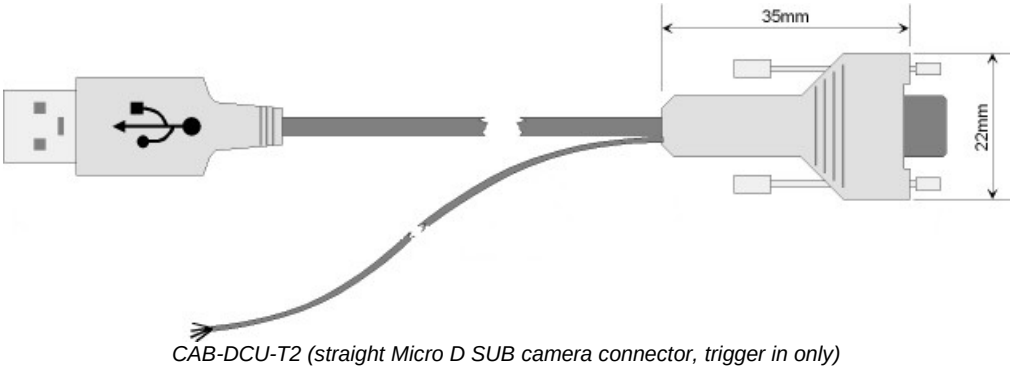
## 5.6.1 Accessories for DCU22xX / DCC1240X

For information on the pin assignment of the cables and connectors see chapter Pin Assignment I/O Connector 487.

**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)*

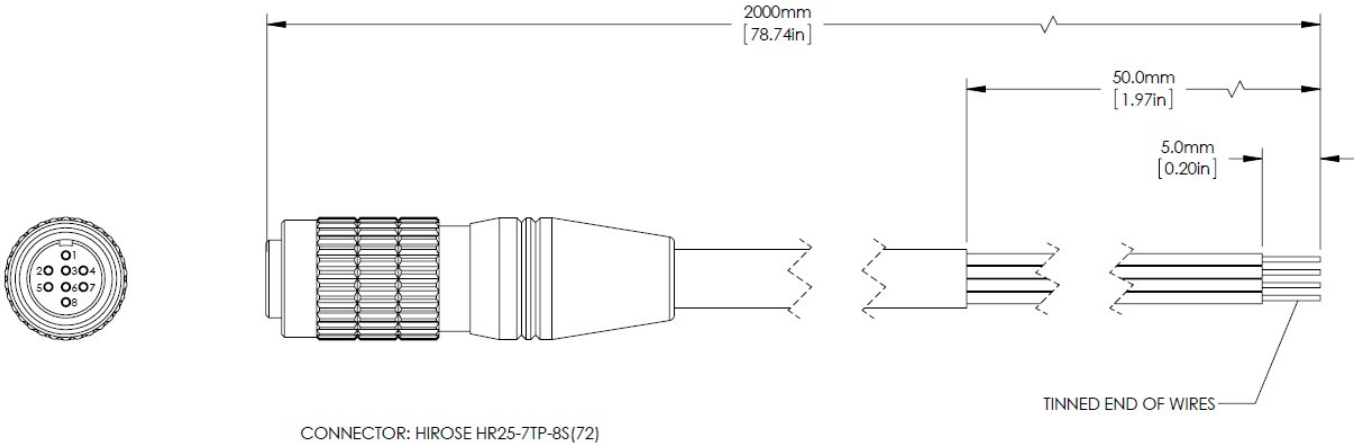
| Type | Length | Cable Type | Function | Connector Camera Side | Connector PC Side |
|---|---|---|---|---|---|
| CAB-DCU-T1 | 3 m | USB cable, AWG 28, single shielded, additional cable for digital I/Os, 4-wire, open wires | USB & Trigger in / out | Micro D-Sub for screw-mounting, angled | USB 2.0 Type A |
| CAB-DCU-T2 | 3 m | | USB & Trigger in | Micro D-Sub for screw-mounting, straight | |

## 5.6.2 Accessories for DCC1x45X

For USB cables and accessories see also Accessories for all DCx cameras 497.

DCC1545M and DCC1645C CS mount cameras are delivered with both CS / C objective mount adapter and CS / SM1 1" optics adapters.

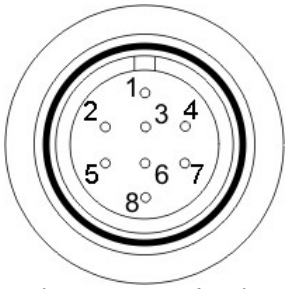
## 5.6.3 Accessories for DCC3240x




| Type | Length | Cable Type | Function | Connector Camera Side |
|---|---|---|---|---|
| CAB-DCU-T3 | 3 m | Shielded high-flexible control cable 8 x0.1mm, Ø 4.9mm | GPIO<br>digital in (trigger)<br>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



## 6.1 Troubleshooting/FAQ

**Installation and connection**

Installation of the uc480 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 uc480 Viewer.

Check the LED on the camera 501:

- LED is **red:** Camera detection failed. Check whether the uc480 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 68, 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 487 have been bent.
- LED **flashes:** A fault has occurred in the camera hardware. Please contact the Thorlabs 518.

**USB DCx camera operation**

The camera can be opened in the software, but captures images sporadically or not at all.

Check the status bar 79 in the uc480 Viewer software. If the status bar indicates transfer errors, 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:

- Pixel clock frequency: Reduce the pixel clock 44 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.

**Using the uc480 Viewer software**

I have added comments and drawings to a camera image. How can I save the image with this data?

To save a camera image with all the included text and drawings, select "Save window" from the Draw/Measure 82 menu. The menu also provides an option for saving only the drawings, so you can load them again later.

### 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 518 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 61 on the host PC?
- Does the host PC meet all system requirements 56?

*Revision 1.2 (green Status LED)*

*Revision 2.0 (red/green LED)*

**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

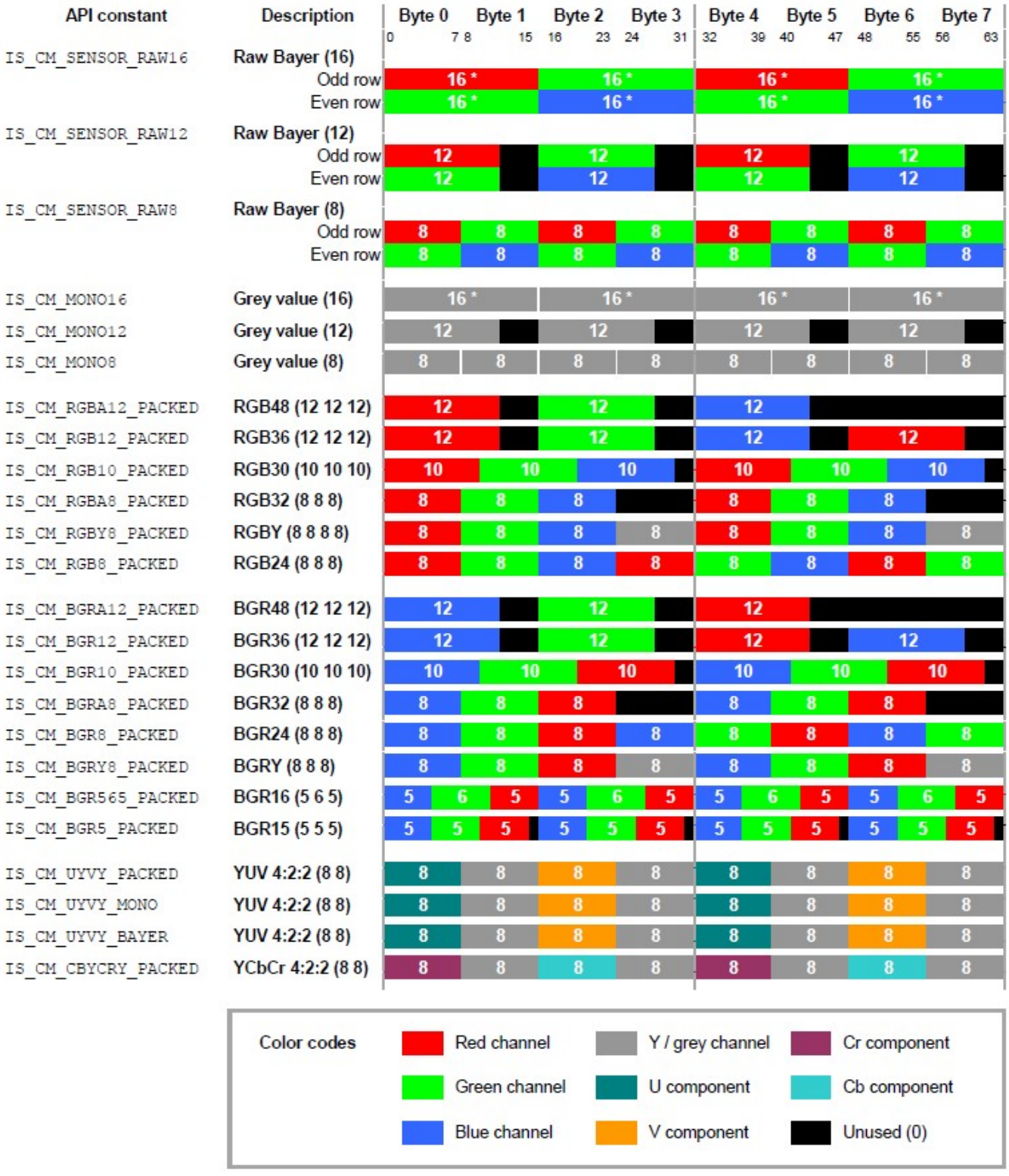
**Attention**

**Obsolete parameters**

The following parameters for color formats are obsolete and should no longer be used (see also is_SetColorMode() 321):

- 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:




*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.

## 6.4 uc480 Parameter File (ini file)

Using the is_ParameterSet() 292 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_api.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() 159) | 0, 1 | 1 |
| Start Y absolute | Activate absolute AOI positioning in the memory (see is_AOI() 159) | 0, 1 | 1 |
| Width | Width of the AOI | Sensor-dependent*1 | 2460 |
| 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<br>1 = Scaling on | 0 |
| Factor | Scaling factor | | |

| Parameter | Description | Value range | Example |
|---|---|---|---|
| **[Multi AOI]** | | | |
| Enabled | Activate/deactivate multi AOI | 0 = Multi AOI off<br>1 = Multi AOI on | 0 |
| Mode | Mode of multi AOI<br>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)<br>1 = Rolling shutter<br>2 = Global shutter<br>4 = Fast linescan<br>64 = Rolling shutter with global start<br>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<br>1 = Off (no anti-blooming)<br>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 and image geometry | 15.104458 |
| Exposure | Current exposure time | Depends on Framerate | 0.334059 |
| Long exposure | Activates long exposure<br>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() 315). For a description of all color formats, see the Color and memory formats 502 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 |

| Parameter | Description | Value range | Example |
|---|---|---|---|
| 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 502 | 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 value | 0, 1 | 0 |
| 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<br>1 = on<br>0 = off | 0 |
| GlobalShutter | Enables the Global Start shutter of the sensor | 0, 1<br>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 | double value | 1.000000 |

| Parameter | Description | Value range | Example |
|---|---|---|---|
| | balance | | |
| 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() 342) | 0…200<br>100 = Saturation 1.0<br>200 = Saturation 2.0 | 0 |
| Color_correction_satV | Saturation-V (see also is_SetSaturation() 342) | 0…200<br>100 = Saturation 1.0<br>200 = Saturation 2.0 | 0 |
| 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() 303 ) | 0 = off<br>1 = on | 0 |
| Brightness exposure Sensor control | Enable the sensor-internal brightness control | 0 = off<br>1 = on | 0 |
| Brightness gain Sensor control | Enable the sensor-internal gain control | 0 = off<br>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 | Sensor-dependent*1 | 1920 |

| Parameter | Description | Value range | Example |
|---|---|---|---|
| | brightness control | | |
| Brightness Hysteresis | Hysteresis value for auto exposure and gain (see IS_SET_AUTO_HYSTERESIS 306) | 0…10 | 2 |
| Brightness Skip Frames | Number of images that will be not analyzed for the control (see IS_SET_AUTO_SKIPFRAMES 305) | 0…1000 | 4 |
| 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 307) | 0…10 | 2 |
| Auto WB Skip Frames | Number of images that will not be analyzed for control (see IS_SET_AUTO_WB_SKIPFRAMES 307) | 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 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 |

| Parameter | Description | Value range | Example |
|---|---|---|---|
| Flash delay | Delay of the flash signal in µs | Depends on sensor setting, can be queried using is_IO() 280 | 0 |
| Flash duration | Duration of the flash signal in µs | Depends on sensor setting, can be queried using is_IO() 280 | 200 |
| PWM mode | PWM mode | see is_IO() 280 | |
| 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() 280 | |
| GPIO direction | Direction of the GPIO | see is_IO() 280 | |
| **[Sequence AOI]** | The sequence AOI is supported by DCC1240x / DCC3240x models only. | | |
| NumberUsedAOI | Number of used AOIs | see is_AOI() 159 | |
| 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() 159 | |
| **[Transfer]** | Not supported by DCx cameras | | |
| ImageDelay_us | | | |
| PacketInterval_us | | | |

*1 This information is provided in Camera and sensor data 460 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 451).

*3 Function obsolete, see chapter Obsolete functions 379.

## 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 uc480 Software Versions

For new features in the current driver version 4.20see What is New in this Version? 16.

**New in Version 4.02**

| Cameras & functions | Described in chapter |
|---|---|
| The USB 3 DCC3240x cameras are now supported under Windows | |

| Cameras & functions | Described in chapter |
|---|---|
| XP (32 bit). | |
| The `is_WriteI2C()` function has been extended with the `IS_I2C_DONT_WAIT` parameter. The default polling may be disabled, whether the byte has been written into the storage cell. | |
| The AVI functions of the `uc480_tools.dll` now supports UNICODE strings for file names. | `isavi_GetAVIFileNameW()` 365<br>`isavi_OpenAVIW()` 372 |
| The sample programs contain two new samples for .NET programming. | See separate uc480 Samples Manual |
| The `is_IO()` function has been extended so that small flash duration and flash delay can be set for flash via the GPIOs. | `is_IO()` 280 |

**New in Version 4.01**

| Cameras & functions | Described in chapter |
|---|---|
| Support of the DCC1240x/DCC3240x with NIR sensor | DCC1240x/DCC3240x Camera and sensor data 461 |
| New camera family GigE uEye LE as board-level camera. | DCx camera family 14 |
| Additional shutter modes and Log mode for DCC1240x/DCC3240x cameras | uc480 Viewer > Properties > Shutter 115 |
| The `is_DeviceFeature()` function was enhanced to control and set the shutter modes and the Log mode of the DCC1240x/ DCC3240x. | `is_DeviceFeature()` 192 |
| New function `is_Blacklevel()` for controlling the black level | `is_Blacklevel()` 170 |
| The `is_IO()` function was enhanced to control the configuration of the GPIOs. | `is_IO()` 280 |

| Information in this manual | Decsribed in chapter |
|---|---|
| With the `is_Blacklevel()` function the following uc480 function is now obsolete:<br>• `is_SetBlCompensation()` | Obsolete functions 379 |

**New in Version 4.00**

| Cameras & functions | Described in chapter |
|---|---|
| New camera family: DCC3240x | DCx camera family 14<br>Model comparison 459<br>Mechanical Specifications DCC3240x 474<br>Camera Interface DCC3240x 491 |
| Added USB 3 specifications to the chapter about the USB interface | USB interface 51 |
| New function `is_AutoParameter()` enables/disables the auto white balance. This function is similar to the `is_SetAutoParameter()` function but does not replace the function completely in version 4.20. | `is_AutoParameter()` 167 |
| The `is_HotPixel()` function supports now Unicode file names for saving and loading of hotpixel lists. | `is_HotPixel()` 260 |
| New function for the conversion of a raw Bayer image into a the desired output format. | `is_Convert()` 188 |
| New function for activating/deactivating the software edge filter. | `is_EdgeEnhancement()` 206 |
| New function for saving the camera parameter set in a file or the camera EEPROM and loading from a file or the camera EEPROM respectively. | `is_ParameterSet()` 292 |
| New function for controlling the camera pixel clock. | `is_PixelClock()` 294 |
| New function for reading camera information. | `is_DeviceInfo()` 196 |

| Cameras & functions | Described in chapter |
|---|---|
| New function for saving and loading image files. | is_ImageFile() 264 |

| Information in this manual | Decsribed in chapter |
|---|---|
| Now, for all functions all return values are listed in the function description. | |
| With the is_ Convert() function the following uc480 functions are now obsolete:<br>• is_ConvertImage()<br>• is_SetConvertParam() | Obsolete functions 379 |
| With the is_EdgeEnhancement() function, the following uc480 function is now obsolete:<br>• is_SetEdgeEnhancement() | Obsolete functions 379 |
| With the is_ParameterSet() function the following uc480 functions are now obsolete:<br>• is_SaveParameters()<br>• is_LoadParameters() | Obsolete functions 379 |
| With the is_PixelClock() function the following uc480 functions are now obsolete:<br>• is_GetPixelClockRange()<br>• is_SetPixelClock() | Obsolete functions 379 |
| With the is_DeviceInfo() function, the following uc480 function is now obsolete:<br>• is_GetEthDeviceInfo() | Obsolete functions 379 |
| With the is_ImageFile() function the following uc480 functions are now obsolete:<br>• is_LoadImage()<br>• is_LoadImageMem()<br>• is_SaveImage()<br>• is_SaveImageEx()<br>• is_SaveImageMem()<br>• is_SaveImageMemEx() | Obsolete functions 379 |

**New in Version 3.90**

| Cameras & functions | Described in chapter |
|---|---|
| Sequence AOI mode (camera models DCC1240x)<br>You can define up to 4 AOIs. These have the same size, but may have a different position. It is also possible to have different settings for each AOI for exposure time and master gain. In addition, you can define for AOI 2, 3, and 4 how often they are readout and the images are transferred. | Sequence AOI 113<br>is_AOI() 159 |
| Additional resolution profiles for CMOS and CCD cameras. | is_ImageFormat() 267 |
| New demo uc480SequenceAoi for showing the sequence AOI mode of the camera models DCC1240x. | |
| New demo C#-Cockpit with expanded settings under C#. | |
| New function for getting and setting of the camera exposure time. | is_Exposure() 216 |
| New function for information about errors when capturing images. | is_CaptureStatus() 174 |
| New function for controlling the digital in-/outputs of the cameras | is_IO() 280 |
| New header file uc480_deprecated.h with all deprecated function definitions and constants. These are no longer part of the uc480.h file. If neccessary the uc480_deprecated.h can be | |

| Cameras & functions | Described in chapter |
|---|---|
| included additionally beside the `uc480.h`. | |
| The DCx Manager is renamed to uc480 Camera Manager. This is the control center for managing all DCx cameras. | uc480 Camera Manager 67 |
| JPEG- and PNG files can be loaded using the `is_LoadImage()` function. | `is_LoadImage()` 395 |

| Information in this manual | Decsribed in chapter |
|---|---|
| The `is_IO()` function integrated all flash and IO functions. The following uc480 functions are now obsolete:<br>• `is_SetIO()`<br>• `is_SetIOMask()`<br>• `is_SetLED()`<br>• `is_GetGlobalFlashDelays()`<br>• `is_SetFlashDelay()`<br>• `is_SetFlashStrobe()` | Obsolete functions 379<br>• `is_SetIO` 440<br>• `is_SetIOMask` 441<br>• `is_SetLED` 443<br>• `is_GetGlobalFlashDelays` 391<br>• `is_SetFlashDelay` 429<br>• `is_SetFlashStrobe` 431 |
| The `is_Exposure()` function integrates the functions for camera exposure. The following uc480 functions are now obsolete:<br>• `is_GetExposureRange()`<br>• `is_SetExposureTime()` | Obsolete functions 379<br>• `is_GetExposureRange()` 390<br>• `is_SetExposureTime()` 427 |
| The `is_CaptureStatus()` function provide information about errors while capturing images. The following uc480 functions are now obsolete:<br>• `is_GetCaptureErrorInfo()`<br>• `is_ResetCaptureErrorInfo()` | Obsolete functions 379<br>• `is_GetCaptureErrorInfo()` 386<br>• `is_ResetCaptureErrorInfo()` 404 |

**New in Version 3.82**

| Cameras & functions | Described in chapter |
|---|---|
| New function to set system-wide options. | `is_Configuration()` 183 |
| New function to set and query the exposure time. | `is_Exposure()` 216 |
| The functions for AVI recording with the uc480_tools.dll are supported by 64 bit operating systems. | AVI Function Descriptions 359 |

**New in Version 3.80**

| Cameras & functions | Described in chapter |
|---|---|
| Support of the Multi AOI mode on DCC1240x camera models | Basice: Area of Interest (AOI) 34<br>uc480 Viewer: Multi AOI 112<br>`is_AOI()` 159<br>DCC1240x Specifications 461 |
| Support of the line scan mode on DCC1240x camera models | Basics: Zeilenmodus 33<br>uc480 Viewer: Shutter 115<br>`is_DeviceFeature()` 192<br>DCC1240x Specifications 461 |
| New function for setting the size and position of an area of interest (AOI) | `is_AOI()` 159 |
| New function for reading and extending the camera's internal hot pixel correction list | `is_HotPixel()` 260 |
| New function for setting special options on some camera models. Currently the following special options are available through this function:<br>• On UI-124x/UI524x models: Set line scan mode<br>• On UI-124x/UI524x models: Toggle between rolling and global shutter mode | `is_DeviceFeature()` 192 |

| Cameras & functions | Described in chapter |
|---|---|
| • On UI-1008XS (uEye XS) model: Choose the HS mode for triggered image capture | |
| Predefined image formats can now be set for all CMOS cameras | `is_ImageFormat()` 267 |
| New uc480 Hotpixel Editor for editing the hotpixel list stored in the camera | uc480 Hotpixel Editor 123 |
| The I2C outputs on some uEye models can now be programmed directly in the uc480 Viewer. | uc480 Viewer: Miscellaneous 110 |
| The uc480 Viewer can now display and save images with a bit depth of more than 8 bit. | uc480 Viewer: Format 99 |

| Information in this manual | Described in chapter |
|---|---|
| The application notes for the individual camera models are now provided in a separate chapter. | Application notes for the cameras 63 |
| The `is_AOI()` function comprises all the functions for setting and positioning an AOI. The following uc480 API commands are therefore obsolete and have been moved to the Obsolete Functions chapter:<br>• `is_SetAOI()`<br>• `is_SetImageAOI()`<br>• `is_SetImageSize()`<br>• `is_SetImagePos()`<br>is_AOI() also allows quickly moving an AOI to a different position. | Obsolete Functions 379<br>`is_AOI()` 159 |

**New in Version 3.70**

| Cameras & functions | Described in chapter |
|---|---|
| New API function for setting the **color space** and **color temperature** | `is_ColorTemperature()` 180 |
| New function parameters for reading out the **exposure time range** | `is_SetExposureTime()` 427 |

| Information in this manual | Described in chapter |
|---|---|
| Updated camera specifications | DCC1240x 461 |
| Schematic timing diagrams for using the flash in hardware trigger mode | Digital In-/Output (Trigger/Flash) 47 |

**New in Version 3.60**

| Functions | Described in chapter |
|---|---|
| New function in uc480 Viewer for selection profiles for camera settings | Start Dialog 76<br>Creating Profiles 117 |

| Information in this manual | Described in chapter |
|---|---|
| Function `is_GetCameraType()` moved to the "Obsolete Functions" section. It is entirely replaced by `is_GetCameraInfo()`. | Obsolete Functions 379<br>`is_GetCameraInfo()` 231 |

**New in Version 3.51**

| Functions | Described in chapter |
|---|---|
| Driver support for Windows 7 | System Requirements 12 |
| Support for 64 bit versions of Windows 7, Windows Vista and Linux | System Requirements 12 |

| Software | Described in chapter |
|---|---|
| Support of Direct3D graphics functions with Overlay<br>The DirectDraw functions have been completely replaced by the new API function. This function allows image scaling and inserting overlay data into the camera's live image without flicker. | Camera properties: Format 99<br>System Requirements 12 |
| Extended Automatic Image Control:<br>• Configurable hysteresis control | Automatic Image Control 46<br>AES/AGC (Automatic Brightness Control) 106 |

| Information in this Manual | Described in chapter |
|---|---|
| Basics of digitizing and bit depth of image data | Basics: Digitizing Images 39 |
| Function `is_GetCameraType()` moved to the "Obsolete Functions" section. It is entirely replaced by `is_GetCameraInfo()`. | Obsolete Functions 379<br>`is_GetCameraInfo()` 231 |
| Merge of the previously separated manuals DCx *Programming Manual* and DCx *User Manual.* | - |
| New section: How to Proceed - DCx Programming | How To Proceed 129 |
| New chapter: Troubleshooting | Troubleshooting 499 |
| Keyword index added | - |
| New chapter: *Quick-start*<br>This chapter explains in a nutshell how to configure your DCx camera and capture images. | Quick-start 54 |
| *New chapter*: Firmware and Camera Start-up | Firmware and Camera Start-up 46 |
| *New chapter*: All DCx models at a glance<br>A table shows the most important features of each DCx series at a glance. | Model comparison 459 |
| Exact measuring results for the trigger delay of all DCx models. | Camera and Sensor Data 460 |

**New in Version 3.32**

| New features | Described in chapter |
|---|---|
| Extended trigger mode<br>The continuous trigger mode allows triggering the DCx repeatedly. The camera no longer has to be made ready for the next trigger before each image capture. | Operating Modes: Trigger Mode 19 |
| Improved DCx *Camera Manager* features | uc480 Camera Manager 67 |
| Test image function<br>The camera transmits a selectable test image that you can use for testing the data transmission. | Camera properties: Test image 110 |
| Support of 10 and 12 bit sensor data<br>Some sensors can output images with a color depth of 10 to 12 bits. This data can now be processed by the DCx software. | Camera and Sensor Data 460 |
| New color formats<br>*uc480* driver version 3.3 supports a wide range of new color | Camera properties: Color 102 |

| | |
|---|---|
| formats for all DCx cameras. These include:<br>o RGB/BGR 30<br>o RGBY<br>o Y12<br>o YCbCr<br>o Enhanced YUV | |
| Full support of *Windows Vista* (32 Bit)<br>From driver version 3.30 onwards, all DCx cameras will run under Windows Vista 32. | System Requirements 12 |
| Localization of the *uc480 Camera Manager*<br>The uc480 Camera Manager offers new features and now also supports over 10 languages that can be switched anytime. | uc480 Camera Manager 67 |

| **New information in the manual** | **Described in chapter** |
|---|---|
| Detailed presentation of all DCx operating modes | Operating Modes 17 |
| Updated connected load data on every camera model | Camera and Sensor Data 460 |

## 6.7 History of uc480 API Functions

**New functions in software version 4.20**

- is_Measure() 290

**New functions in software version 4.01**

- is_Blacklevel() 170

**New functions in software version 4.00**

- is_AutoParameter() 167
- is_Convert() 188
- is_DeviceInfo() 196
- is_EdgeEnhancement() 206
- is_ImageFile() 264
- is_ParameterSet() 292
- is_PixelClock() 294

**New functions in software version 3.90**

- is_CaptureStatus() 174
- is_Exposure() 216
- is_IO() 280

**New functions in software version 3.81**

- is_Configuration() 183

**New functions in software version 3.80**

- is_AOI() 159
- is_DeviceFeature() 192
- is_HotPixel() 260

**New functions in software version 3.70**

- is_ColorTemperature() 180

**New functions in software version 3.52/3.60**

- is_ImageFormat() 267

**New functions in software version 3.40**

- is_DirectRenderer() 198
- is_GetImageInfo() 244
- is_GetSensorScalerInfo() 252
- is_SetSensorScaler() 343

**New functions in software version 3.33**

- is_Direct3D() 198
- is_GetTimeout() 256

**New functions in software version 3.32**

- is_GetTimeout() 256
- is_SetTimeout() 350

- is_SetTriggerCounter() 351

**New functions in software version 3.30**

- is_GetCameraLUT() 235
- is_GetCaptureErrorInfo() 386
- is_GetColorConverter() 236
- is_GetSupportedTestImages() 253
- is_GetTestImageValueRange() 255
- is_ResetCaptureErrorInfo() 404
- is_SetColorConverter() 315
- is_SetSensorTestImage() 345

**New function in software version 3.20**

- is_SetOptimalCameraTiming() 338

## 6.8 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

**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

**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

**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

**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

**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

**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

**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

## 6.9 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<br>CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12 |
| DCU223x [*1]<br>DCU224x [*1] | EN 61000-6-2:2001 | EN 61000-6-4:2001 | UL 60950-1, 2nd Edition, 2011-12-19<br>CSA C22.2 No. 60950-1-07, 2nd Edition, 2011-12 |
| DCC1545M [*1]<br>DCC1645C [*1] | EN 61000-6-2:2005 | EN 61000-6-3:2001 + A11:2004 | UL 60950-1, 2nd Edition, 2011-12-19<br>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<br>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 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.

## 6.10 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 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 electrical and electronic equipment

- sold after August 13th 2005
- marked correspondingly with the crossed out “wheelie bin” logo (see Figure 58)
- 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 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, 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*

## 6.11 Warranty

*Thorlabs* warrants material and production of the DCx Cameras for a period of 24 months starting with the date of shipment. During this warranty period *Thorlabs* will see to defaults by repair or by exchange if these are entitled to warranty.

For warranty repairs or service the unit must be sent back to *Thorlabs*. The customer will carry the shipping costs to *Thorlabs,* in case of warranty repairs *Thorlabs* will carry the shipping costs back to the customer.

If no warranty repair is applicable the customer also has to carry the costs for back shipment.

In case of shipment from outside EU duties, taxes etc. which should arise have to be carried by the customer.

*Thorlabs* warrants the hard- and software determined by *Thorlabs* for this unit to operate fault-free provided that they are handled according to our requirements. However, *Thorlabs* does not warrant a fault free and uninterrupted operation of the unit, of the software or firmware for special applications nor this instruction manual to be error free. *Thorlabs* is not liable for consequential damages.

**Restriction of warranty**

The warranty mentioned before does not cover errors and defects being the result of improper treatment, software or interface not supplied by us, modification, misuse or operation outside the defined ambient stated by us or unauthorized maintenance.

Further claims will not be consented to and will not be acknowledged. *Thorlabs* does explicitly not warrant the usability or the economical use for certain cases of application.

*Thorlabs* reserves the right to change this instruction manual or the technical data of the described unit at any time.

## 6.12 Exclusion of Liability and Copyright

*Thorlabs GmbH* 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.

Under no circumstances can we guarantee that a particular objective can be achieved with the purchase of this product.

Insofar as permitted under statutory regulations, we assume no liability for direct damage, indirect damage or damages suffered by third parties resulting from the purchase of this product. In no event shall any liability exceed the purchase price of the product.

Please note that the content of this User Manual is neither part of any previous or existing agreement, promise, representation or legal relationship, nor an alteration or amendment thereof. All obligations of *Thorlabs GmbH* result from the respective contract of sale, which also includes the complete and exclusively applicable warranty regulations. These contractual warranty regulations are neither extended nor limited by the information contained in this User Manual. Should you require further information on this product, or encounter specific problems that are not discussed in sufficient detail in the User Manual, please contact your local *Thorlabs* dealer or system installer.



Status: 2013

# Index

## A



## B



## C

## H



## I

## T



## U



## V



## W