CCAPI

Release 14 Jun 2006

Software manual

C-Cam Camera Systems

Implementation for the:

Windows NT/2000/XP

and

Linux

operating systems

C-Cam Technologies

a division of **Vector International**

Contact address

C-Cam Technologies
division of
Vector International

Interleuvenlaan 46,
B-3001 Leuven
Belgium

Tel. +32 (0)16 40 20 16
Fax +32 (0)16 40 03 23

email info@vector-international.be
http://www.vector-international.be

Table of Contents

1 Introduction

This document describes the Application Program Interface for all cameras, controlled via our PCI Interface, via our USB-connection, and in the future also over the IEEE1394-bus.

The aim of CCAPI is to present the user with a stable API for all cameras, current and future. Keeping this API up to date with new cameras and interfaces will be substantially easier than creating and maintaining separate APIs for every camera/interface combination.

Rebuilding your source code for this new API is relatively straightforward.

New acquisition modes have been added. In the past acquisition and data processing was only performed sequentially as image acquisition was ‘synchronous’, which means you had to wait for the call to return with the image data. Now you can perform ‘asynchronous IO’ by specifying this on the CC-Open call. The API then uses the “OVERLAPPED’ mode as documented in the Windows documentation. This ‘Overlapped’ mode allows the application program to attend to other functions while the IO operation finishes in the background. This can enhance overall program speed significantly.

Also new is the possibility to specify a Window of Interest (WOI) in different ways. In the former version, the first and last pixels of the WOI were the basic references as co-ordinates. In the new CCAPI it is also possible to address the WOI using a pixel position together with the relative size of the WOI. This simplifies calculations when the WOI is moved over the sensor area.

If you have questions regarding this document, please e-mail to c-cam@vector-international.be. We will be glad to help you.

The engineering team of C-Cam hopes you enjoy their effort in enhancing the industrial digital camera revolution.

C-Cam Technologies

Leuven, August 26th, 2005.

2 Function summary

Management Functions

Camera Control Functions

Capture Functions

Miscellaneous Functions

Error Handling Functions

3 API Functions – in alphabetical order

3.1 CC_CameraManagement

•**Function:**

Reads and writes various information to the camera.
Applicable to USB cameras only.
This function is documented here for completeness and is for internal use or advanced use only.

•**Return value:**

FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CameraManagement( Handle Cam ,
                                          int flags ,
                                          unsigned long offset ,
                                          unsigned long size ,
                                          char * srcdst ,
                                          enum CC_PROGRESS_FLAGS Flags ,
                                          void * Arg1 ,
                                          long Arg2
                                          ) ;
```

Variable	C Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
flags	*int*	See list below	Defining operation modes
offset	*unsigned long*		Start address
size	*unsigned long*		Count of bytes to be written or read
srcdst	*char **		Pointer to buffer holding data to be written or receiving data to be read
Flags	*enum CC_PROGRESS_FLAGS*		Currently not used. Intended to be used as shown in CC_OpenEx.
Arg1	*void **		
Arg2	*long*		

• **flags**

Index	Application
CC_CM_WRITE_I2C *	To write to the I2C Eeprom holding the firmware for the USB-Controller to laod on boot
CC_CM_READ_I2C *	To read from the I2C Eeprom holding the firmware for the USB-Controller to laod on boot
CC_CM_WRITE_INFO *	To write to the area in I2C Eeprom where 112 bytes are reserved for user storage of camera parameters and such
CC_CM_READ_INFO	To read from the area in I2C Eeprom where 112 bytes are reserved for user storage of camera parameters and such
CC_CM_WRITE_SDF *	To write the Serial Data Flash on selected cameras.
CC_CM_READ_SDF *	To read the Serial Data Flash on selected cameras.
CC_CM_READ_FIRMWARE *	To write directly into the Program/Data RAM of the USB-Controller.
CC_CM_WRITE_FIRMWARE *	To read directly into the Program/Data RAM of the USB-Controller.

Operations marked with a '*' are not allowed in standard Camera Applications.

•**Example code**

```
CC_CameraManagement( Cam , CC_CM_READ_INFO, 0 , 4 , & user_info1 ,
                                   CC_PROGRESS_FLAGS_NONE , NULL , NULL ) ;
```

3.2 CC_CamStatus

•Function:
Returns status data from camera.

•Return value:
FALSE when operation was not successful. TRUE if successful.

•Syntax definition:

```
DLLINOUT BOOL WINAPI CC_CamStatus( Handle Cam ,
                                    CCCAM_STATUS Index ,
                                    long * Status
                                    );
```

Variable	C Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Index	CC_CAM_STATUS	Enumeration type	Selects the type of information
Status	* long	Depends on camera and on requested information	Actual value of requested status

•Index
Depending up on the interface and camera option, the camera or the interface can return some status about the camera attached:

Index	Application
CC_CAM_STATUS_PCI	When not transmitting valid data, the camera may transmit auxiliary data over the image transfer channel. The resulting data depends on the actual camera attached. See corresponding camera manual.
CC_CAM_STATUS_CAMERA	Some cameras will transmit a status back while receiving a command. The resulting data depends on the actual camera attached.

```
typedef enum {CC_CAM_STATUS_PCI = 0x0,
              CC_CAM_STATUS_CAMERA
              } CC_CAM_STATUS ;
```

•Example code

```
CC_CamStatus(camhandle, CAM_STATUS_PCI , &status );
```

3.3 CC_CaptureAbort

- **Function:** Function to abort the acquisition procedure in the asynchronous[1] mode.
- **Return value:** FALSE when operation was not successful. TRUE if successful.
- **Related functions:** CC_CaptureWait, CC_CaptureStatus, CC_CaptureSequence
- **Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureAbort(Handle Cam);
```

Variable	C Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)

- **Example code:**

```
    CC_CaptureAbort(camhandle);
```

[1] Asynchronous mode: image acquisition in parallel with the software application.

3.4 CC_CaptureAbortEx

•**Function:** Function to abort the acquisition procedure in the asynchronous[2] mode.

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Related functions:** CC_CaptureAbort, CC_CaptureWait, CC_CaptureStatus, CC_CaptureSequence

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureAbort(Handle Cam ,
                                          enum CC_CAPTUREABORT Flags);
```

Variable	C Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Flags	unsigned int	See list below	Defining operation modes

Depending up on the application, the different things can be reset in the camera, these are:

Flag	Application
CC_CA_AUXILIARIES	See camera information.
CC_CA_USBRESET	Clears the Endpoint in the USB controller in the camera and clears the USB stack for this device
CC_CA_MAIN	Resets the camera logic. Issuing a CC_CaptureAbortEx() with only this flag equals issuing a CC_CaptureAbort()

Although these Flags are defined as an enumeration, they are meant to be bit-wise OR-ed to make up the Flags-specification in the CC_CaptureAbortEx call.

```
enum CC_CAPTUREABORT {
                    CC_CA_AUXILIARIES   = 0x01 ,
                    CC_CA_USBRESET      = 0x02 ,
                    CC_CA_MAIN          = 0x04
                    } ;
```

• **Example code:**

```
CC_CaptureAbortEx(camhandle , CC_CA_MAIN | CC_CA_USBRESET );
```

[2] Asynchronous mode: image acquisition in parallel with the software application.

3.5 CC_CaptureArm[3]

•**Function:** Acquisition with trigger mode with single image transfer.

•**Related function:** CC_CaptureData

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureArm( HANDLE Cam ,
                                   CC_TRIGGER_MODE TriggerMode ) ;
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
TriggerMode	CC_TRIGGER_ MODE	Enumeration type	Trigger type. Modes are listed in a type structure, see this paragraph below.

•**TriggerMode :**

Depending on the interface and camera option, the camera and/or the interface can work in several trigger modes. These modes are:

Trigger mode	Application
CC_NO_TRIGGER	Camera runs in normal mode. As soon as the capture function is called, the acquisition is started immediately. Only a single image will be captured. Software trigger.
CC_INTERFACE_TRIGGER_SINGLE	The interface will wait until the interface gets a trigger pulse. The interface will not respond to other successive triggers. Hardware trigger.
CC_INTERFACE_TRIGGER_CONTINUOUS	Camera expects a remote trigger that will be generated by the PCI interface and will capture an image for every trigger. Hardware trigger.
CC_CAMERA_TRIGGER_SINGLE	The camera will wait until the camera gets a trigger pulse. The camera will not respond to a successive trigger. Hardware trigger.
CC_CAMERA_TRIGGER_CONTINUOUS	The camera will capture an image for every trigger pulse. Hardware trigger.

[3] Only supported for USB cameras

Trigger mode	Application
CC_CAMERA_CONTINUOUS	Successive single shots - as fast as it goes. Software trigger.
CC_CAMERA_CONTINUOUS_TIMED	Successive single shots – timed. Software trigger.
CC_CAMERA_CONTINUOUS_ROLLING	Continuous rolling shutter operation. Software trigger.

```
typedef enum { CC_NO_TRIGGER = 0x0 ,
               CC_INTERFACE_TRIGGER_SINGLE ,
               CC_INTERFACE_TRIGGER_CONTINUOUS ,
               CC_CAMERA_TRIGGER_SINGLE ,
               CC_CAMERA_TRIGGER_CONTINUOUS ,
               CC_CAMERA_CONTINUOUS ,
               CC_CAMERA_CONTINUOUS_TIMED ,
               CC_CAMERA_CONTINUOUS_ROLLING
              } CC_TRIGGER_MODE;
```

Error handling:

•The camera handle input is not checked. It must be obtained from CC_Open or CC_OpenEx.
•CC_ERROR_CAPTURE_ARM_INVALID : An unknown trigger mode was specified.

•Example code:

```
CC_CaptureArm (cam ,                          // load camera location
               CC_CAMERA_TRIGGER_SINGLE   // trigger mode
               );
```

3.6 CC_CaptureBackground[4]

•**Function:** Acquires images in the background using physical memory freed at boot time.

•**Related functions:** CC_CaptureBackgroundEx, CC_CaptureBackgroundAbort

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureBackground( HANDLE Cam ,
                                    unsigned long Physaddr ,
                                    unsigned long TransferSize  ) ;
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Physaddr	Unsigned long		Hardware Start address of available physical memory
TransferSize	Unsigned long		Buffer size, must be smaller or equal to the size of the freed / reserved physical memory

•**Example code:**

```
see MapPhysMem() example
```

[4]Only supported for PCI-based camera systems

3.7 CC_CaptureBackgroundAbort[5]

•**Function:** Function to abort the background acquisition.

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Related functions:** CC_CaptureBackground

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureBackgroundAbort(Handle Cam);
```

Variable	C Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previously opened device (= camera)

•**Example code:**

```
CC_CaptureBackgroundAbort(camhandle);
```

[5]Only supported for PCI-based camera systems

3.8 CC_CaptureBackgroundEx[6]

•**Function:** Acquires images in the background using physical memory freed at boot time.

•**Related functions:** CC_CaptureBackgroundEx, CC_CaptureBackgroundAbort

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureBackgroundEx( HANDLE Cam ,
                                             unsigned long Physaddr ,
                                             unsigned long TransferSize ,
                                             unsigned long Flags ) ;
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Physaddr	Unsigned long		Hardware Start address of available physical memory
TransferSize	Unsigned long		Buffer size, must be smaller or equal to the size of the freed / reserved physical memory
Flags	Unsigned long		Defines the acquisition mode

•**Flags:**

Flag	Application
CC_CBG_SINGLE	Fill the supplied buffer once
CC_CBG_CONTINUOUS	The hardware will continuously fill up the buffer, wrapping around to the beginning upon reaching the end of the buffer
CC_CBG_NO_DMA_INTS	Block the hardware from issuing interrupts to the Operating System / Driver. This lets the acquisition run with zero CPU overhead.

Although these Flags are defined as an enumeration, they are meant to be bit-wise OR-ed to make up the Flag-specification in the CC_CaptureBackgroundEx call.

[6]Only supported for PCI-based camera systems

```
enum CC_CAPTURE_BACKGROUND_FLAGS {
                CC_CBG_SINGLE  = 0 ,
                CC_CBG_CONTINUOUS = 0x01 ,
                CC_CBG_NO_DMA_INTS = 0x02
                } ;
```

•**Example code:**

```
see MapPhysMem() example
```

3.9 CC_CaptureData[7]

•**Function:** Acquisition with trigger mode with single image transfer.

•**Related function:** CC_CaptureArm

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureData( HANDLE Cam ,
                                     PVOID Buffer ,
                                     ULONG TransferSize ,
                                     USHORT TimeOut ,
                                     OVERLAPPED * pUserOverlapped ) ;
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Buffer	PVOID	Not applicable	Pointer to array of unsigned char to store the image data.
TransferSize	ULONG	Exact amount of data expected	Size of data to be transferred, depending on image size in pixels and ADC resolution
TimeOut	USHORT	0 = no time-out 1 to 99 = seconds > 100 = 10^{-3} sec.	Time-out period. Started when this function is called, ended when image has been transferred or when the time-out period is exceeded. Only applicable when using LIBUSB driver
pUserOverlapped	OVERLAPPED	NULL	Pointer to an optional OVERLAPPED structure managed by the user application. Not used in USB systems

Error handling:

•The camera handle input is not checked. It must be obtained from CC_Open or CC_OpenEx.
•CC_ERROR_IMAGE_UNDERRUN : The operation has ended, but less data than requested was returned.

[7] Only supported for USB cameras

•**Example code:**

```
CC_CaptureData (cam ,           // camera handle
                buffer ,        // destination buffer
                transfers ,     // amount to be transferred – in bytes
                500 ,           // 500 msec
                NULL            // do not overlap
                );
```

3.10 CC_CaptureSequence[8]

•**Function:** Acquisition with trigger mode and background operation mode.

•**Related function:** CC_CaptureSingle

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureSequence( HANDLE Cam,  PVOID Buffer,
                    ULONG BufferSize, ULONG NumberOfFrames ,ULONG Framesize,
                    ULONG NotificationInterval , void (*CallBack)(void) ,
                    CC_TRIGGER_MODE TriggerMode, USHORT TimeOut );
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Pbuffer	PVOID	Not applicable	Pointer to array of unsigned char to store the image data.
BufferSize	ULONG	Min. 2 times the size of the image	Circular buffer to be filled with images in sequence
NumberOfFrames	LONG	Actual number of images to be taken	(-1) for indefinite sequence
FrameSize	ULONG	Specifies size of each image to be taken	
NotificationInterval	ULONG	Number of frames between callbacks	0 for no callback
CallBack	void (*)(void)	Function pointer NULL if no callback required	Routine to be called whenever the number of frames as specified by the Notification interval has been transferred into the Buffer
TriggerMode	CC_TRIGGER_MODE	Enumeration type	Trigger type. Modes are listed in a type structure, see this paragraph below. Not all modes as defined in CC_CaptureSIngle are valid

[8] Only supported on PCI-based cameras

Variable name	C Type	Range	Purpose
Timeout	USHORT (16-bit)	0 = no time-out 1 to 99 = seconds > 100 = milliseconds	Time-out period. Started when this function is called, ended when image has been transferred or when the time-out period is exceeded.

•**TriggerMode** :

Depending up on the interface and camera option, the camera or the interface can work in several trigger modes. These modes are:

Trigger mode	Application
CC_INTERFACE_TRIGGER_CONTINUOUS	The interface will wait until the interface gets a trigger pulse. The interface will keep responding to sequential trigger pulses send to the camera.
CC_CAMERA_TRIGGER_CONTINUOUS	The camera will wait until the camera gets a trigger pulse. The camera will keep responding to sequential trigger pulses send to the camera. Only applicable in capture continuous mode.
CC_CAMERA_CONTINUOUS	The camera will continuously transmit images until halted by the user application

```
typedef enum {     CC_NO_TRIGGER = 0x0 ,
                   CC_INTERFACE_TRIGGER_SINGLE ,
                   CC_INTERFACE_TRIGGER_CONTINUOUS ,
                   CC_CAMERA_TRIGGER_SINGLE ,
                   CC_CAMERA_TRIGGER_CONTINUOUS ,
                   CC_CAMERA_CONTINUOUS
                   } CC_TRIGGER_MODE ;
```

•**Error handling:**

•The camera handle input is not checked. It must be obtained from CC_Open
•The size of the data to be transferred is not checked. The user is able to download less data than specified in the WOI.
•CC_ERROR_TRIGGER_MODE_NOT_DEFINED: An unknown trigger mode was specified.
•CC_ERROR_TRIGGER_MODE_NOT_APPLICABLE: A trigger has been asked which was not applicable for this camera or interface.
•CC_ERROR_CAPTURE_TIMEOUT: Time-out period has been expired.
•CC_ERROR_CAPTURE_IN_PROGRESS: Acquisition was still in progress.

•**Remarks:**

This call is **not** supported on the **PCI-LVDS** interface (CCi4 LVDS)

The camera must have been opened in CAPTURE_NO_WAIT. However, this is a synchronous call and will not return until:

- the number of frames specified have been transferred
- an error came up
- an indefinite sequence has been halted by CC_Abort()

The optional CallBack routine should take as little processing time as possible, because CC_CaptureSequence waits for the return of the call back before it can queue the next image transfer.

•**Example code:**

```
CC_CaptureSequence( cam,          // load camera location
                    Buffer,          // image data buffer
                    BufferSize,      //
                    -1,              // forever and a day
                    Yheight * Xwidth * 2,  // image size for 10 or 12 bit
                    1,               // notify for every frame
                    MyCallBack,      // call back function
                    CC_INTERFACE_TRIGGER_CONTINUOUS, // trigger mode
                    300            // Time-out period of 300 milli seconds
                    ) ;
```

3.11 CC_CaptureSingle

•**Function:** Acquisition with trigger mode with single image transfer.

•**Related function:** CC_SetWOI, CC_CaptureWait, CC_CaptureArm, CC_CaptureData

•**Function return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureSingle( HANDLE Cam,
                                    PVOID Buffer,
                                    ULONG TransferSize,
                                    CC_TRIGGER_MODE TriggerMode,
                                    USHORT TimeOut,
                                    OVERLAPPED * pUserOverlapped );
```

Variable name	C Type	Range	Purpose
Cam	HANDLE	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Pbuffer	PVOID	Not applicable	Pointer to array of unsigned char to store the image data.
TransferSize	ULONG	Exact amount of data expected	Size of data to be transferred, depending on image size in pixels and ADC resolution
TriggerMode	CC_TRIGGER_ MODE	Enumeration type	Trigger type. Modes are listed in a type structure, see this paragraph below.
Timeout	USHORT (16-bit)	0 = no time-out 1 to 99 = seconds > 100 = 10^{-3} sec.	Time-out period. Started when this function is called, ended when image has been transferred or when the time-out period is exceeded.
pUserOverlapped	OVERLAPPED (windows structure)	See note [1].	Pointer to an optional OVERLAPPED structure managed by the user application.

Note [1] : pUserOverlapped

1. Camera opened with CC_CAPTURE_WAIT argument
    1. The CC_CaptureSingle call will be executed synchronously and will only return at the end of the operation.
    2. Windows then doesn't require this structure. We advise to specify a NULL value.

2. Camera opened with CC_CAPTURE_NO_WAIT argument
    1. The CC_CaptureSingle call will return immediately after the requested operation has been registered inside the Windows IO Manager. The effective operation will be executed asynchronously to the calling thread.
    2. If NULL, no event is signalled (directly) to the user. Use CC_CaptureStatus() or CC_CaptureWait() to detect if capture has finished. CCAPI.DLL will use an internal OVERLAPPED structure to satisfy Windows.
    3. If the user supplies this structure himself, he also must manage it himself. He may e.g. use the windows function WaitForSingleObject to be signalled if done.

•The user can request multiple operations by repeatedly executing this call. Windows IO Manager will queue the operations internally and will serialise the requests to the hardware until this queue becomes empty. He must take proper care in supplying different user-managed OVERLAPPED structures to effectively serialise the operations.

•TriggerMode :

Depending on the interface and camera option, the camera and/or the interface can work in several trigger modes. These modes are:

Trigger mode	Application
No trigger	Camera runs in normal mode. As soon as the capture function is called, the acquisition is started immediately. Only a single image will be captured
Interface trigger single	The interface will wait until the interface gets a trigger pulse. The interface will not respond to other successive triggers.
Camera trigger single	The camera will wait until the camera gets a trigger pulse. The camera will not respond to a successive trigger.

```
typedef enum { CC_NO_TRIGGER = 0x0 ,
               CC_INTERFACE_TRIGGER_SINGLE ,
               CC_INTERFACE_TRIGGER_CONTINUOUS ,
               CC_CAMERA_TRIGGER_SINGLE ,
                           CC_CAMERA_TRIGGER_CONTINUOUS ,
                                    CC_CAMERA_CONTINUOUS
               } CC_TRIGGER_MODE ;
```

•Error handling:

•The camera handle input is not checked. It must be obtained from CC_Open.
•The size of the data to be transferred is not checked. The user is able to download less data than specified in the WOI.
•CC_ERROR_TRIGGER_MODE_NOT_DEFINED: An unknown trigger mode was specified.
•CC_ERROR_TRIGGER_MODE_NOT_APPLICABLE: A trigger has been asked which was not applicable for this camera or interface.
•CC_ERROR_CAPTURE_TIMEOUT: Time-out period has been expired.

- CC_ERROR_CAPTURE_IN_PROGRESS: Acquisition was still in progress.

- **Example code:**

```
CC_CaptureSingle( cam,              // load camera location
                  Buffer,           // image data buffer
                  ImageSize * 2,    // image data transferred (16-bit)
                  CC_INTERFACE_TRIGGER_SINGLE, // trigger mode
                  300,              // Time-out period of 300 milli seconds
                  NULL ) ;          // no overlapped operation
```

3.12 CC_CaptureStatus

- **Function:**
  Function to check the acquisition procedure in the asynchronous mode. The acquisition continues during the call of this function.

- **Return value:**
  - Asynchronous mode:
    - FALSE when the acquisition is still busy or has failed, check with GetLastError to see the status.
    - TRUE if acquisition ended successfully.
  - Synchronous mode:
    Do not use in synchronous mode, status may be incorrect. Use GetLastError after CC_CaptureSingle returns to check the status.

- **Related functions:**
  - CC_CaptureAbort
  - CC_CaptureWait

- **Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureStatus(Handle Cam);
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)

- **Error handling:**
  - CC_ERROR_CAPTURE_TIMEOUT:
    - Not enough data was released by the camera.
    - No camera present or bad cable connection.
    - WOI was inverse or incorrect.
  - CC_ERROR_CAPTURE_NOT_FINISHED:
    - Still waiting for an image as time-out is not yet active.
    - Still waiting for trigger.
  - CC_ERROR_CAPTURE_ABORTED:
    - The function CC_CAPTURE_ABORTED was called before the capture status was checked.

- **Example code:**

```
CC_CaptureStatus(camhandle);
```

3.13 CC_CaptureWait

•**Function:**
Function to wait until the acquisition in the asynchronous mode finished. If the function is used in synchronous mode, the function may wait forever!

•**Return value:**
TRUE when acquisition operation succeeds. FALSE if acquisition fails, check with GetLastError

•**Related functions:**
- •CC_CaptureAbort
- •CC_CaptureStatus

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_CaptureWait(Handle Cam);
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)

•**Error handling:**

•CC_ERROR_CAPTURE_TIMEOUT:
- •Not enough data was released by the camera.
- •No camera present or bad cable connection.
- •WOI was inverse or incorrect.

•CC_ERROR_CAPTURE_ABORTED:
- •The function CC_CAPTURE_ABORTED was called before the capture status was checked.

•**Example code:**

```
CC_CaptureWait(camhandle);
```

3.14 CC_Close

- **Function:**
  To close the device.
  This function must be called before your application ends. After this, no other functions in `CCAPI` can be called.

- **Related function:** *CC_Open* , *CC_OpenEx*

- **Return value:** FALSE when operation was not successful[9]. TRUE if successful.

- **Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_Close (Handle Cam);
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)

- **Error handling**:

  - CC_ERROR_NONE: no error occurred.
  - CC_ERROR_INVALID_HANDLE: handle to close was not valid or it was closed earlier.

- **Example code:**

```
CC_Close(camhandle);                    // camera location
```

[9] Usually when the device was not opened in the first place.

3.15 CC_GetDLLVersion

- **Function:**
  Function to retrieve the version number of CC API-library.

- **Related function:** *CC_GetDriverVersion*.

- **Return value:** FALSE when operation was not successful. TRUE if successful.

- **Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_GetDLLVersion(DWORD * pVersionNumber);
```

Variable	Type	Range	Purpose
pVersionNumber	DWORD *	Not applicable	To receive the DLL version number

- **Error handling**:

  - CC_ERROR_NONE: This is always returned. No error occurred.

- **Format of release:**

  The VersionNumber is filled with the release number, followed by the version number. Returned is a 32-bit Hex format with the upper 16 bits containing the Major Release Number and the lower 16 bits containing the Minor Release Number

  E.g.: 0001 0000 Hex means : Version 1.0

- **Example code:**

```
CC_GetDLLVersion(pVersionNumber);
```

3.16 CC_GetDriverVersion

•**Function:** Function to retrieve the release and version numbers of the CC driver.

•**Related function:** *CC_GetDLLVersion*.

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_GetDriverVersion(Handle Cam ,
                                          DWORD * pVersionNumber);
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Device location
pVersionNumber	DWORD *	Not applicable	To receive the Driver version number

•Error handling: none

•**Format of release:**

Returned is a 32-bit number with the upper 16 bits containing the Major Release Number and the lower 16 bits containing the Minor Release Number

E.g.: 0002 0003 (in Hex) means : Version 2.3

•**Example code:**

```
CC_GetDriverVersion(pVersionNumber);
```

3.17 CC_GetPCIBoardInfo

•**Function:** Function to retrieve the PCI board type and board revision number.

•**Related function: -**

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_GetPCIBoardInfo(Handle Camhandle ,
                                        char * pBoardType ,
                                        char * pBoardRev ) ;
```

Variable	Type	Range	Purpose
Camhandle	Handle	A 32 bit number returned by CC_Open	Device location
pBoardType	char *		To receive the PCI board type
pBoardRev	char *		To receive the PCI board revision number

•Error handling: none

•**Format of returned strings:**

The strings are returned as 2 character strings terminated with a zero character. The PCI board type can be any of the following :
L1 for PCI-LVDS card
CL for PCI-CL card (camera link)
LS for PCI-LS card (serial LVDS)

The first character of the board revision number is always a letter (A .. Z) indicating major update, the second character is always a number (0 .. 9) indicating a minor update.

•**Example code:**

```
CC_GetPCIBoardInfo( CameraNumber , pBoardType , pBoardRev ) ;
```

3.18 CC_GetPushedLastError

•**Function:** Function to retrieve the more error information.

•**Related function:** GetLastError (Windows API), ernno (Linux API)

•**Return value:** Returns a possible native OS Error code, especially when using Class Drivers, like USB of IEEE systems. Returns 0 if no native OS Error code available for the preceding action.

•**Syntax definition:**

```
DLLINOUT int WINAPI CC_GetLastPushedError(Handle Camhandle ) ;
```

Variable	Type	Range	Purpose
Camhandle	Handle	A 32 bit number returned by CC_Open	Device location

•Error handling: none

•**Example code:**

```
CC_GetPushedLastError( Camhandle ) ;
```

3.19 CC_LoadCamera

•**Function:**

Function to load the C-Cam camera with its logic program, before operation. The logic is retrieved from a *.ttb logic file. This operation should be performed whenever a power-down occurred.

! Note that in PCI-systems a CC_LoadCamera is only possible when a valid CC_LoadInterface has been performed first.

! Note that not all cameras need a CC_LoadCamera e.g. the CCf15 is operated on the PCI-LVDS interface itself. If this is the case the function will return without indicating an error.

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_LoadCamera( Handle Cam , char * FileSpec ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
FileSpec	Char *	A valid FileSpec for the *.ttb file name	To locate the logic file. A path + file name may be entered or only a file name. In the last case, the current working directory is searched for the file, if not found; the 'FPGA Logic' directory is searched where the C-Cam package has been installed.

•**Error handling**:

- •CC_ERROR_FUNCTION_NOT_APPLICABLE
  - •Logic not applicable for this kind of camera.
- •CC_ERROR_FILE_NOT_FOUND
  - •File name does not exist.
  - •Wrong directory.
- •CC_ERROR_NOT_ENOUGH_MEMORY
  - •Logic was too large to be stored into the PC's memory
- •CC_ERROR_CAN_NOT_OPEN_FILE
  - •File might be damaged.
  - •File is not a logic file.

•**Example code:**

```
CC_LoadCamera( camhandle , "Cci4c.ttb" ) ;
```

3.20 CC_LoadCameraEx

•Function:

Function to load the C-Cam camera with its logic program, before operation. The logic is retrieved from a *.ttb logic file. This operation should be performed whenever a power-down occurred.

! Note that in PCI-systems a CC_LoadCamera is only possible when a valid CC_LoadInterface has been performed first.

! Note that not all cameras need a CC_LoadCamera e.g. the CCf15 is operated on the PCI-LVDS interface itself. If this is the case the function will return without indicating an error.

This function allows the user to pass the address of a callback function that will be called at various times while CCAPI is performing the actual LoadCamera operation.

•Return value: FALSE when operation was not successful. TRUE if successful.

•Syntax definition:

```
DLLINOUT BOOL WINAPI CC_LoadCameraEx( Handle Cam , char * FileSpec ,
                                      unsigned int Target ,
                                      enum CC_PROGRESS_FLAGS Flags
                                      void * Arg1 , unsigned long Arg2 ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
FileSpec	Char *	A valid FileSpec for the *.ttb file name	To locate the logic file. A path + file name may be entered or only a file name. In the last case, the current working directory is searched for the file, if not found, the 'FPGA Logic' directory is searched where the C-Cam package has been installed.
Target	Unsigned int	0 .. See specific camera manual	Specifies which FPGA in the camera to load, for most cameras this is 0
Flags	enum CC_PROGRESS_FLAGS	enumeration	

Variable	Type	Range	Purpose
Arg1	void *		Call-back function
Arg2	unsigned long		Size indicator. FileSize divided by this Size determines the number of callbacks with block info.

•Flags:

```
enum  CC_PROGRESSFLAGS {
        CC_PROGRESSFLAGS_NONE = 0x0 ,
        CC_PROGRESSFLAGS_CALL_BACK
        } ;
```

	Application
CC_PROGRESSFLAGS_NONE	No extra functionality over CC_LoadCamera()
CC_PROGRESSFLAGS_CALL_BACK	CC_LoadCameraEx will call back with progress codes

•CallBack function:

This function has the following prototype:

void ProgressCallBack(enum PROGRESS_CODE pc , int value) ;

•Progress codes:

```
enum CC_PROGRESSSTATE {
        CC_PROGRESS_INFO_FILEOK = 0 ,
        CC_PROGRESS_INFO_CONFIGOK ,
        CC_PROGRESS_INFO_BLOCKOK ,
        CC_PROGRESS_INFO_ADDRESS ,
        CC_PROGRESS_INFO_ENDED
        } ;
```

	Application
CC_PROGRESS_INFO_FILEOK	Indicated file was found and loaded in memory
CC_PROGRESS_INFO_CONFIGOK	Download of FPGA logic started.
CC_PROGRESS_INFO_BLOCKOK	A block of data was sent to the camera, the integer value indicates the current count of blocks. If this count is 0 the last block was sent.
CC_PROGRESS_INFO_ADDRESS	An address was sent to the camera
CC_PROGRESS_INFO_ENDED	Operation ended successfully. This code is typically sent before successfully returning.

•**Error handling**:

- •CC_ERROR_FUNCTION_NOT_APPLICABLE
  - •Logic not applicable for this kind of camera.
- •CC_ERROR_FILE_NOT_FOUND
  - •File name does not exist.
  - •Wrong directory.
- •CC_ERROR_NOT_ENOUGH_MEMORY
  - •Logic was too large to be stored into the PC's memory
- •CC_ERROR_CAN_NOT_OPEN_FILE
  - •File might be damaged.
  - •File is not a logic file.
- •CC_ERROR_PROGRESS_FLAGS_INVALID
  - •Invalid Progress Info requested

•**Example code:**

```
CC_LoadCameraEx( Camhandle , "bci4-3u20nf.ttb" , 0 ,
                              CC_PROGRESSFLAGS_CALL_BACK ,
                              DisplayProgressBarCB , 4096 ) ;
```

3.21 CC_LoadInterface

•**Function:**

Function to load the C-Cam interface with its logic program, before operation. The logic is retrieved from a *.ttb logic file. This operation should be performed whenever a power-down occurred.

! Note that accessing the PCI interface without a successful CC_LoadInterface will or may hang the PC as the C-Cam interface card will not respond or time-out on the IO operation.

! Note that loading an incompatible logic file to a PCI interface may hang the PC and could provoke a 'latch-up' condition in the hardware possible causing malfunction to the card. If this happens, power off the PC immediately to relieve the 'latch-up' condition.

! Functions returns without error when being called for a non-PCI camera system.

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_LoadInterface( Handle Cam , char * FileSpec ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
FileSpec	Char *	Any *.ttb file name	To locate the logic file. A path + file name may be entered or only a file name. In the last case, the current working directory is searched for the file, if not found, the 'FPGA Logic' directory is searched where the C-Cam package has been installed.

•**Error handling**:

- •CC_ERROR_FUNCTION_NOT_APPLICABLE
  - •Logic not applicable for this kind of camera.
- •CC_ERROR_FILE_NOT_FOUND
  - •File name does not exist.
  - •Wrong directory.
- •CC_ERROR_NOT_ENOUGH_MEMORY
  - •Logic was too large to be stored into the PC's memory
- •CC_ERROR_CAN_NOT_OPEN_FILE
  - •File might be damaged.
  - •File is not a logic file.

•**Example code:**

```
CC_LoadInterface( Camhandle , "CCLVDS.ttb" ) ;
```

3.22 CC_MapPhysMem

•**Function:**

This function is used to map physical memory, reserved from main memory at boot time, into the applications address space in order to be able to access this memory via the standard Windows methods.

•**Related function:** *CC_UnMapPhysMem*

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_MapPhysMem( Handle Cam ,
                                      ULONG PhysicalAddress,
                                    ULONG Size ,
                                    void * * UserAddress ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previously opened device (= camera)
PhysicalAddress	ULONG	Unsigned 32-bit number	Indicates the hardware address of the 'to be mapped' physical memory
Size	ULONG	Unsigned 32-bit number	The length of the block to be mapped
Useraddress	void * *	pointer	The user address where the physical memory can be mapped to for user access.

•**Example code:**

```
// Map a buffer for 3 images (of 1280 * 1024  at 12 bit pixel depth)
// in physical memory on a 128Mbyte system where 8Mbyte have been
// reserved for physical memory access.
// The reservation was done at boot time by adding the /MAXMEM=120
// in the boot.ini file.

#define PHYSMEMSTART     0x07800000
#define PHYSMEMMAXSIZE   0x00800000
#define BUFSIZE          3 * 1280 * 1024 * 2
```

```
Handle camhandle ;
unsigned short * pBuffer ;

ASSERT( BUFSIZE <=  PHYSMEMMAXSIZE) ; // check that it will fit

// open the device
Camhandle = CC_OpenEX( "BCi4 LS" ,  CC_FLAG_NONE , NULL , NULL ) ;
// now can map the physical buffer to a virtual address we can use
CC_MapPhysMem( Camhandle , PHYSMEMSTART , BUFSIZE , &pBuffer) ;
// load interface and camera
CC_LoadInterface(  Camhandle , "pcilshu.ttb" ) ;
CC_LoadCamera(  Camhandle , "bci4-3ls20nf" ) ;
// set everything up using CC_SetParameter() and others
...
// then start the background capture
CC_CaptureBackgroundEx( Camhandle , PHYSMEMSTART , BUFSIZE ,
                                                       CC_CBG_SINGLE) ;


// now images are being acquired in the background




...
// no need to do a CC_CaptureBackgroundAbort() as we didn't ask for
// our buffer to be continuously filed with new images, but rather
// restricted it to a single sweep of 3 images, or i.e. for just BUFSIZE
// data

// free the memory descriptors for the buffer
CC_UnMapPhysMem(  Camhandle , pBuffer ) ;
```

3.23 CC_Open

•**Function:**

When this function is called, it opens a camera with a specific name and assigns specific minimum and maximum parameters to the handle. This identifier must be used to access the device. Nearly all other functions refer to this identifier and make use of the parameter limits.

This function must be used before your application can access the specified camera. After this, other functions in `CCAPI` can be called.

•**Related function:** `CC_Close, CC_OpenEx`

•**Return value:** FALSE when operation was not successful[10]. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_Open( LPCSTR CameraName ,
                              ULONG CameraNumber ,
                        CC_CAPTURE_MODE CaptureMode ) ;
```

Variable	Type	Range	Purpose
CameraName	LPCSTR	List of C-Cam cameras	Camera identification
CameraNumber	ULONG	0 to # of cameras attached to PC	Camera identification
CaptureMode	CC_CAPTURE_MODE	See list below	Acquisition method

•**Camera names:**

To open a camera, a specific camera name is needed. We direct you to the camera's user manual to retrieve the proper name.

e.g.

Camera name	Description
“CCi4 LVDS”	CCi4 camera with PCI-LVDS interface
“CChs13”	CChs13 high speed camera

[10] Usually when the driver was not installed or activated. Also make sure the CCAPI dll / shared object is in the same directory as your software application, or in the appropiate system directory

•**CaptureMode:**

Depending up on the application, the camera is able to work in different acquisition modes. These modes are:

Capture mode	Application
CC_CAPTURE_WAIT	Snapshot mode. The camera captures an image and delivers it to the image buffer. After this action, the camera will wait until a command is given for the next acquisition. The CC_CaptureSingle function returns if the image is in the PC's memory or if an error occurred.
CC_CAPTURE_NO_WAIT	Overlapped operation mode. The camera captures an image and the CC_CaptureSingle function returns immediately returning FALSE. During the acquisition, the program is able to perform other calculations. Acquisition and processing can be performed in parallel. The program can detect when the acquisition is ended. This mode is also necessary to enable use of the CC_CaptureSequence method. This mode is currently only supported for our PCI-card based camera systems.

```
typedef enum { CC_CAPTURE_WAIT = 0x0,
              CC_CAPTURE_NO_WAIT
             } CC_CAPTURE_MODE ;
```

•**Example code:**

```
CC_Open("CCi4 LVDS" , 0 , CC_CAPTURE_NO_WAIT ) ;
```

3.24 CC_OpenEx

•**Function:**

This function is called instead of CC_Open when more functionality is required, especially when using USB cameras.

This function (or CC_Open()) must be used before your application can access any camera. After this, the other functions from the `CCAPI` can be called.

•**Related function:** `CC_Close, CC_Open`

•**Return value:** FALSE when operation was <u>not</u> successful[11]. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_OpenEx( LPCSTR CameraName ,
                              ULONG CameraNumber ,
                        ULONG OpenFlags ,
                        void * Arg1 ,
                        void * Arg2 ) ;
```

Variable	Type	Range	Purpose
CameraName	LPCSTR	List of C-Cam cameras	Camera identification
CameraNumber	ULONG	0 to # of cameras attached to PC	Camera identification
OpenFlags	ULONG	See list below	Defining operation modes
Arg1	void *	-	Either a buffer structure or call-back function
Arg2	void *	-	Currently not used

•**Camera names:**

To open a camera, a specific camera name is needed. We direct you to the camera's user manual to retrieve the proper name.

e.g.

[11] Usually when the driver was not installed or activated. Also make sure the CCAPI dll / shared object is in the same directory as your software application or in the appropriate system directory.

Camera name	Description
“CCi4 LVDS”	CCi4 camera with PCI-LVDS interface
“BCi4-3 USB”	BCi4 USB2.0 camera

•**OpenFlags:**

Depending up on the application, the camera is able to work in different acquisition modes. These modes are:

Flag	Application
CC_FLAG_NONE	Empty flags, implies a CC_CAPTURE_WAIT
CC_FLAG_CAPTURENOWAIT	Overlapped operation mode. The camera captures an image and the CC_CaptureSingle function returns immediately returning FALSE. During the acquisition, the program is able to perform other calculations. Acquisition and processing can be performed in parallel. The program can detect when the acquisition is ended. This mode is also necessary to enable use of the CC_CaptureSequence method. This mode is currently only supported for PCI based systems.
CC_FLAG_VIRTUALCAMERA	Translate camera commands only. Used to control Camera Link based cameras and to obtain Command lists to store in e.g. .ini files. Needs either a buffer structure (void *) or a call-back (void(*)(unsigned short)) in Arg1. This flag is mutually exclusive with CC_FLAG_LOG_CAMERA_COMMANDS
CC_FLAG_VC_CALLBACK	Use callback instead of buffer. Only valid when CC_FLAG_VIRTUALCAMERA has also been specified.
CC_FLAG_USB_LIBWIN	Force use LIBWIN-USB to connect USB Camera in case both drivers (CCUSB.SYS and LIBUSB0.SYS) are active.
CC_FLAG_USB_ENDPOINT0	Force use of ENDPOINT0 to control USB Camera. Has only effect for USB cameras with both In- and OUT- endpoints.
CC_FLAG_USE_UNIQUE_ID	Search for the camera with its Unique ID. This is very effective if there is a mix of cameras connected to the system.
CC_FLAG_USE_ALIAS	Search for the camera using the given Alias. This is an alternate way of recognising cameras in the system.

Flag	Application
CC_FLAG_LOG_CAMERA_COMMANDS	Logs camera control commands into a user supplied buffer. Needs a buffer structure (void *) in Arg1. This flag is mutually exclusive with CC_FLAG_VIRTUALCAMERA
CC_ FLAG_DISABLE_OPENFAILMSGBOX	Disables the pop-up MessageBox when trying to open a non-existing (or not-present) camera.

Although these Flags are defined as an enumeration, they are meant to be bit-wise OR-ed to make up the Flag-specification in the CC_OpenEx call.

```
typedef enum {
        CC_FLAG_NONE = 0x0 ,
        CC_FLAG_CAPTURENOWAIT = 0x01 ,
        CC_FLAG_VIRTUALCAMERA = 0x02 ,
        CC_FLAG_VC_CALLBACK = 0x04 ,
        CC_FLAG_USB_LIBWIN = 0x08 ,
        CC_FLAG_USB_ENDPOINT0 = 0x10 ,
        CC_FLAG_USE_UNIQUE_ID = 0x20 ,
        CC_FLAG_USE_ALIAS = 0x40 ,
        CC_FLAG_LOG_CAMERA_COMMANDS = 0x80 ,
        CC_FLAG_DISABLE_OPENFAILMSGBOX = 0x100
        } CC_OPENFLAGS ;
```

The following structure uses an open array (e.g. buffer[1]) to specify the buffer. This avoids scattering the data over the structure and a separate allocated buffer. See the examples. (Also see the BMP structure as used by Windows.)

```
typedef struct {
      int          idx ; // where we are
      int          nel ; // number of elements
      unsigned short     buffer[1] ;
      } translated_commands ;
```

•**Example code:**

```
#define      BUFENTRIES 128
translated_commands * MyVCBuffer ;
FILE *       fd ;

MyvCBuffer = malloc( sizeof(translated_commands) + BUFENTRIES *
                                          sizeof( unsigned short ) ) ;
MyVCBuffer->idx = 0 ;
MyVCBuffer->nel = BUFENTRIES ;
CC_OpenEx( "BCi4-3 USB", 0, CC_FLAG_VIRTUALCAMERA, MyVCBuffer,  NULL ) ;

... various operations fill up the MyVCBuffer

//saving the buffer
```

```
fd = fopen( "VCCommands.bin", "wb" ) ;
fwrite( MyVCBuffer, sizeof(translated_commands) + MyVCBuffer->idx *
                              sizeof( unsigned short ), 1, fd ) ;
fclose( fd ) ;
```

•**Another example**

```
#define     BUFENTRIES 10000
translated_commands * MTCBuffer ;
FILE *      fd ;

MTCBuffer = calloc( sizeof(translated_commands) + BUFENTRIES *
                                          sizeof( unsigned short ) ) ;
MTCBuffer->idx = 0 ;
MTCBuffer->nel = BUFENTRIES ;
CC_OpenEx( "BCi4-3 USB", 0, CC_FLAG_LOG_CAMERA_COMMANDS  , MTCBuffer,
                                                          NULL ) ;

... various operations fill up the MTCBuffer, if the buffer overflows it
restarts from the first entry. The 'idx' member of the structure tells us
whether it overflowed or not. As we used 'calloc' to allocate the
structure, all elements are cleared (set to '0').
If MTCBuffer->buffer[MTCBuffer->idx] points to a non-zero value there has
been a wraparound
```

3.25 CC_RegisterErrorHandler

•Function:

This function can be used to simplify the use of Error Handling (see description in chapter 3).
The CC_RegisterErrorHandler can only be called after a successful CC_Open call.
Not all error conditions are trapped, e.g. The capture functions returns the standard error code in case of a Time Out and do not invoke the selected Error Handler. An asynchronous call always returns with a Failure, the GetLastError() indicates the actual state of the operation. This situation is also not trapped.

•Related function:

•Return value: FALSE when operation was not successful[12], TRUE if successful.

•Syntax definition:

```
DLLINOUT BOOL WINAPI CC_RegisterErrorHandler( HANDLE Cam,
                                       CC_ERROR_NOTIFICATION Operation ,
                                       void (* CallbackFunction)( int ));
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Operation	CC_ERROR_ NOTIFICATION	Enumeration	What to do when an error occurs
CallbackFunction	Void (*) (int)	Function pointer NULL if no call-back required	Function to be called when an error occurs if activated in operation

•Special type structure:

Depending on the type of error handler, operation can be defined as:

CC_ERROR_ NOTIFICATION	Operation
CC_ERROR_NOTIFICATION_HANDLED_ BY_APPLICATION	If an error occurs, the error will be handled by the application. No action will be done by the error handler function. (Default)
CC_ERROR_NOTIFICATION_HANDLED_ BY_CALLBACK	If an error occurs, the user specified callback function will be called.
CC_ERROR_NOTIFICATION_HANDLED_ BY_API_MESSAGEBOX[13]	If an error occurs, a message box will be displayed with the error code translation.

```
typedef enum { CC_ERROR_NOTIFICATION_HANDLED_BY_APPLICATION = 0 ,
```

[12] Usually when the driver was not installed or activated. Also make sure the CCAPI.dll is in the same directory as your software application.
[13] Not for Linux OS

```
                              CC_ERROR_NOTIFICATION_HANDLED_BY_CALLBACK ,
                              CC_ERROR_NOTIFICATION_HANDLED_BY_API_MESSAGEBOX
                              } CC_ERROR_NOTIFICATION ;
```

•**Example code:**

```
CC_RegisterErrorHandler( Cam ,
                         CC_ERROR_NOTIFICATION_HANDLED_BY_CALLBACK ,
                         MyCallbackFunction ) ;
```

3.26 CC_SetParameter

•**Function:**

This function is to set the camera’s parameters.

•**Related function:** CC_SetWOI , CC_CaptureSingle

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_SetParameter( HANDLE Cam,
                                        CC_PARAMETER Par,
                                        ULONG Value );
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Par	CC_PARAMETER	Enumeration	Parameter type
Value	ULONG	Depending on parameter type checking will be done	Value for parameter Although a ULONG is specified, a number of settings only use the 16 or 8 lower bits of this value.

•**Camera parameters:**

Depending up on the camera type, several parameters can be controlled:

Some of the parameters are only partially explained. Refer to the camera User Manual for specific information.

Parameter	Description
CC_PAR_INTEGRATION_TIME	Exposure time. This is the time between the first reset (= opening of the row) and the readout or closing of this row. Value in µ seconds.
CC_PAR_GAIN	Gain control of the analogue signal before the ADC.
CC_PAR_OFFSET	Offset control of the analogue signal before the ADC.

Parameter	Description
CC_PAR_PIXEL_PRECHARGE	Sensor well depth control. The value controls the reset voltage of the pixels Value: 0...255
CC_PAR_ANAVAL0 CC_PAR_ANAVAL1 CC_PAR_ANAVAL2 CC_PAR_ANAVAL3	Various analogue settings in the camera. 8 bit values. See the specific camera information documents
CC_PAR_ANAVAL01P CC_PAR_ANAVAL23P	16 bit analogue camera settings using combined transfer of 2 successive 8bit values. See the specific camera information documents
CC_PAR_CAMERA_MODE	Camera readout mode. With this parameter, one can toggle between several camera or interface test images and the image from the image sensor. Value: Enumeration (see below).
CC_PAR_DATA_MODE	Output data format. This defines the ADC resolution and the data position of the image. Value: enumeration. See below.
CC_PAR_YSTART	First row of WOI[14]. Value between 0 and sensor pixel height – 1. Value must be smaller or equal than CC_PAR_YEND.
CC_PAR_YINC	Row increment pitch, also known as sub-sample resolution. Value: 1..32.
CC_PAR_YEND	Last row of WOI. Value between 0 and sensor pixel height – 1. Value must be equal or larger than CC_PAR_YSTART.
CC_PAR_XSTART	First column of WOI. Value between 0 and sensor pixel width – 1. Value must be smaller or equal than CC_PAR_XEND.
CC_PAR_XINC	Column increment pitch, also known as sub-sample resolution. Value: 1..32.

[14] WOI = Window of interest. Rectangle part of the sensor area defined by the camera user.

Parameter	Description
CC_PAR_XEND	Last column of WOI. Value between 0 and sensor pixel width – 1. Value must be equal or larger than CC_PAR_XSTART.
CC_PAR_PWM	Brightness control by means of Pulse Width Modulation. Only a part of the dynamic range is visualised. This part is shifted over the whole dynamic range of the sensor[15]. Value: 0..255.
CC_PAR_ADC_DELAY	ADC stabilisation delay
CC_PAR_Y_DELAY	
CC_PAR_X_DELAY1	
CC_PAR_X_DELAY2	
CC_PAR_CORRECTION_MODE	Calibration correction
CC_PAR_CTRLBIT	Sets/Resets Single Control bits in the camera. Uses a 16 bit value that is a combination of 2 8bit values. The upper 8 bits define the bit number (0 to 7) and the lower 8 bits specify whether it is on or off. See applicable camera documents for usage.
CC_PAR_SENSOR_RESET	Reset sensor (or camera) before operation.
CC_PAR_INTERFACE_FLASH_DELAY	
CC_PAR_INTERFACE_FLASH_WIDTH	
CC_PAR_INTERFACE_FLASH_SETTINGS	
CC_PAR_SKIP_XNCS_WAIT	
CC_PAR_OFFSET_B_FINE	
CC_PAR_INTSEL	
CC_PAR_CAMERA_FLASH_DELAY	

[15] In case of a CCf15 camera, 4-decade range is moved over the 6-decade full dynamic range of the sensor.

Parameter	Description
CC_PAR_CAMERA_FLASH_WIDTH	
CC_PAR_CAMERA_TRIGGER_SETTINGS	
CC_PAR_OFFSET_RED	
CC_PAR_OFFSET_GREEN	
CC_PAR_OFFSET_BLUE	
CC_PAR_REPEAT_TIME	
CC_PAR_DLS_OPERATION	
CC_PAR_DLS_SET_PIXEL_PER_LINE	
CC_PAR_DLS_SET_LRATE	
CC_PAR_FRAME_COUNT	Specifies how many frames the camera is to generate. In the case of LineScan mode a selection of a single line equals a frame.
CC_PAR_CAMERA_CONTROL	

[1] = Implemented

[2] = Not yet implemented

[3] = (ADC_DELAY + X_DELAY1) AND (ADC_DELAY + X_DELAY2) $\geq$ 4 , to function properly.

[4] = minimum integration time is the readout time of 2 lines. The readout time of one line is a function of the size of the line. See the respective documentation of the cameras on how to calculate the line time.

•**CC_PAR_DATA_MODE:**

A camera can work internally in 8, 10, 12 mode – or more if specified – depending on its ADC resolution of the camera. An 8-bit ADC camera will operate internally in 8 bit. A camera with 10 or 12 bit ADC resolution is handled by the interface as 16 bit data. Optionally only a part of the camera data can be selected and transferred to the buffer on the PC. The range of data selected for this transfer is to be defined by the Data mode parameter.
Some cameras or interfaces can/will perform a 'ceiling' operation when selecting 8 bits out of 10 or 12 input bits to avoid wrapping around: e.g. a 12 bit value of 301 will be 'ceiled' to 255 when selecting bits 7..0 as output. See the applicable camera documents. Selecting 8 bits out of 10 or 12 input bits effectively performs a 'digital' gain on the image data.

Data mode	Application
CC_DATA_8BIT_7_DOWNTO_0	The interface transfers his data from bit 7 down to bit 0 to the 8 bit buffer on the PC.
CC_DATA_8BIT_8_DOWNTO_1	The interface transfers his data from bit 8 down to bit 1 to the 8-bit buffer on the PC.
CC_DATA_8BIT_9_DOWNTO_2	The interface transfers his data from bit 9 down to bit 2 to the 8 bit buffer on the PC.
CC_DATA_8BIT_10_DOWNTO_3	The interface transfers his data from bit 10 down to bit 3 to the 8-bit buffer on the PC. Only applicable for a 12 bit camera.
CC_DATA_8BIT_11_DOWNTO_4	The interface transfers his data from bit 11 down to bit 4 to the 8-bit buffer on the PC.
CC_DATA_16BIT_9_ DOWNTO_0	The interface transfers his data from bit 9 down to bit 0 to the 16-bit buffer on the PC.
CC_DATA_16BIT_11_DOWNTO_2	The interface transfers his data from bit 11 down to bit 2 to the 16-bit buffer on the PC.
CC_DATA_16BIT_11_DOWNTO_0	The interface transfers his data from bit 11 down to bit 0 to the 16-bit buffer on the PC.

```
typedef enum {CC_DATA_8BIT_7_DOWNTO_0 = 0x0,
              CC_DATA_8BIT_8_DOWNTO_1,
              CC_DATA_8BIT_9_DOWNTO_2,
              CC_DATA_8BIT_11_DOWNTO_4,
              CC_DATA_16BIT_9_DOWNTO_0,
              CC_DATA_16BIT_11_DOWNTO_2,
              CC_DATA_16BIT_11_DOWNTO_0
              CC_DATA_8BIT_10_DOWNTO_3
             } CC_DATA_MODE ;
```

•CC_PAR_CAMERA_MODE:

A camera can also work in different modes. During the process of writing software or testing the camera, it is a help to use a test pattern. Depending up on the camera type, they can come from the camera or interface. Of cause, one can always select the image sensor itself.

Camera mode	Application
CC_CAMERA_NORMAL	Image is coming from the sensor.
CC_CAMERA_DIAG_X	Diagnostic data change in the X direction
CC_CAMERA_DIAG_Y	Diagnostic data change in the Y direction
CC_CAMERA_DIAG_X_XOR_Y	C-Cam test pattern in two directions

```
typedef enum {     CC_CAMERA_NORMAL,
                   CC_CAMERA_DIAG_X,
                   CC_CAMERA_DIAG_Y,
                   CC_CAMERA_DIAG_X_XOR_Y
              } CC_CAMERA_MODE ;
```

•Error handling:

- •CC_ERROR_PARAMETER_NOT_APPLICABLE:
  - •The requested parameter is not applicable for this type of camera.
- •CC_ERROR_PARAMETER_NOT_DEFINED
  - •The requested parameter does not exist
  - •The requested parameter is misspelled.
- •CC_ERROR_DATA_MODE_NOT_DEFINED
  - •The requested data mode does not exist
  - •The requested data mode is misspelled.
- •CC_ERROR_CORRECTION_MODE_NOT_APPLICABLE
  - •This type of camera does not accept correction

- •CC_ERROR_CORRECTION_MODE_NOT_DEFINED
  - •The requested parameter does not exist
  - •The requested parameter is misspelled.
- •CC_ERROR_OUT_OF_RANGE_PAR_3
  - •Check limits of the parameter selected.

•Example code:

```
CC_SetParameter(Cam,
                   CC_DATA_MODE,
                   CC_PIXEL_8BIT_7_DOWNTO_0
               );
```

3.27 CC_SetWOI

•**Function:**

This function sets the Window Of Interest (WOI) for the camera sensor device.

•**Related function:**
- •CC_SetParameter
- •CC_CaptureSingle

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_SetWOI( HANDLE Cam ,
                                USHORT Xpar1 ,
                                USHORT Ypar1 ,
                                USHORT Xpar2 ,
                                USHORT Ypar2 ,
                                USHORT Xinc ,
                                USHORT Yinc ,
                                CC_WOI_MODE WOIMode ,
                                ULONG * FrameSize ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
Xpar1	USHORT	0 to horizontal sensor size – 1	First horizontal pixel reference
Ypar1	USHORT	0 to vertical sensor size – 1	First vertical pixel reference
Xpar2	USHORT	0 to horizontal sensor size – 1	Second horizontal pixel reference
Ypar2	USHORT	0 to vertical sensor size – 1	Second vertical pixel reference
Xinc	USHORT	1 to 32	Horizontal sub-sample resolution[16]
Yinc	USHORT	1 to 32	Vertical sub-sample resolution
WOIMode	CC_WOI_MODE	Enumeration	Reference of WOI

[16] If camera is able to do so, else the increment value defaults to the value 1.

Variable	Type	Range	Purpose
FrameSize	ULONG *	Not applicable	Frame size in pixels as calculated by the API

•**Special type structure:**

Depending up on the application, the Window Of Interest (WOI) can be defined as:

WOI mode	Application	Full frame example (1280x1024)
Left Top / Right Bottom	First and last pixels of the rectangle shaped WOI are given.	0, 0, 1279, 1023
Left Top / Width Height	The first pixel of the WOI is given. Then the two dimensional size of the WOI is given.	0, 0, 1280, 1024
Centre / Width Height	The gravity point of the WOI is given. Then the two dimensional size of the WOI is given.	639, 511, 1280, 1024
Right Bottom / Width Height	The last pixel of the WOI is given. Then the two dimensional size of the WOI is given.	1279,1023,1280,1024

```
typedef enum { CC_WOI_LEFTTOP_RIGHTBOTTOM = 0x0 ,
               CC_WOI_LEFTTOP_WIDTHHEIGHT ,
               CC_WOI_CENTER_WIDTHHEIGHT ,
               CC_WOI_RIGHTBOTTOM_WIDTHHEIGHT
               } CC_WOI_MODE ;
```

•**Example code:**

```
CC_SetWOI(Cam,
             Xstart,Ystart,
             Xend, Yend,
             1, 1,
             CC_WOI_LEFTTOP_RIGHTBOTTOM,
             &FrameSize);
```

3.28 CC_StrError

•**Function:**

This function is returns a pointer to string representation of an error code. If the function cannot map the error code to a CCAPI defined value, it will chain on to strerror() provided by the C-run time library.

•**Related function:** *strerror()*

•**Return value:** pointer to string.

•**Syntax definition:**

```
DLLINOUT char * WINAPI CC_StrError( HANDLE Cam, int errorcode ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera) NULL is accepted to allow decoding of errors when CC_Open() or CC_OpenEx() returns an invalid handle.
errorcode	int		A number returned by a call to GetLastError() or by CC_GetLastPushedError().

•**Example code:**

```
char     * pes ;
char     msg[256] ;

pes = CC_StrError(Cam, GetLastError() ) ;
strcpy( msg , "verbose Error Message : " ) ;
strcat( msg , pes ) ;
MessageBox( NULL , NULL , msg , 0 ) ;
```

3.29 CC_UnMapPhysMem

•**Function:**

This function is used to unmap physical memory, reserved from main memory at boot time, from the applications address space in order to be able to access this memory via the standard Windows methods.

•**Related function:** *CC_MapPhysMem*

•**Return value:** FALSE when operation was not successful. TRUE if successful.

•**Syntax definition:**

```
DLLINOUT BOOL WINAPI CC_UnMapPhysMem(HANDLE Cam,,
                                       void * UserAddress ) ;
```

Variable	Type	Range	Purpose
Cam	Handle	A 32 bit number returned by CC_Open	Handle to previous opened device (= camera)
UserAddress	void **	pointer	The virtual address of the previously mapped data

•**Example code:**

```
// Map a buffer for 3 images (of 1280 * 1024  at 12 bit pixel depth)
// in physical memory on a 128Mbyte system where 8Mbyte have been
// reserved for physical memory access.
// The reservation was done at boot time by adding the /MAXMEM=120
// in the boot.ini file.
#define PHYSMEMSTART     0x07800000
#define PHYSMEMMAXSIZE   0x00800000
#define BUFSIZE          3 * 1280 * 1024 * 2


unsigned short * pBuffer ;

ASSERT( BUFSIZE <=  PHYSMEMMAXSIZE) ; // check that it will fit
CC_MapPhysMem( Camhandle , PHYSMEMSTART , BUFSIZE , &pBuffer) ;

...

CC_UnMapPhysMem(  Camhandle , pBuffer ) ;
```

4 Error handling

Every function in this DLL returns a Boolean value indicating success or failure, except for the CC_Open function that returns a handle.
If a function succeeds, then the return value will be non-zero. If a function fails, the return value will be FALSE.
If the CC_Open function fails, then a handle with value zero will be returned. The application can not call CCAPI functions if CC_Open returned a failure, except the CC_StrError() call which ignores the handle.
To get more information when a function fails, you have to call the Windows function GetLastError(), or check the errno variable in Linux..
This function returns a DWORD value that indicates what went wrong. Below is a summary of all possible Error Codes. If GetLastError() returns a value below 0x1000, then the error code is a WIN32 error code.
When using Microsoft Visual Basic for the main application, you have to use the Err object to find out what went wrong in a failed function. The LastDLLError property of the Err object holds the error code.

4.1 Error codes

When a function is successful, GetLastError returns `CC_ERROR_NONE`.

CC_ERROR_NONE	=	0x0000

When any of the parameters is out of range, the following codes will be generated:

CC_ERROR_OUT_OF_RANGE_PAR_1	=	0x1000
CC_ERROR_OUT_OF_RANGE_PAR_2	=	0x1001
CC_ERROR_OUT_OF_RANGE_PAR_3	=	0x1002
CC_ERROR_OUT_OF_RANGE_PAR_4	=	0x1003
CC_ERROR_OUT_OF_RANGE_PAR_5	=	0x1004
CC_ERROR_OUT_OF_RANGE_PAR_6	=	0x1005
CC_ERROR_OUT_OF_RANGE_PAR_7	=	0x1006
CC_ERROR_OUT_OF_RANGE_PAR_8	=	0x1007
CC_ERROR_WOI_SIZE_NOT_ALLOWED	=	0x1010
CC_ERROR_WOI_MODE_UNKNOWN	=	0x1011

CC_ERROR_FUNCTION_NOT_APPLICABLE	=	0x2000
CC_ERROR_PARAMETER_NOT_APPLICABLE	=	0x2001
CC_ERROR_PARAMETER_NOT_DEFINED	=	0x2002
CC_ERROR_WOI_MODE_NOT_DEFINED	=	0x2003
CC_ERROR_CAMERA_MODE_NOT_DEFINED	=	0x2004
CC_ERROR_CAMERA_MODE_NOT_APPLICABLE	=	0x2005
CC_ERROR_TRIGGER_MODE_NOT_DEFINED	=	0x2006
CC_ERROR_TRIGGER_MODE_NOT_APPLICABLE	=	0x2007
CC_ERROR_CAPTURE_MODE_NOT_DEFINED	=	0x2008
CC_ERROR_CAPTURE_MODE_NOT_APPLICABLE	=	0x2009
CC_ERROR_CORRECTION_MODE_NOT_DEFINED	=	0x200a
CC_ERROR_CORRECTION_MODE_NOT_APPLICABLE	=	0x200b
CC_ERROR_DATA_MODE_NOT_DEFINED	=	0x200c
CC_ERROR_DATA_MODE_NOT_APPLICABLE	=	0x200d
CC_ERROR_OPENEX_FLAGS_INVALID	=	0x200e

The following codes deal with camera management:

CC_ERROR_CAMERA_UNKNOWN	=	0x3000
CC_ERROR_NOT_ENOUGH_MEMORY	=	0x3001
CC_ERROR_DEV_NOT_EXIST	=	0x3002
CC_ERROR_GEN_FAILURE	=	0x3003
CC_ERROR_INVALID_HANDLE	=	0x3004
CC_ERROR_FILE_NOT_FOUND	=	0x3005
CC_ERROR_CAN_NOT_OPEN_FILE	=	0x3006
CC_ERROR_INTERFACE_LOAD_FAILURE	=	0x3007
CC_ERROR_BUFFER_TOO_SMALL	=	0x3008
CC_ERROR_INCOMPATIBLE_DRIVER_VERSION	=	0x3009
CC_ERROR_INCORRECT_PCIBOARD_TYPE	=	0x300a
CC_ERROR_INCORRECT_PCIBOARD_REVISION	=	0x300b
CC_ERROR_NV_WRITE	=	0x300c
CC_ERROR_PCIBOARD_MANDATORY	=	0x300d
CC_ERROR_INCORRECT_LOGIC_FILE	=	0x300e
CC_ERROR_INTERFACE_LOGIC_NOT_FOUND	=	0x300f
CC_ERROR_CAMERA_LOGIC_NOT_FOUND	=	0x3010
CC_ERROR_FUNCTION_NOT_SUPPORTED	=	0x3011
CC_ERROR_INVALID_ARG1	=	0x3012
CC_ERROR_INVALID_ARG2	=	0x3013
CC_ERROR_WRITE_SDF	=	0x3014
CC_ERROR_READ_SDF_POST	=	0x3015
CC_ERROR_READ_SDF	=	0x3016
CC_ERROR_READ_SDF_CNT	=	0x3017
CC_ERROR_INVALID_ARG	=	0x3018
CC_ERROR_PROGRESS_FLAGS_INVALID	=	0x3019
CC_ERROR_LOCK_FAILED	=	0x301a
CC_ERROR_USB_CONFIGURATION	=	0x301b
CC_ERROR_UNKNOWN_PCIBOARD_TYPE	=	0x301c
CC_ERROR_DEV_OPEN_FAILURE	=	0x301d
CC_ERROR_LOAD_TARGET_ERROR	=	0x301e
CC_ERROR_CAMERA_MANAGEMENT_FLAGS_INVALID	=	0x301f
CC_ERROR_CAMERA_MANAGEMENT_ARGS_INVALID	=	0x3020
CC_ERROR_CAMERA_MANAGEMENT_READ	=	0x3021
CC_ERROR_CAMERA_MANAGEMENT_WRITE	=	0x3022

The following codes are generated when using capture functions:

CC_ERROR_CAPTURE_TIMEOUT	=	0x4000
CC_ERROR_CAPTURE_IN_PROGRESS	=	0x4001
CC_ERROR_CAPTURE_NOT_FINISHED	=	0x4002
CC_ERROR_CAPTURE_ABORTED	=	0x4003
CC_ERROR_BUFFER_OVERRUN	=	0x4004
CC_ERROR_INVALID_CAPTURE_MODE	=	0x4005
CC_ERROR_CONTINUOUS_CAPTURE_FAILED	=	0x4006
CC_ERROR_CONTINUOUS_CAPTURE_ABORTED	=	0x4007
CC_ERROR_CAPTURE_ARM_INVALID	=	0x4008
CC_ERROR_IMAGE_UNDERRUN	=	0x4009
CC_ERROR_CAPTURE_INVALID_ARG	=	0x400a

5 Include files for Visual Basic and other programming languages

See corresponding directories in the release file package.

Alphabetical Index