greateyes

DISCOVER WHAT
THE EYE CAN'T SEE

Instruction Manual

Software **D**evelopment **K**it
for **C++**

Version 12 Rev. 5

greateyes GmbH | Justus-von-Liebig-Straße 2 | 12489 Berlin | Germany
www.greateyes.de | info@greateyes.de | office: +49 30 912075-250

Table of Contents

1. Version Information

Version of the Camera DLL: 12.3 Date: 2020-07-27
Version of Documentation: 12.3

2. System Requirements

The PC either must provide a

- Linux 32 bit or 64 bit operating system with **libc 2.5** or newer and support for an **USB 2.0** interface (via **libusb 1.0** or newer) or an **Ethernet interface**

- Microsoft Windows 32bit or 64bit (Windows 7, 8 or 10) with support for an **USB 2.0 or Ethernet** interface incl. **Visual C++ Redistributable Packages for Visual Studio 2013**

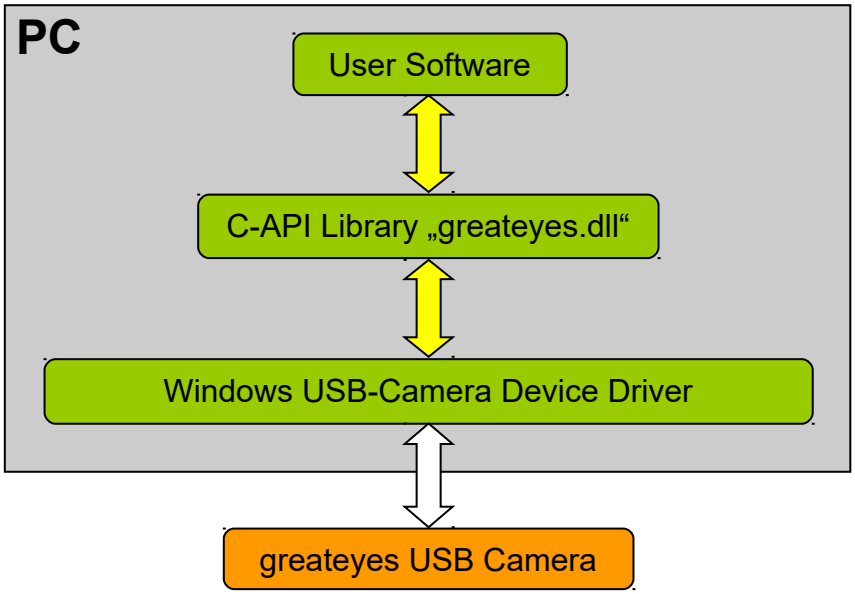
3. Supported camera types and programming features

Supported camera models	**GE 1024 1024 xxx xxx** GE 1024 1024 BI UV all Types GE 1024 1024 BI MID GE 1024 1024 BI BR GE 1024 1024 DD NIR GE 1024 1024 FI **GE 1024 256 xxx xxx** GE 1024 256 FI all Types GE 1024 256 BI all Types GE 1024 256 DD NIR **GE 2048 512 xxx xxx** GE 2048 512 BI all Types GE 2048 512 FI all Types **GE 2048 2048 xxx xxx** GE 2048 2048 FI all Types GE 2048 2048 BI all Types **GE 4096 4096 xxx xxx** GE 4096 4096 BI all Types **ALEXx Series** **ELSEx Series**
Readout Speed (Pixel Readout Frequency)	50kHz, 100kHz, 250kHz, 500kHz, 1MHz, 3MHz
Exposure Time	1.. 2,000,000 ms
ADC Dynamic Range	16 bit / 18 bit (camera specific)
Oversampling	19 bit / 20 bit via oversampling and decimation at lowest pixel readout frequencies, i.e. 50 kHz, 250kHz, optional
Binning X (Hardware based)	Supported
Binning Y (Hardware based)	Supported
TTL Trigger Outputs	Supported
Trigger Modes (Internal or external trigger IN)	Supported
Temperature Control	Supported
Temperature Readout	Supported
Shutter Control (Automatic, manual timings)	Supported
Offset Adjust	Not documented yet
Gain Adjust	Supported
Crop Mode	Supported
High Capacity Mode	Supported
Burst Mode	Supported
Video Mode (continuous single image taking)	Supported (beta status)

4. Camera data flow

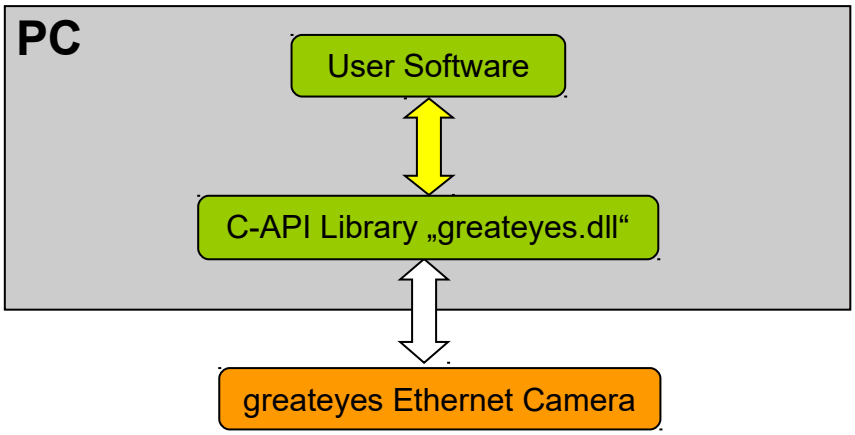
The User Software calls functions provided within the C-API Library ("greateyes.dll") to control all relevant parameters and the readout of the image data from the camera.

The scheme below shows the generic data flow of the "User Software" to and from the camera using an USB 2.0 interface.



Please note: If a camera with an USB interface is to be used the installation of the appropriate USB device driver is an important prerequisite to operate the camera – please follow the steps in the "Instruction_Manual_Driver_Installation_WinX.pdf" accurately.

The scheme below shows the generic data flow of the "User Software" to and from the camera using an Ethernet interface.



5. Camera Access Functions

The C-API library "greateyes.dll" exports several functions for camera control and image data readout that are described in detail within the following subsections.

Most functions of the C-API Library "greateyes.dll" are of boolean type, hence they return a true / false status that reflects the (un)successful execution of the command.

Most functions also return an integer type error code "**statusMSG**" providing more detailed status information. Please see section Possible values of parameter statusMSG for a list of possible status codes and a description of their meaning.

5.1. Connecting & disconnecting a greateyes camera

There are three ways to connect to a greateyes camera:

Local via USB:

Suitable for all greateyes cameras with USB interface. You can connect to up to four greateyes cameras with USB interface.

"SetupCameraInterface()"

→ "GetNumberOfConnectedCams()"

→ "ConnectCamera()"

Directly via Ethernet:

Suitable for all greateyes cameras with Ethernet interface. You can connect to up to four greateyes cameras with Ethernet interface.

"SetupCameraInterface()"

→ "**ConnectToSingleCameraServer()**"

→ "ConnectCamera()"

Remote USB camera via Ethernet:

Suitable for all greateyes cameras with USB interface. You can connect to up to four USB cameras connected with one server.

"SetupCameraInterface()"

→ "**ConnectToMultiCameraServer()**"

→ "GetNumberOfConnectedCams()"

→ "ConnectCamera()"

5.1.1. SetupCameraInterface()

Setup camera connection type with this function. Default connection type is USB. If you always and only use greateyes cameras via USB, you don't have to call this function.

Declaration:

C:
```
bool        SetupCameraInterface(int type, char* ipAddress,
               int& statusMSG, int addr)
```

Result:	true	Command successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

type	connectionType_USB (default) = 0	
		Connect camera via USB
	connectionType_Ethernet = 3	
		Connect camera via Ethernet TCP/IP or connect to a camera server with up to four USB cameras
ipAddress		IP address of greateyes camera server; only required if you want to connect via Ethernet interface
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.1.2. ConnectToSingleCameraServer()

Establish TCP connection to a greateyes camera with Ethernet interface. Please have a look at Connecting & disconnecting a greateyes camera for connection scheme.

Declaration:

C: `bool ConnectToSingleCameraServer(int addr)`

Result:	true:	Connection to the server was successful.
	false:	Server is not connected or an error has occurred.

Parameters the user must set:

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

5.1.3. ConnectToMultiCameraServer()

Establish TCP connection to a greateyes camera server which is connected to up to four greateyes USB cameras. Please have a look at Connecting & disconnecting a greateyes camera for connection scheme.

C: `bool ConnectToMultiCameraServer()`

Result:	true:	Connection to the server was successful.
	false:	Server is not connected or an error has occurred.

5.1.4. DisconnectCameraServer()

Disconnect from greateyes camera server. Call "**DisconnectCamera()**" first.

Declaration:

C: `bool DisconnectCameraServer(int addr)`

Result:	true:	Disconnecting from the server was successful.
	false:	Disconnecting was not successful; an error has occurred.

Parameters the user must set:

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

5.1.5. GetNumberOfConnectedCams()

Returns the number of greateyes USB cameras connected directly to the PC (via USB) or via the greateyes multi camera server (TCP/IP).

For connection via USB interface or via the greateyes multi camera server it is necessary to call "GetNumberOfConnectedCams()" befor calling "**ConnectCamera()**" Please have a look at Connecting & disconnecting a greateyes camera for the connection scheme.

Declaration:

C: `int GetNumberOfConnectedCams()`

Result: Number of connected cameras.

5.1.6. ConnectCamera()

Connects to up to four greateyes cameras simultaneously, either (exclusively) via USB or via Ethernet (TCP/IP).
Please have a look at Connecting & disconnecting a greateyes camera for the connection scheme.

Declaration:

C:
```
bool ConnectCamera(int& modelId, char*& modelStr, int&
   statusMSG, int addr)
```

Result:	true:	Camera is connected and ready for operation.
	false:	Camera is not connected or an error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

modelId	Integer value which is specific for the camera model that has been connected. Note: This number does not represent a serial number.
modelStr	String value containing the model name of the camera that has been connected.
statusMSG	Possible values of parameter statusMSG

5.1.7. DisconnectCamera()

Call this function to close the communication to the camera in any case.
Call this function before calling DisconnectCameraServer().

Declaration:

C: `bool DisconnectCamera(int& statusMSG, int addr);`

Result:	true	The camera has been closed successfully.
	false	The function has been called but e.g. the camera was not connected beforehand or an error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.1.8. InitCamera()

Call this function once to initialize the connected camera.

Declaration:

C: `bool InitCamera(int& statusMSG, int addr);`

Result:	true	The camera has been initialized successfully.
	false	The function has been called but e.g. the camera was not connected beforehand or an error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.1.9. Code Example

```
int  cameraAddr =0;
char*  ip = "192.168.1.234"
int  numberOfCamsConnected = 0;
int  connectionType = connectionType_USB;  //or connectionType_Ethernet

//set connectionType
if ( SetupCameraInterface( connectionType, ip, lastStatus, cameraAddr) == false )
{
        return false;
}

//ethernet: connect to single camera server (camera with ethernet interface)
if (connectionType == connectionType_Ethernet)
{
        if (ConnectToSingleCameraServer( cameraAddr ) == true)
        {
                numberOfCamsConnected = 1;
        }
}
//usb: get number of devices connected to the pc
else
{
        numberOfCamsConnected = GetNumberOfConnectedCams();
}

if (numberOfCamsConnected == 0)
{
        //no camera found
        return false;
}

//connect to camera; no matter which interface
if ( ConnectCamera(modelID, modelPtr, lastStatus, cameraAddr) == false)
{
        //on error - connecting to camera
        if (connectionType == connectionType_Ethernet)
        {
                DisconnectCameraServer(cameraAddr);
        }
        return false;
}

//initialize camera
if (InitCamera(lastStatus, cameraAddr) == false)
{
        //on error camera init
        DisconnectCamera(cameraAddr);

        if (connectionType == connectionType_Ethernet)
        {
                DisconnectCameraServer(cameraAddr);
        }
        return false;
}

return true;
```

5.2. *Get Functions*

5.2.1. GetDLLVersion()

Call this function to get the version number of the SDK.

Declaration:

C: `char*` `GetDLLVersion(int& size);`

Result: Returns the SDK version string.

Parameters returned by the function:

size — Number of characters in string.

5.2.2. GetFirmwareVersion()

Call this function to to get firmware version number of the greateyes camera.

Declaration:

C: `int` `GetFirmwareVersion(int addr);`

Result: Returns the camera firmware version number.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.2.3. GetImageSize()

Call this function to obtain the size of the next image. The size of the image depends on sensor size, crop mode set and binning mode set. Use these information to allocate the memory for the image. For example:
uint8_t *imgBuf = new uint8_t[width * height * bytesPerPixel]

C: `bool GetImageSize(int& width, int& height, int& bytesPerPixel, int addr);`

Result: success true/false

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

width		Number of columns along the readout register (pixels in x-direction)
height		Number of lines (pixel in y-direction)
bytesPerPixel		Number of bytes per pixel
	2	16 Bit: (for cameras with 16/18 bit ADC) Use a 16 bit type pointer or an 8 bit type pointer and allocate 2 bytes per pixel. Note: Saturation at 65535 counts for 18 bit ADC, too.
	3	24 Bit: (for cameras with 18 bit ADC) Use an 8 bit type pointer and allocate 3 bytes per pixel. Save 25% memory compared to 32 bit.
	4	32 Bit: (for cameras with 18 bit ADC) Use a 32 bit type pointer or an 8 bit type pointer and allocate 4 bytes per pixel. Easy to handle but 8 bit are useless.

5.2.4. GetSizeOfPixel()

Call this function to get the physical pixel size of the sensor.

C: `int GetSizeOfPixel(int addr);`

Result:	[µm]	Pixel size. This is a constant and only depends on the connected camera model.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.2.5. DllIsBusy()

Call this function to check busy status of the camera SDK. The SDK is busy when a function is already running, e.g. during image taking when the camera is exposing.

Declaration:

C: `bool DllIsBusy(int addr);`

Result:	true	camera SDK is busy.
	false	camera SDK is not busy and ready to perform a new task.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.2.6. GetMaxExposureTime()

Call this function to get the maximum exposure time supported by the camera model. The maximum value mainly depends on the firmware version of the camera.

Declaration:

C: `int GetMaxExposureTime(int addr);`

Result: maximum exposure time in ms

5.2.7. GetMaxBinningX() / GetMaxBinningY()

Returns the maximum possible value for parameter binningX / binningY, see SetBinningMode(). The result depends on the sensor type/geometry and on the crop mode setting that is currently active.

Declaration:

C:
```
int             GetMaxBinningX(int& statusMSG, int addr);
int             GetMaxBinningY(int& statusMSG, int addr);
```

Result: max. possible value for parameter binningX or binningY.

Parameters the user must set:

addr	0..3	Index of connected camera devices. This Index begins at addr = 0 for the first device

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.2.8. SupportedSensorFeature()

This function provides information about the supported sensor features of the sensor.

Declaration:

C:
```
bool        SupportedSensorFeature(int feature, int& statusMSG,
                int addr);
```

Result: feature supported / not supported

Parameters the user must set:

feature	0 = sensorFeature_capacityMode:	Sensors with this feature can operate in the capacity mode.
	1 = sensorFeature_binningX:	Sensors with this feature can bin in x – direction (serial).
	2 = sensorFeature_cropX:	Sensors with this feature can operate in the crop mode.
addr	0..3	Index of connected camera devices. This Index begins at addr = 0 for the first device

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.2.9. GetNumberOfSensorOutputModes()

Returns the number of possible output modes for a given camera model. Usually a sensor has one single output only. For larger format sensors, e.g. 4096px x 4096px sensors, (modelID = 12) up to 10 output modes are specified. Please also see SetupSensorOutputMode() for more details.

Declaration:

C:
```
int         GetNumberOfSensorOutputModes(int addr);
```

Result: Number of possible output modes.

Parameters the user must set:

addr	0..3	Index of connected camera devices. This Index begins at addr = 0 for the first device

5.2.10. GetSensorOutputModeStrings()

Returns a string descriptor for a selected sensor output mode, see function GetNumberOfSensorOutputModes().

Declaration:

C:
```
const char* GetNumberOfSensorOutputModes(int index, int
            addr);
```

Result: String descriptor for selected output mode.

Parameters the user must set:

index		Index of output mode [0 .. (NumberOfSensorOutputModes - 1)].
addr	0..3	Index of connected camera devices. This Index begins at addr = 0 for the first device

5.2.11. GetLastMeasTimeNeeded()

Returns the total time that was required to perform the last measurement. This includes the exposure time, potential shutter open/close delays as well as the actual time that was required to read out the CCD and transfer the data to the client PC.

Declaration:

C: `float GetLastMeasTimeNeeded(int addr);`

Result: [ms] Total time required for last measurement

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.3. *Set Functions*

5.3.1. SetExposure()

Call this function to set the exposure time for the subsequent measurements.

Declaration:

C:
```
bool        SetExposure(int exposureTime, int& statusMSG,
               int addr);
```

Result:		
	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

exposureTime	1..2,000,000	Exposure time in ms.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.2. SetReadOutSpeed()

Call this function to setup the readout speed.

C:
```
bool        SetReadOutSpeed(int readoutSpeed, int& statusMSG,
                int addr);
```

Result:		
	true	Readout speed set successfully.
	false	The function has been called but e.g. the chosen readout speed was not availeble or an error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

readoutSpeed	readoutSpeed_1_MHz, readoutSpeed_3_MHz, readoutSpeed_500_kHz, readoutSpeed_250_kHz, readoutSpeed_100_kHz, readoutSpeed_50_kHz	
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.3. SetBinningMode()

Call this function to setup the binning mode.

C:
```
bool      SetBinningMode(int binningX, int binningY,
                int& statusMSG, int addr);
```

Result:

true	Binning mode set successfully.
false	The function has been called but e.g. the chosen binning mode was not available or an error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

binningX — 1 .. numPixelInX — Number of pixels to be binned in x-direction

In contrast to a binning in y-direction this does not reflect in a reduction of the total time required for the image readout since the same amount of pixels needs to be clocked through the horizontal shift register(s). However, binning or summing in x-direction can be used to reduce the overall signal-to-noise ratio (SNR) in scenes with low exposures.

Note: For backwards compatibility, when using cameras with earlier firmware revisions (rev. 11 or lower) the binningX parameter is interpreted differently as shown below:

2^{binningX} = Number of pixels to be binned in x-direction

In firmware revisions 11 or lower the binning in x-direction was realized in software, thus it does not reduce the overall signal-to-noise ratio (SNR).

binningY — 1 .. numPixelInY — Number of pixels to be binned in y-direction

Horizontal binning is realized in hardware on the CCD sensor and hence is useful to e.g. accelerate the time which is required to read out the CCD sensor and improves the overalls signal-to-noise ratio (SNR).

Note: For backwards compatibility, when using cameras with earlier firmware revisions (rev. 11 or lower) the binningY parameter is interpreted differently as shown below:

0	No binning of lines
1	Binning of 2 lines
2	Binning of 4 lines
3	Binning of 8 lines
4	Binning of 16 lines

	5	Binning of 32 lines
	6	Binning of 64 lines
	7	Binning of 128 lines
	8	Full vertical binning

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.4. SetShutterTimings()

Call this function to set shutter timings to be used in automatic shutter mode.

Declaration:

C:

```
bool        SetShutterTimings(int openTime, int closeTime,
               int& statusMSG, int addr);
```

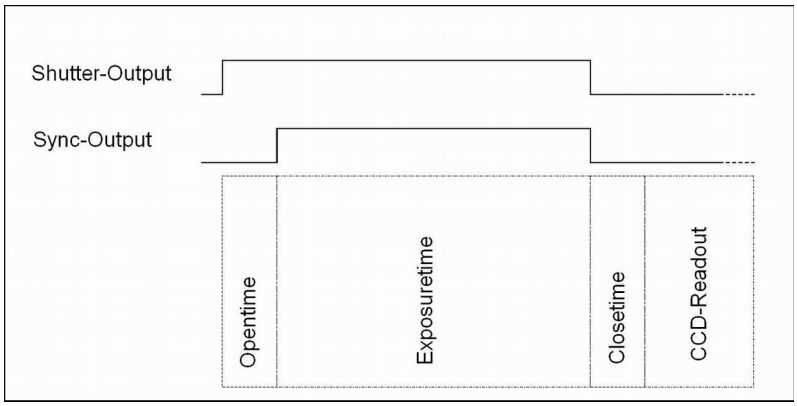
Result:	true	No error has occurred; command successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

openTime	[1..255] ms	Time to wait before the start of the exposure to ensure that the shutter is fully opened.
CloseTime	[1..255] ms	Time to wait after exposure and before starting the readout to ensure that the shutter is fully closed.
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG



5.3.5. OpenShutter()

Call this function to open or close the shutter manually, or to automatically open/close the shutter before/after the actual exposure time window.

Declaration:

C: `bool OpenShutter(int state, int& statusMSG, int addr);`

Result:	true	No error has occurred; command successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

state:	0	close shutter
	1	open shutter
	2	auto shutter Automatically open and close the shutter before/after the actual exposure time window based on timings defined via a call of SetShutterTimings().
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.6. SyncOutput()

Call this function to manually set the “SYNC” TTL trigger output high/low.

Declaration:

C: `bool SyncOutput(bool syncHigh, int& statusMSG, int addr);`

Result:	true	No error has occurred; command successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

syncHigh:	true	Sets the “SYNC” TTL trigger output to TTL high level.
	false	Sets the “SYNC” TTL trigger output to TTL low level.
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.7. SetupBurstMode()

Call this function to configure the burst mode.

Declaration:

C: `bool SetupBurstMode(int numberOfMeasurements, int& statusMSG, int addr);`

Result:	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

numberOfMeasurements		Number of measurements to be taken in series.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.8. ActivateBurstMode()

Call this function to activate the burst mode. If the burst mode is active, the camera will perform several sequential measurements upon calling StartMeasurement_DynBitDepth() or PerformMeasurement_Blocking_DynBitDepth(). Call SetupBurstMode() to configure the burst mode.
Note: In contrast to a single image acquisition all measured data are transmitted as a single large image array wherein the data of individual images are appended.

Declaration:

C: `bool ActivateBurstMode(bool status, int& statusMSG, int addr);`

Result:	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

status		Sets burst mode on/off.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

Example usage:

```
if (SetupBurstMode(numberOfMeasurements, statusMSG, addr) == false)
    return false;

if (ActivateBurstMode(true, statusMSG, addr) == false)
    return false;

if (GetImageSize(width, height, bytesPerPixel, ..) == false)
    return false;

 //allocate memory
uint8_t* imgBuf= new uint8_t[width * height *  bytesPerPixel];

 //get measurements
if (PerformMeasurement_Blocking_DynBitDepth(.., imgBuf, ..) == false)
    return false;

// imgBuf contains measurements now
return true;
```

5.3.9. SetupCropMode2D()

Call this function to define the number of lines and columns to be readout using crop mode. The crop mode defines a rectangular window starting at Pixel (0,0) with respect to the selected output amplifier position on the CCD sensor. The rest of the (unwanted) image data will be dumped/discarded on the CCD during the readout process.
Note: Certain sensor types allow for a cropped readout in y-direction only.
SupportedSensorFeature()

Declaration:

C:
```
bool        SetupCropMode(int col, int line, int& statusMSG,
             int addr);
```

Result:	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

col		Number of columns (x-direction) to be read out.
line		Number of lines (y-direction) to be read out.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.10. ActivateCropMode()

Call this function to activate the crop mode. In crop mode you can read out the sensor starting from the lowermost up to a certain line (y-direction) that previously was defined using SetupCropMode2D().

Declaration:

C:
```
int         ActivateCropMode(bool status, int& statusMSG,
               int addr);
```

Result: Number of lines that currently is set for crop mode.

Parameters the user must set:

status		Sets crop mode on/off
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

Example usage:

```
if (SetupCropMode2D(col, line, statusMSG, addr) == false)
    return false;

if (ActivateCropMode(true, statusMSG, addr) == false)
    return false;

if (GetImageSize(width, height, bytesPerPixel, ..) == false)
    return false;

//allocate memory
unsigned char* imgBuf= new unsigned char[width * heigth *  bytesPerPixel];

//get measurements
if (PerformMeasurement_Blocking_DynBitDepth(.., imgBuf, ..) == false)
        return false;

// imgBuf contains measurements now
return true;
```

5.3.11. SetupGain()

Call this function to set different pre-amplifier gain modes of the camera.

Declaration:

C: `bool SetupGain(int gainSetting, int& statusMSG, int addr);`

Result:	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

gainSetting	0	Sets gain setting to maximum dynamic range.
	1	Sets gain setting to provide highest sensitivity.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.12. SetupCapacityMode()

Call this function to set different output node capacity modes. Parts of the CCDs support for switching between two different output node capacities, a standard mode and a high signal mode. The latter is useful in binned readout operation and usually allows for a two- to threefold output node capacity with the trade-off being an output sensitivity (gain) reduced by approx. the same factor.

Declaration:

C:
```
bool SetupCapacityMode(bool capacityMode, int& statusMSG,
 int addr);
```

Result:	true	No error has occurred; parameter was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

capacityMode	false	Standard Capacity (Low Noise)
	true	Extended Capacity (High Signal)
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.13. SetupTransferOptions()

Call this function to set different special readout options for USB cameras.

Declaration:

C:
```
bool        SetupTransferOptions(bool safeFifoMode,
                bool safeUsbMode);
```

Result:	true	No error has occurred; parameters were set successfully.
	false	Unsuccessful; an error has occurred.

Parameters the user must set:

safeFifoMode:	(*default:* false)	In safeFifoMode a ClearFifo() always is performed before any StartMeasurement_DynBitDepth() and TemperatureControl_GetTemperature() function. It takes about 50 ms of extra time for each measurement. Disabling safeFifoMode accelerates the mentioned read out functions, but in some cases it can cause errors (rarely).
safeUsbMode:	(*default: false)*	Sometimes slow computers may cause problems with USB communication at internally triggered exposure times between 1 – 1000 ms. If enabled, only internally triggered measurements with exposure times between 1 – 1000 ms will start in safeUsbMode. Note: The safeUsbMode does not support **simultaneous image acquisitions** from multiple cameras with USB interface connected to one computer.

5.3.14. SetupSensorOutputMode()

Sets the sensor output mode for the subsequent measurements. It determines the amplifiers and registers used for image readout.
Usually sensor of smaller formats have a single output only. Large format sensors, with e.g. 2048px x 2048px usually provide two output nodes, one at each end of the serial readout register. 4096px x 4096px sensors (modelID = 12) support image readout via two serial registers and hence up to four output nodes in total.

Declaration:

C:
```
bool        SetupSensorOutputMode(int sensorOutputMode,
             int addr);
```

Result:

true	No error has occurred; parameter was set correctly.
false	An error has occurred;

Parameters the user must set:

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.
SensorOutputMode	[0 ..(NumberOfSensorOutputModes - 1)]	

Note: Each bit in a 4 Bit Code enables one of up to four output amplifiers for readout. Hence, possible options are e.g.:

1	1 Amp. (E) - image is read out through amplifier (E) using one output register
2	1 Amp. (F) - image is read out through amplifier (F) using one output register
4	1 Amp. (G) - image is read out through amplifier (G) using one output register
8	1 Amp. (H) - image is read out through amplifier (H) using one output register
3	2 Amps, 1 Reg.(E,F) - image is read out through amplifier (E) and (F) using one output register
5	2 Amps, 2 Reg.(E,G) - image is read out through amplifier (E) and (G) using two output registers
9	2 Amps, 2 Reg.(E,H) - image is read out through amplifier (E) and (H) using two output registers
6	2 Amps, 2 Reg.(F,G) - image is read out through amplifier (F) and (G) using two output registers
10	2 Amps, 2 Reg.(F,H) - image is read out through amplifier (F) and (H) using two output registers
12	2 Amps, 1 Reg.(G,H) - image is read out through amplifier (G) and (H) using one output register
15	4 Amps (E,F,G,H) - image is read out through all four amplifiers, using two output registers

5.3.15. ClearFifo()

Call this function to clear FIFO memory.

Declaration:

C: `int ClearFifo (int& statusMSG, int addr);`

Result: Number of cleared memory blocks (in bytes).

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.3.16. SetExtTriggerTimeOut()

Call this function to set timeout for external trigger.

Declaration:

C:
```
bool SetExtTriggerTimeOut(int extTriggerTimeOut
      int addr);
```

Result:	true	No error has occurred; parameters were set successfully.
	false	Unsuccessful; an error has occurred.

Parameters the user must set:

extTriggerTimeOut		1 .. 65535 ms
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

5.3.17. SetLEDStatus()

Call this function to switch backside LED's on/off.

Declaration:

C:
```
bool        SetLEDStatus(bool status, int& statusMSG,
                int addr);
```

Result:		
	true	No error has occurred; parameters were set successfully.
	false	Unsuccessful; an error has occurred.

Parameters the user must set:

status	true	LEDs on
	false	LEDs off
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.3.18. SetBitDepth()

Call this function to set bit depth of incoming data array for cameras with 18 bit ADC (max. 20 bit dynamic range through oversampling).
For cameras with 16 bit ADC the incoming data array is always 16 bit.

Declaration:

C:
```
bool        SetBitDepth(int bytesPerPixel, int& statusMSG,
                int addr);
```

Result:		
	true	No error has occurred; parameters were set successfully.
	false	Unsuccessful; an error has occurred.

Parameters the user must set:

bytesPerPixel	2..4	Sets incoming data array to 2,3 or 4 bytes per pixel. See GetImageSize(). *Default*: *4*
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.4. Camera Cooling

5.4.1. TemperatureControl_Init()

Call this function to initialize the sensor cooling hardware of your camera.

Declaration:

C:
```
int TemperatureControl_Init(int coolingHardware,
int& minTemperature,int& maxTemperature,int&
statusMSG, int addr);
```

Result:		
	-1	An error has occurred.
	≥0	Number of available cooling level. (This value is obsolete! This value is relevant for obsolete functions `TemperatureControl_SetTemperatureLevel()` and `TemperatureControl_GetLevelString()` only. No error has occurred.

Parameters the user must set:

coolingHardware		Choose an option matching your cooling hardware. The appropriate coolingHardware setting for your camera can be found in a text file called "Temperature-Hardware-Option.txt" within the SDK folder of your software release. If no such file was provided, please contact us. (sw@greateyes.de)
addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

minTemperature	[°C]	minimal possible value for parameter temperature of the function TemperatureControl_SetTemperature()
maxTemperature	[°C]	minimal possible value for parameter temperature of the function TemperatureControl_SetTemperature()
statusMSG		Possible values of parameter statusMSG

5.4.2. TemperatureControl_SetTemperature()

This function sets the target temperature of the CCD sensor cooling system.

Declaration:

C: `bool TemperatureControl_SetTemperatureLevel( int temperature, int& statusMSG, int addr);`

Result:		
	true	No error has occurred; temperature was set correctly.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

temperature	[°C]	Target temperature of the temperature control. Threshold value: Parameter `minTemperature` and `maxTemperature` of the function TemperatureControl_Init()
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.4.3. TemperatureControl_GetTemperature()

Call this function to read back the actual temperature of the CCD sensor or of the backside of the thermoelectric cooling element (TEC).

Declaration:

C:
```
bool        TemperatureControl_GetTemperature(int thermistor,
               int& temperature, int& statusMSG, int addr);
```

Result:		
	true	No error has occurred; temperature was read successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

thermistor		Chooses the sensing element (e.g. thermistor) → 0: sensor temperature → 1: temperature of TEC backside
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

temperature	contains temperature in degree of Celsius [°C]
statusMSG	Possible values of parameter statusMSG

Note: If the function returns statusMSG = 11, the camera resets cooling control because the TEC backside temperature is getting too high. In this case you should set coolingLevel to room temperature by calling the function TemperatureControl_SetTemperature() or by a call of TemperatureControl_SwitchOff().

5.4.4. TemperatureControl_SwitchOff()

Call this function to switch off the cooling of the camera sensor.

Declaration:

C:
```
bool        TemperatureControl_SwitchOff(int& statusMSG,
               int addr);
```

Result:	true	No error has occurred; command successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

addr	0..3	Index of connected devices. This index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.5. Image Acquisition

5.5.1. PerformMeasurement_Blocking_DynBitDepth()

This function is used to obtain the 16, 24 or 32 bit image data with 16,18, 19 or 20 bit dynamic range from the CCD sensor. It starts a measurement and performs a readout of the image data after the exposure time has passed. This function blocks the calling thread for the duration of the measurement.

Declaration:

C: `bool PerformMeasurement_Blocking(bool correctBias, bool showSync, bool showShutter, bool triggerMode, int triggerTimeOut, void* pIndataStart, int& statusMSG, int addr);`

Result:	true	Image data was acquired successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

correctBias	false	Bias correction disabled.
	true	Each line is intensity corrected dependent on the dark pixel values at the left and right periphery of this line. The higher the sensor resolution the more accurate this correction will work since there are more dark pixel to calculate the correction parameters. Note: This correction lowers the saturation level.
showSync (fire signal)	false	The Sync-Output is disabled. It is always low.
	true	The Sync-Output is enabled. In this case the Sync-Output of the camera goes high during the exposure time window and remains low otherwise. (The output is TTL level.)
showShutter	false	Shutter control is disabled. Shutter settings defined via OpenShutter() will be ignored and the shutter will always remain closed during the measurement. This is useful e.g. for background measurements such that there is no need to change shutter mode via the OpenShutter() function beforehand.
	true	Shutter control is enabled. Shutter settings defined via OpenShutter() will be used (manually opened/closed/auto shutters will stay in there previous state throughout the measurement).
triggerMode	false	The camera is triggered internally. Each time the functions StartMeasurement_DynBitDepth() or PerformMeasurement_Blocking_DynBitDepth() are called, an image is taken and read out of the camera.

	true	The camera is triggered externally. In this case the camera delivers an image when the external trigger input changes from L to H (TTL). If there no trigger is received within the triggerTimeOut window, the function returns a timeout error (see statusMSG).
triggerTimeOut		Trigger timeout in ms. This value only is used if the parameter triggerMode is set to true.
pindataStart		This is a pointer to a memory area where the image data returned by the CCD sensor will be stored. Before calling PerformMeasurement_Blocking_DynBitDepth() or the calling program needs to allocate the correct amount of memory depending on the image geometry. Call GetImageSize() beforehand to obtain `width`, `height` and `bytesPerPixel` of the image.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.5.2. StartMeasurement_DynBitDepth()

This function is used to start a measurement process running in a thread.
Once the thread finished the image data can be obtained through using GetMeasurementData_DynBitDepth().
Use the DllIsBusy() function to check the status of the ongoing measurement.

Declaration:

C:
```
bool StartMeasurement_DynBitDepth(bool correctBias, bool showSync, bool showShutter, bool tiggerMode, int& statusMSG, int addr);
```

Result:	true	Starting the measurement successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

correctBias	false	Bias correction disabled.
	true	Each line is intensity corrected dependent on the dark pixel values at the left and right periphery of this line. The higher the sensor resolution the more accurate this correction will work since there are more dark pixel to calculate the correction parameters. Note: This correction lowers the saturation level.
showSync (fire signal)	false	The Sync-Output is disabled. It is always low.
	true	The Sync-Output is enabled. In this case the Sync-Output of the camera goes high during the exposure time window and remains low otherwise. (The output is TTL level.)
showShutter	false	Shutter control is disabled. Shutter settings defined via OpenShutter() will be ignored and the shutter will always remain closed during the measurement. This is useful e.g. for background measurements such that there is no need to change shutter mode via the OpenShutter() function beforehand.
	true	Shutter control is enabled. Shutter settings defined via OpenShutter() will be used (manually opened/closed/auto shutters will stay in there previous state throughout the measurement).
triggerMode	false	The camera is triggered internally. Each time the functions StartMeasurement_DynBitDepth() or PerformMeasurement_Blocking_DynBitDepth() are called, an image is taken and read out of the camera.
	true	The camera is triggered externally. In this case the camera delivers an image when the external trigger input changes from L to H (TTL). If there no trigger is received

within the triggerTimeOut window, the function returns a timeout error (see statusMSG).

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.5.3. GetMeasurementData_DynBitDepth()

This function is used to obtain the 16, 24 or 32 bit image data with 16/18 bit dynamic range of the last measurement performed using StartMeasurement_DynBitDepth().

Declaration:

C:
```
bool        GetMeasurementData (void* pIndataStart,
                int& statusMSG, int addr);
```

Result:	true	Image data was acquired successfully.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

pindataStart		This is a pointer to a memory area where the image data returned by the CCD sensor will be stored. Before calling PerformMeasurement_Blocking_DynBitDepth() or the calling program needs to allocate the correct amount of memory depending on the image geometry. Call GetImageSize() beforehand to obtain `width`, `height` and `bytesPerPixel` of the image.
addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

Parameters returned by the function:

statusMSG	Possible values of parameter statusMSG

5.5.4. StopMeasurement()

This function stops an ongoing measurement that was initiated by the StartMeasurement_DynBitDepth() function.
After the measurement has stopped the DllIsBusy() function will return false and the GetMeasurementData_DynBitDepth() function will return statusMSG = 12 (MeasurementStopped).

Note: StopMeasurement() does not work if the measurement was initiated by calling the PerformMeasurement_Blocking_DynBitDepth() function.

Declaration:

C: `bool StopMeasurement(int addr);`

Result:	true	No error has occurred; measurement stopped.
	false	An error has occurred; use statusMSG to analyse the type and source of the error.

Parameters the user must set:

addr	0..3	Index of connected devices. This Index begins at addr = 0 for the first device.

5.5.5. Example

```
//get image size
if (GetImageSize( width, height, bytesPerPixel, lastStatus, cameraAddr) == false)
        return false;

//allocate memory
uint8_t* pInData=new uint8_t[width * height * bytesPerPixel];

//start measurement
if ( StartMeasurement_DynBitDepth( .., lastStatus, cameraAddr) == false)
        return false;

//wait for measurement
while ( DllIsBusy( cameraAddr ) )
{
        std::cout << "." ;
}
std::cout <<  "measurement finished" << std::endl;

//obtain measurement
if ( GetMeasurementData_DynBitDepth(pInData, lastStatus, cameraAddr) == false)
        return false;

// pInData contains image now
uint8_t* charData = pInData;
uint16_t* shortData = reinterpret_cast <uint16_t*>(pInData);
uint32_t* longData = reinterpret_cast <uint32_t*>(pInData);
int pixelVal = 0;

//print out pixel values of the first line
std::cout <<  "first line: " << std::endl;
for (int x = 0; x < width; x++)
{
        switch (bytesPerPixel)
        {
        case 2:
                pixelVal = *shortData;
                shortData++;
                break;
        case 3:
                pixelVal = charData[0] + (charData[1] << 8) + (charData[2] << 16);
                charData += bytesPerPixel;
                break;
        case 4:
                pixelVal = *longData;
                longData++;
                break;
        default:
                break;
        }
        std::cout <<  pixelVal << std::endl;
}

delete [] pInData;
```

5.5.6. Possible values of parameter statusMSG

Value	*Meaning*	*Description*
0	Camera OK & connected	No error has occurred; command or image data readout was successfully. A camera is (still) connected and ready for operation.
1	No camera connected	No greateyes camera is connected or camera was not found.
2	Couldn't open USB device	There is a problem with the USB-Controller and/or USB connection (i.e. cabling).
3	WriteConfigTable failed	Writing of data to the camera has failed.
4	WriteReadRequest failed	Reading of data from the camera has failed.
5	No trigger	No trigger signal received within timeout window.
6	New camera detected	A new Camera has been plugged in and is ready for operation. This message may be used to trigger e.g. the initial transfer of camera settings upon camera plug-in or program start.
7	Unknown model ID	The camera connected is not supported by the library. Please consult greateyes for updates and/or support.
8	Out of range	One of the supplied parameters exceeds the valid range of values.
9	No new data	No new data available.
10	Busy	Camera is busy at the moment.
11	Cooling turned off	Cooling control resets because backside temperature is too hot
12	Measurement stopped	Measurement stopped by StopMeasurement function.
13	Burst Mode – too much pixels	Maximum amount of pixel exceeded in burst mode. Set a lower total number of measurements or configure a higher binning level.
14	Timing table not found	The timing table for selected readout speed not found.
15	Not critical	Function stopped but there is no critical error (no valid result; catched division by zero). please try to call the function again.
16	Illegal combination of binning and crop mode	For firmware revision 11 and earlier it is not possible to combine crop and binning of lines.

6. Including the DLL

C/C++: To access the greateyes.dll it is required to include the header file (greateyes.h) in your source and add the "greateyes.lib" and "greateyes.dll" files to your project.

7. Copyright Information

8. Contact Information

greateyes GmbH
Justus-von-Liebig-Straße 2
12489 Berlin
Germany

phone: +49 (0) 30 912075-250
fax: +49 (0) 30 912075-251
Web: www.greateyes.de
email: info@greateyes.de
sw@greateyes.de