DCAM-API
Function Reference

Build 3601

July 2011

HAMAMATSU

READ BEFORE USE

This document and the software sample codes are internal documents of HPK and are disclosed upon request in order to enable the user to create an application using a HPK digital camera.

This document and the software sample codes are disclosed only for the purpose described above, and do not constitute a license, transfer, or any other entitlement for the owner.

All of risk and result of using software depending on this document remains with the user.

This document may include technical inaccuracies or typographical errors.

HPK does not guarantee any damage arising from such errors or this document.

HPK makes no commitment to update or keep current the information contained in this document.

All brand and product names are trademarks or registered trademarks of their respective owners.

INTRODUCTION

This manual describes the DCAM-API specification used to operate digital cameras manufactured by HAMAMATSU (hereafter referred to simply as “digital cameras”). The DCAM-API software development kit is referred to as the “SDK”. The DCAM-API portion that controls the digital cameras is referred to as the “module”.

The SDK consists of source code for a module and a sample application to show how to access DCAM-API. SDK users are free to use the software in any way they like, such as partially modifying source codes and creating completely separate programs.

This SDK is designed to be particularly easy to understand. For this reason, the number of functions has been limited to the minimum, and function calling formats are written in the C programming language.

An extended function is also defined which advanced integrators to control the additional functionality of a digital camera and/or specific interface can use.

Numeric values appearing in this text may differ depending on the digital camera used to capture images. Numeric values should be regarded simply as guides, and not as exact values.

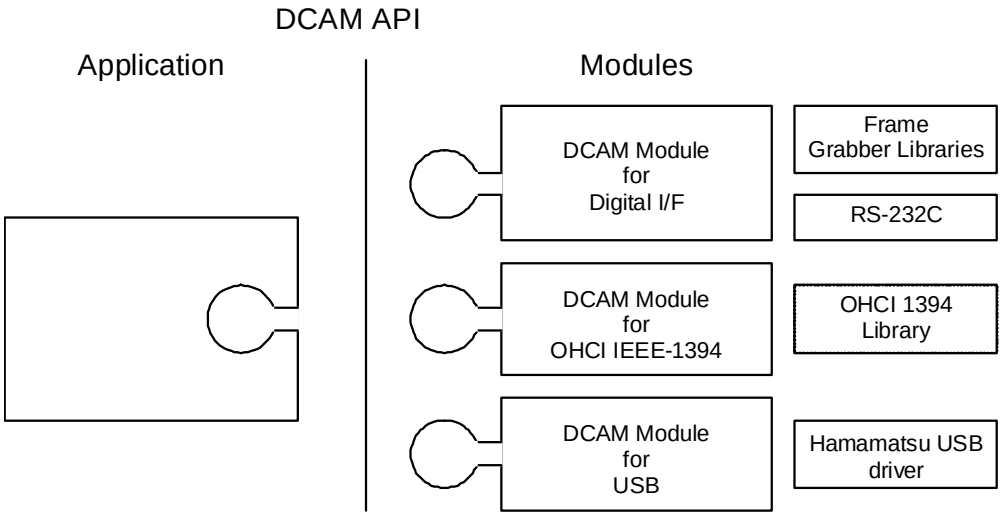
OVERVIEW

Layer structure

Digital cameras can be generally classified by the connections used; some use digital interface connections with a frame grabber, while others may use IEEE-1394 or USB interface connections.

When using a digital interface connection, a serial port is required for sending control commands and a frame grabber for receiving digital data. The user must be able to use these two ports skillfully in order to control the digital camera.

IEEE-1394 and USB connections also require their own appropriate control.



Mechanism

The camera specific interface buses and libraries are suppressed with DCAM-API. You only need to access the DCAM-API layer. The Modules layer is reserved for advanced DCAM-API integration. Modules can be added periodically to your system to give you access to new cameras and new interface technologies without having to recompile your software.

Types of functions

DCAM-API functions can be grouped into a number of types.

- Initialization / termination processing
- Camera information acquisition
- Parameter setting and acquisition
- Capture control
- Accessing of image data and bitmap data
- Extended

The DCAM-API does not contain routines for displaying images. Because a number of methods for displaying images can be envisioned, depending on the application, it is not possible to support all of these through modules. When installing display routines, the camera status is checked and the image refresh timing detected, and images are drawn at that timing. For more detailed information, please see the sample source codes.

Terminology

Capture mode

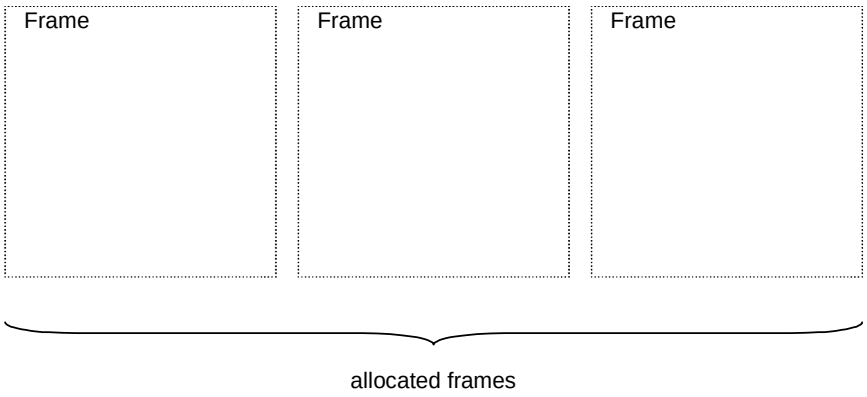
This is the mode in which images are captured by the camera.

Snap	This captures image data. It is used mainly for capturing data for single images.
Sequence	This is used to capture image data continuously.

Image units

Normally, images are two-dimensional, with a vertical and horizontal direction.

Frame	This is one unit used for image data. For one frame, the data for one pixel is aligned from left to right and top to bottom. This is the unit for a series of image data.



Trigger mode

The cameras can capture images with and without synchronization to external signals. We call this option "Trigger mode" and you can change this option with dcam_settriggermode(). We also call the external signals as "External Trigger".

Internal trigger mode	In this mode, the camera does not synchronize with external trigger. Camera runs freely with self timing.
Edge trigger mode	Exposure begins at the timing at which external triggers are switched. A specified time is used for the exposure time.
Level trigger mode	The level of the external triggers is set to a certain period of exposure time.
Software trigger mode	The camera starts capturing when the trigger comes from the host software by the dcam_firetrigger() function. The camera will not accept triggers from other equipment.
TDI trigger mode	Each external trigger shifts image on the sensor vertically one line at a time while reading out one line of image data.
TDI internal trigger mode	This is same as TDI trigger mode except this mode does not require external trigger. Camera runs with self timing.
Start trigger mode	The camera waits for an external trigger to change trigger mode to internal trigger mode and output image.
Synchronous readout trigger mode	The trigger starts the read out of the current exposure and starts a new exposure. The exposure time is the period between two triggers.

External trigger polarity

Positive logic	Exposure begins at the rising edge in Edge mode, and at H level in Level mode.
Negative logic	Exposure begins at the falling edge in Edge mode, and at L level in Level mode.

Camera status

The camera status determines which functions you can call. Some functions can change the camera status. There are four camera statuses as described below.

UNSTABLE	Parameter settings and other functions are called, but are not in the status in which they were set.
STABLE	Parameters and functions are as set, but because no frame memory has been assured, acquisition cannot begin.
READY	Frame memory has been assured and acquisition can be started.
BUSY	Image acquisition is currently being executed.

This page intentionally left almost blank

FUNCTION LIST

```
// Initialization and termination processing
BOOL    dcam_init( void* reserved1, int32* pCount, const char* reserved2 );
BOOL    dcam_getmodelinfo( int32 index, int32 dwStringID, char* buf, _DWORD bytesize );
BOOL    dcam_open( HDCAM* ph, int32 index, const char* reserved );
BOOL    dcam_close( HDCAM h );
BOOL    dcam_uninit( void* reserved1, const char* reserved2 );

// Camera information
BOOL    dcam_getstring( HDCAM h, int32 dwStringID, char* buf, _DWORD bytesize );
BOOL    dcam_getcapability( HDCAM h, _DWORD* pCapability, _DWORD dwCapTypeID );

// Format of transmitted data
BOOL    dcam_getdatatype( HDCAM h, DCAM_DATATYPE* pType );
BOOL    dcam_getbitstype( HDCAM h, DCAM_BITSTYPE* pType );
BOOL    dcam_setdatatype( HDCAM h, DCAM_DATATYPE type );
BOOL    dcam_setbitstype( HDCAM h, DCAM_BITSTYPE type );

// Image size
BOOL    dcam_getdatasize( HDCAM h, SIZE* pSize );
BOOL    dcam_getbitssize( HDCAM h, SIZE* pSize );
BOOL    dcam_getdatasizeex( HDCAM h, DCAM_SIZE* pSize );
BOOL    dcam_getbitssizeex( HDCAM h, DCAM_SIZE* pSize );

// Parameter acquisition
BOOL    dcam_queryupdate( HDCAM h, _DWORD* pFlag, _DWORD reserved );
BOOL    dcam_getbinning( HDCAM h, int32* pBinning );
BOOL    dcam_getexposuretime( HDCAM h, double* pSec );
BOOL    dcam_gettriggermode( HDCAM h, int32* pMode );
BOOL    dcam_gettriggerpolarity( HDCAM h, int32* pPolarity );

// Parameter setting
BOOL    dcam_setbinning( HDCAM h, int32 binning );
BOOL    dcam_setexposuretime( HDCAM h, double sec );
BOOL    dcam_settriggermode( HDCAM h, int32 mode );
BOOL    dcam_settriggerpolarity( HDCAM h, int32 polarity );

// Capture control
BOOL    dcam_precapture( HDCAM h, DCAM_CAPTUREMODE mode );
BOOL    dcam_getdatarange( HDCAM h, int32* pMax, int32* pMin );
BOOL    dcam_getdataframebytes( HDCAM h, _DWORD* pSize );
BOOL    dcam_allocframe( HDCAM h, int32 framecount );
BOOL    dcam_getframecount( HDCAM h, int32* pFrame );
BOOL    dcam_capture( HDCAM h );
BOOL    dcam_firetrigger( HDCAM h );
BOOL    dcam_idle( HDCAM h );
BOOL    dcam_wait( HDCAM h, _DWORD* pCode, _DWORD timeout, HDCAMSIGNALabort );
BOOL    dcam_getstatus( HDCAM h, _DWORD* pStatus );
BOOL    dcam_gettransferinfo( HDCAM h, int32* pNewestFrameIndex, int32* pFrameCount );
BOOL    dcam_freeframe( HDCAM h );

// User Memory Support
BOOL    dcam_attachbuffer( HDCAM h, void** frames, _DWORD size );
BOOL    dcam_releasebuffer( HDCAM h );
```

```
// Data access and LUTs
BOOL    dcam_lockdata( HDCAM h, void** pTop, int32* pRowbytes, int32 frame );
BOOL    dcam_lockbits( HDCAM h, BYTE** pTop, int32* pRowbytes, int32 frame );
BOOL    dcam_unlockdata( HDCAM h );
BOOL    dcam_unlockbits( HDCAM h );
BOOL    dcam_setbitsinputlutrange( HDCAM h, int32 inMax, int32 inMin );
BOOL    dcam_setbitsoutputlutrange( HDCAM h, BYTE outMax, BYTE outMin );

// Extended function
BOOL    dcam_extended( HDCAM h, _ui32 iCmd, void* param, _DWORD size );

// Error information
int32   dcam_getlasterror( HDCAM h, char* buf, _DWORD bytesize );
```

USING THE DCAM-API FROM AN APPLICATION

When a digital camera is being controlled using the DCAM-API, functions should be called using the following procedure.

- Initialize the camera.
- Set the camera parameters.
- Start capturing data.
- Make sure capturing has been completed, and acquire data.
- Carry out the camera termination processing.
- Obtain error information.

Initialization and termination processing

Functions

```
// Initializes the driver
BOOL dcam_init(void* reserved1, int32* pCount, LPCSTR reserved2 );
// Gets camera product information
BOOL    dcam_getmodelinfo( int32 index, int32 dwStringID, char* buf, _DWORD bytesize);
// Initializes camera
BOOL dcam_open( HDCAM* ph, int32 index, LPCSTR reserved);
// Processes camera termination
BOOL dcam_close( HDCAM h);
// Terminates the driver
BOOL dcam_uninit(void* reserved1, LPCSTR reserved2 );
```

Call timing

First, the driver is initialized. When the application installation handle is transferred and initialization has been successfully completed, the number of cameras that can be controlled is obtained.

The camera initialization function is executed when a camera is initialized. This is executed when an application is booted, for instance. Other functions will not work correctly until the initialization function has been executed.

The camera termination processing function is used for closing, when a camera has been held (assured), or resources are being released. This is executed when control of the digital camera is no longer needed, for instance, when the application is exited. When the termination function is called, other functions will not work properly until the initialization function is called again.

Driver initialization

Cameras are initialized using the dcam_init() function. This function initializes a frame grabber or serial port required by the digital camera, and enables control of the digital camera.

Camera product information

Before the dcam_open() function is used, the product information for the camera can be obtained.

DCAM_IDSTR_VENDOR	Vendor information
DCAM_IDSTR_MODEL	Product name
DCAM_IDSTR_BUS	Name of bus being used by camera
DCAM_IDSTR_CAMERAID	Name identifying the camera
DCAM_IDSTR_CAMERAVERSION	Camera version
DCAM_IDSTR_DRIVERVERSION	Driver version
DCAM_IDSTR_MODULEVERSION	Version of DCAM Module
DCAM_IDSTR_DCAMAPIVERSION	Version of DCAM-API the Module supports

Example

DCAM_IDSTR_VENDOR	Hamamatsu
DCAM_IDSTR_MODEL	C4742-95-12NRG
DCAM_IDSTR_BUS	IEEE-1394 OHCI
DCAM_IDSTR_CAMERAID	9X0001
DCAM_IDSTR_CAMERAVERSION	1.00.14
DCAM_IDSTR_DRIVERVERSION	4.00.1998
DCAM_IDSTR_MODULEVERSION	2.1.2.1
DCAM_IDSTR_DCAMAPIVERSION	2.1.2

Camera initialization

Cameras are initialized using the dcam_open() function. This function obtains the necessary camera handle used by other DCAM-API functions.

Termination processing

Termination processing of a camera is carried out using the dcam_close() function. Calling this function releases the frame grabber or serial port being used for the digital camera. After this function has been called, the digital camera can no longer be controlled.

Camera information

Functions

```
// Gets camera information in the form of a character string
BOOL dcam_getstring( HDCAM h, int32 dwStringID, char* buf, _DWORD bytesize);
// Gets functions supported by the camera
BOOL dcam_getcapability( HDCAM h, _DWORD* pCapability, _DWORD dwCapTypeID);
```

Call timing

These functions can be used at any time after the camera has been opened.

Character string information

The “dwStringID” is specified using the dcam_getstring() function, allowing various types of data to be obtained.

DCAM_IDSTR_VENDOR	Vendor information
DCAM_IDSTR_MODEL	Product name
DCAM_IDSTR_BUS	Name of bus being used by camera
DCAM_IDSTR_CAMERAID	Name identifying the camera
DCAM_IDSTR_CAMERAVERSION	Camera version
DCAM_IDSTR_DRIVERVERSION	Driver version
DCAM_IDSTR_MODULEVERSION	Version of DCAM Module
DCAM_IDSTR_DCAMAPIVERSION	Version of DCAM-API the Module supports

Example

DCAM_IDSTR_VENDOR	Hamamatsu
DCAM_IDSTR_MODEL	C4742-95-12NRG
DCAM_IDSTR_BUS	IEEE-1394 OHCI
DCAM_IDSTR_CAMERAID	9X0001
DCAM_IDSTR_CAMERAVERSION	1.00.14
DCAM_IDSTR_DRIVERVERSION	4.00.1998
DCAM_IDSTR_MODULEVERSION	2.1.2.1
DCAM_IDSTR_DCAMAPIVERSION	2.1.2

The ASCII code returns the character string. Any value between 32 (blank) and 126 (~ tilde) can be used.

Capability information

The dcam_getcapability() function can be used to obtain various kinds of information that the camera has.

DCAM_QUERYCAPABILITY_FUNCTIONS	Functions which the camera has
DCAM_QUERYCAPABILITY_DATATYPE	Data formats that can be specified by the camera
DCAM_QUERYCAPABILITY_BITSTYPE	Bitmap format that can be specified by the camera
DCAM_QUERYCAPABILITY_EVENTS	return available events.

When the DCAM_QUERYCAPABILITY_FUNCTIONS are used, the function information that the camera has is obtained.

DCAM_CAPABILITY_BINNING2	2×2 binning possible
DCAM_CAPABILITY_BINNING4	4×4 binning possible
DCAM_CAPABILITY_BINNING8	8×8 binning possible
DCAM_CAPABILITY_TRIGGER_EDGE	External trigger edge possible
DCAM_CAPABILITY_TRIGGER_LEVEL	External trigger level possible
DCAM_CAPABILITY_TRIGGER_POSI	Supports positive polarity for external trigger
DCAM_CAPABILITY_TRIGGER_NEGA	Supports negative polarity for external trigger
DCAM_CAPABILITY_USERMEMORY	Supports direct capturing to user memory.
DCAM_CAPABILITY_TRIGGER_SOFTWARE	Supports software trigger.

Format of transferred data

Function

```
// Obtain current image data type
BOOL dcam_getdatatype( HDCAM h, DCAM_DATATYPE* pType);
// Change image data type
BOOL dcam_setdatatype( HDCAM h, DCAM_DATATYPE pType);

// Obtain current bitmap type
BOOL dcam_getbitstype( HDCAM h, DCAM_BITSTYPE* pType);
// Change bitmap type
BOOL dcam_setbitstype( HDCAM h, DCAM_BITSTYPE pType);
```

Call timing

The bitmap type and image data type are called before other parameters are set, and before LUT settings are entered, to determine the operation mode of the digital camera. Some digital cameras have multiple operation modes, and the parameter values that can be set differ depending on the available modes.

Bitmap type

The following bitmap types can be used with the DCAM-API.

DCAM_BITSTYPE_INDEX8	256-color index color
DCAM_BITSTYPE_INDEX24	24-bit full-color

Image data type

The following types of image data can be used with DCAM-API.

DCAM_DATATYPE_UINT8	8-bit integer type with no sign
DCAM_DATATYPE_UINT16	16-bit integer type with no sign
DCAM_DATATYPE_RGB24	24-bit color
DCAM_DATATYPE_RGB48	48-bit color

Image size

Function

```
// Obtains image data size
BOOL dcam_getdatasize( HDCAM h, SIZE* pSize);

// Obtains bitmap image size
BOOL dcam_getbitssize( HDCAM h, SIZE* pSize);
```

Call timing

The image size is confirmed after the transfer data type and parameters have been set. When data is obtained using the dcam_getdata() function and the dcam_getbits() function, the image size should be obtained using these functions.

Parameter setting and acquisition

Functions

```
// Obtains status change
BOOL dcam_queryupdate( HDCAM h, _DWORD* pFlag, _DWORD dwReserved);;

// Sets parameter
BOOL dcam_setbinning( HDCAM h, int32 binning);
BOOL dcam_setexposuretime( HDCAM h, double sec);
BOOL dcam_settriggermode( HDCAM h, int32 mode);
BOOL dcam_settriggerpolarity( HDCAM h, int32 pol);

// Obtains parameter
BOOL dcam_getbinning( HDCAM h, int32* binning);
BOOL dcam_getexposuretime( HDCAM h, double* sec);
BOOL dcam_gettriggermode( HDCAM h, int32* mode);
BOOL dcam_gettriggerpolarity( HDCAM h, int32* pol);
```

Call timing

Settings and acquisition involving the digital camera are usually carried out before capturing is done and after capturing has been completed. If the setting function is called while data is being captured, the error code DCAMERR_BUSY may be returned in some cases.

Status changes

The user can check to see if the camera status has changed.

DCAM_UPDATE_RESOLUTION	Resolution has changed
DCAM_UPDATE_AREA	Image size has changed
DCAM_UPDATE_DATATYPE	Image data type has changed
DCAM_UPDATE_BITSTYPE	Bitmap data type has changed
DCAM_UPDATE_EXPOSURE	Exposure time has changed
DCAM_UPDATE_TRIGGER	Trigger setting has changed

Items that have changed since the last time they were checked are obtained. When updated information is obtained, the flag is reset, and is not set unless a further change is made.

Binning

This is used to handle multiple pixels as a single pixel. Normally 1, 2, 4, or 8 can be set for this parameter. Values that can be set differ depending on the camera. Be sure to confirm values that can be set using dcam_getcapability().

Exposure time

This specifies the exposure time for the camera. The reciprocal of this value does not necessarily serve as the frame rate. This is because, in addition to the exposure time, there are times when transfer time is required, and when external trigger operation is being used. Also, depending on the camera, there may be times when it is not possible to use the specified exposure time. If a precise exposure time is required, use the acquisition function to acquire the exposure time, which was actually specified.

Trigger mode

The method of synchronization can be specified. External edge triggers and external level triggers can also be used, in addition to internal synchronization.

Trigger polarity

This changes the trigger polarity. Either positive or negative polarity can be specified.

Capture control

Functions

```
// Prepares for capturing
BOOL dcam_precapture( HDCAM h, DCAM_CAPTUREMODE mode);
// Assures acquisition frame and confirms
BOOL dcam_allocframe( HDCAM h, int32 framecount );
BOOL dcam_getframecount( HDCAM h, int32* pFrame);
// Starts acquisition
BOOL dcam_capture( HDCAM h);
// Aborts acquisition
BOOL dcam_idle( HDCAM h);
// Interrupts acquisition and sets system in standby status
BOOL dcam_wait( HDCAM h, _DWORD* pCode, _DWORD timeout, HDCAMSIGNAL abort );
// Gets capturing status
BOOL dcam_getstatus( HDCAM h, _DWORD* pStatus);
BOOL dcam_gettransferinfo( HDCAM h, int32* pNewestFrameIndex, int32* pFrameCount);
// Frees acquired frame
BOOL dcam_freeframe( HDCAM h);
```

Call timing

There are 7 types of capture control functions. First, calling the dcam_precapture() function makes the camera available to capture data. At this point, you will to specify the capture mode. Call the dcam_allocframe() function to allocate a frame buffer. Call dcam_getframecount() function to check the frame count. Use dcam_capture() to begin capturing. This function will return immediately. Use the dcam_wait() function to wait for the next captured image data to become available for processing. To terminate image capturing without waiting for it to end, dcam_idle() function is called. When the image data is no longer needed, calling dcam_freeframe() function releases the frame memory.

Types of capture operation modes

The following types of capture operation modes are available.

Snap	This mode is used for capturing of a specified number of images.
Sequence	This mode is used for continuous capturing of images. Image data is retained for a specified number of images.

Preparation

The dcam_precapture() function is called in order to prepare to capture images. The capture operation mode is specified here. This sets the camera to stable state.

Allocating frames

The dcam_allocframe() function is used to specify the number of image data frames to be allocated for image capture. This sets the camera from stable state to ready state.

The dcam_getframecount() function can be used to confirm the actual number of frames allocated.

Start

When the dcam_precapture() and dcam_allocframe() have been completed successfully, the dcam_capture() function is used to begin capturing images and sets the camera to busy state. This function has no arguments. Processing is carried out based on the capture operation mode specified with the dcam_precapture() function.

Wait

The dcam_wait() function is used to confirm that capturing of image data has been completed. The arguments specify the wait time and the event handle used to abort the capture. Consequently, in a multi-thread environment the wait time can be set to “DCAM_WAIT_INFINITE” and the event handle conveyed.

Status acquisition

The dcam_getstatus() function is used to acquire the camera status. The dcam_gettransferinfo() function is used to obtain the frame number and the number of frames captured since the start of acquisition.

Abort

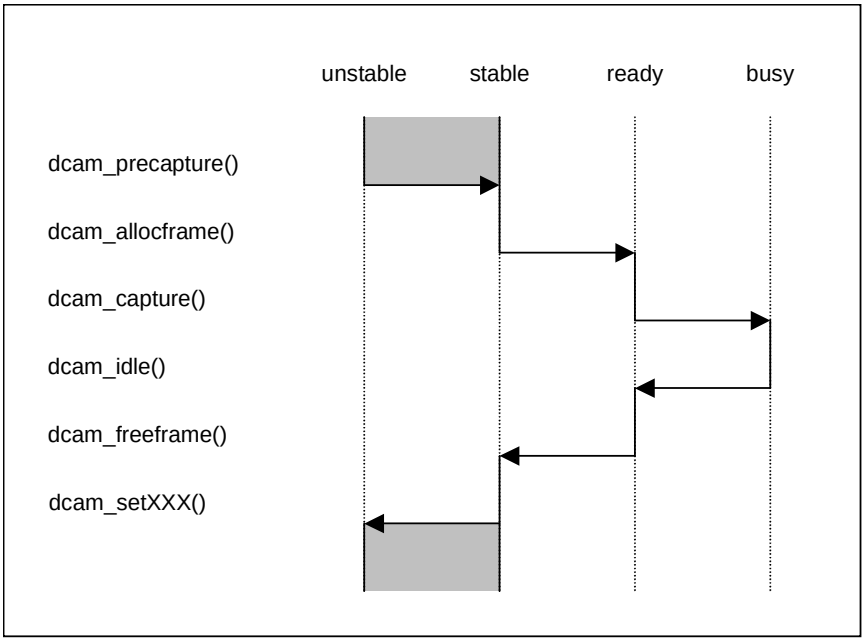
The dcam_idle() function is used to abort acquisition. This sets the camera from busy state to ready state.

Freeing a frame

If access to image data is no longer necessary, the dcam_freeframe() function can be used to free the frame buffer. Image data can no longer be accessed after the buffer has been released. If the dcam_allocframe() function is called before the frame buffer is released, the module assures the buffer once again.

Shifts in camera status

Calling an API changes the camera status as shown below.



User Memory

Functions

```
// User Memory Support
BOOL    dcam_getdataframebytes( HDCAM h, _DWORD* pSize);
BOOL    dcam_attachbuffer( HDCAM h, void** frames, _DWORD size);
BOOL    dcam_releasebuffer( HDCAM h);
```

Call timing

If you want to use the buffer allocated by your application memory, you can use dcam_attachbuffer() function instead of dcam_allocframe(). dcam_attachbuffer() assigns your allocated buffer as the capturing buffer. All image data will be written directly to this buffer. When releasing this buffer, you must use dcam_releasebuffer() instead of dcam_freeframe().

When you use the dcam_attachbuffer() function, dcam_lockdata() cannot be used.

Your application should get the required byte size for each frame by dcam_ getframebytes() function.

Attach Buffer

dcam_attachbuffer() function assigns your application allocated memory for capturing. Your application should inquire the buffer size that the module requires by using dcam_getdataframebytes() function.

Release Buffer

After the end of capturing, the module does not need assignment of user memory. Your application should call dcam_releasebuffer() function to release it.

Data access and LUTs

Functions

```
// Accesses captured data
BOOL dcam_lockdata( HDCAM h, void** pTop, int32* pRowbytes, int32 frame);
BOOL dcam_unlockdata( HDCAM h);

// Accesses bitmap data
BOOL dcam_lockbits( HDCAM h, BYTE** pTop, int32* pRowbytes, int32 frame);
BOOL dcam_unlockbits( HDCAM h);

// Sets LUT for creating bitmap data
BOOL dcam_setbitsinputlutrange( HDCAM h, int32 max, int32 min);
BOOL dcam_setbitsoutputlutrange( HDCAM h, BYTE outMax, BYTE outMin);
```

Call timing

After the dcam_allocframe() function has been called, data can be accessed until the dcam_freeframe() function is used to free the frame. LUT settings can be entered any time after the camera has been opened. With the dcam_lockbits() function, however, the LUT settings in effect at that point are used.

LUT

Many industrial-use monochrome digital cameras have more than 8 bits of output per pixel. For this reason, they cannot display ordinary images without some type of modification. To display the desired range of pixel data in the image, a conversion table called a Look Up Table is used. This enables input of 8 bits or more to be converted to the appropriate value for the image display.

Accessing data for displays

The dcam_lockbits() function can be used to access a specified frame as bitmap data. There may be times when a lock cannot be applied if data is being captured in the Snap or Sequence mode. A lock can be canceled by calling the dcam_unlockbits() function.

Accessing captured data

The dcam_lockdata() function can be used to access the actual captured data, as it is, for a specified frame.

Index color

256-color bitmapping uses index color with a palette. Because this color information is specified by the application, the DCAM module does not know the range to be used. The application uses the dcam_setbitsoutputlutrange() function to specify the index color to be used.

Extended function

Function

```
// Executes an extended function.
BOOL dcam_extended( HDCAM h, _ui32 iCmd, void* param, _DWORD size);
```

Call timing

This can be used after the application has been opened.

Function

The dcam_extended() function allows you to access camera functions that cannot be accessed through the general DCAM-API interface. For detailed information, please refer to the separate reference manual on DCAM-API extended functions.

Error information

Function

```
// Obtains error information
int32       dcam_getlasterror( HDCAM h, char* buf, _DWORD bytesize);
```

Call timing

This can be used after the application has been opened.

Function

This obtains error codes generated by the various cameras.

SAMPLES

Initialization and termination processing

```
int main( int argc, char** argv )
{
	HDCAM h;

	// Initializes a driver
	int	nCamera;
	if( ! dcam_init( NULL, &nCamera))
	{
		// Driver error
		return 0;
	}

	if( nCamera == 0 ))
	{
		// No camera
		return 0;
	}

	// Initializes a camera
	if( ! dcam_open( &h, 0))
	{
		// Camera error
		return 0;
	}

	// Application can use h as HDCAM.

	// Carries out termination processing for a camera
	dcam_close(h);
	return 0;
}
```

Image capturing and data transfer

```
BOOL getdata( HDCAM h, unsigned short* dst, int32 dRow, int frame);

BOOL snap( HDCAM h, unsigned short* buf, int32 rowbytes)
{
	BOOL	ret = FALSE;
	// Prepares for camera capture
	if(	dcam_precapture( h, DCAM_CAPTUREMODE_SNAP)
	 &&	dcam_allocframe( h, 1))
	{
		// Begins capture
		_DWORD		dw = DCAM_EVENT_CYCLEEND;
		if(	dcam_capture( h)
		 &&	dcam_wait( h, &dw, DCAM_WAIT_INFINITE, NULL )
		 &&	getdata( h, buf, rowbytes, 0))
		{
			ret = TRUE;
		}
	}
	return ret;
}

BOOL getdata( HDCAM h, unsigned short* dst, int32 dRow, int frame)
{
	unsigned short* src;
	int32 sRow, row;
	SIZE	sz;

	if( dcam_getdatasize( h, &sz) == FALSE)
		return FALSE;
	if( dcam_lockdata( h, &src, &sRow, 0) == FALSE)
		return FALSE;

	row = min( abs( dRow), abs( sRow));
	for( int y = 0; y < sz.cy; y++) {
		memcpy( dst, src, row);
		src = (unsigned short*)( (char*)src + sRow);
		dst = (unsigned short*)( (char*)dst + dRow);
	}
	return TRUE;
}
```

Use User Memory

```
BOOL snap( HDCAM h, unsigned short** buf, int32 framecount)
{
	_DWORD		bufsize;
	void*	buf[ 3];
	BOOL	ret = FALSE;
	// Prepares for camera capture
	if(	dcam_precapture( h, DCAM_CAPTUREMODE_SNAP)
	 &&	dcam_getdataframebytes( h, & bufsize)) {
		// Allocate user memory
		int i;
		for( i = 0; i < framecount; i++) {
			buf[ i] = malloc( bufsize);
		}

		// Begins capture
		_DWORD		dw = DCAM_EVENT_CYCLEEND;
		if(	dcam_attachbuffer( h, buf, sizeof( *buf) * framecount)
		 &&	dcam_capture( h)
		 &&	dcam_wait( h, &dw, DCAM_WAIT_INFINITE)) {
			ret = TRUE;
		}
		dcam_releasebuffer( h);
	}
	return ret;
}
```

REFERENCE

Types and constants

Error codes

Among the error codes used with the DCAM-API, the following values are defined.

DCAMERR_ABORT	Processing was aborted.
DCAMERR_BUSY	Processing inhibited because of busy status.
DCAMERR_INVALIDHANDLE	Camera handle is invalid.
DCAMERR_INVALIDPARAM	The parameter is invalid.
DCAMERR_NOMEMORY	Insufficient memory
DCAMERR_NOTIMPLEMENT	Function has not been implemented.
DCAMERR_NOTBUSY	Camera is not busy state. Function is available only during busy state.
DCAMERR_NOTREADY	Camera is not ready state.
DCAMERR_NOTSTABLE	Camera is not stable state.
DCAMERR_NOTSUPPORT	Message ID is understood, but is not supported by this driver.
DCAMERR_TIMEOUT	Function returns by timeout.
DCAMERR_UNKNOWNBITSTYPE	Unknown bit transfer type ID.
DCAMERR_UNKNOWNDATATYPE	Unknown data transfer type ID.
DCAMERR_UNKNOWNMSGID	Unknown message ID.
DCAMERR_UNKNOWNPARAMID	Unknown parameter ID.
DCAMERR_UNKNOWNSTRID	Unknown character string ID.
DCAMERR_UNREACH	This routine may not be called. This is internal error.
DCAMERR_FAILOPEN	Error occurred when attempt was made to open camera.
DCAMERR_FAILOPENBUS	Error occurred when attempt was made to open bus.
DCAMERR_FAILOPENCAMERA	Error occurred when attempt was made to open camera.
DCAMERR_FAILREADCAMERA	Error occurred when attempt was made to access camera for reading.
DCAMERR_FAILWRITECAMERA	Error occurred when attempt was made to access camera for writing.
DCAMERR_NOCAMERA	Camera does not exist.
DCAMERR_NODRIVER	Driver does not exist.
DCAMERR_NOMODULE	Module for driving camera does not exist.
DCAMERR_NORESOURCE	I/O resource is not sufficient (other than available memory or hard disk capacity).
DCAMERR_UNKNOWNCAMERA	Unknown camera was found, but is not supported.
DCAMERR_UNSTABLE	The camera status has not stabilized.

DCAM_DATATYPE

These are pixel data types used in DCAM-API, and the following numeric values are defined.

DCAM_DATATYPE_NONE	Used if “DATATYPE” has not been specified.
DCAM_DATATYPE_UINT8	Unsigned 8-bit integer
DCAM_DATATYPE_UINT16	Unsigned 16-bit integer
DCAM_DATATYPE_RGB24	24-bit RGB color
DCAM_DATATYPE_RGB48	48-bit RGB color

DCAM_BITSTYPE

This is the pixel shape for bitmap data, for which the following values are defined. These are used with dcam_getbits().

DCAM_BITSTYPE_INDEX8	8-bit index
DCAM_BITSTYPE_RGB24	24-bit RGB

DCAM_QUERYCAPABILITY_xxx

This is the type when the camera is asked the function and data type.

DCAM_QUERYCAPABILITY_FUNCTIONS	asks the available functions
DCAM_QUERYCAPABILITY_DATATYPE	asks the available data type
DCAM_QUERYCAPABILITY_BITSTYPE	asks the available bitmap bits type

DCAM_CAPABILITY_xxx

This is a graph that shows the functions supported by the camera.

DCAM_CAPABILITY_BINNING2	Supports 2 x 2 binning.
DCAM_CAPABILITY_BINNING4	Supports 4 x 4 binning.
DCAM_CAPABILITY_BINNING8	Supports 8 x 8 binning.
DCAM_CAPABILITY_TRIGGER_EDGE	Supports external trigger edge mode.
DCAM_CAPABILITY_TRIGGER_LEVEL	Supports external trigger level mode.
DCAM_CAPABILITY_TRIGGER_POSI	Supports positive polarity for external trigger.
DCAM_CAPABILITY_TRIGGER_NEGA	Supports negative polarity for external trigger.
DCAM_CAPABILITY_USERMEMORY	Supports direct capturing to user memory.
DCAM_CAPABILITY_TRIGGER_SOFTWARE	Supports software trigger.

DCAM_FRAMECOUNT_xxx

This is used when specifying the number of frames when capturing images.

DCAM_FRAMECOUNT_MAX	Holds as many frames as possible

DCAM_STATUS_xxx

This is indicates the camera status.

DCAM_STATUS_BUSY	Image capture is in progress.
DCAM_STATUS_READY	Image capture is enabled.
DCAM_STATUS_STABLE	Camera settings can be entered.
DCAM_STATUS_UNSTABLE	Camera settings cannot be entered.

DCAM_EVENT_xxx

This indicates an event that occurred in the camera.

DCAM_EVENT_FRAMEBEGIN	Camera output has finished and Module just starts to record.
DCAM_EVENT_FRAMEEND	Frame capture has been completed.
DCAM_EVENT_CYCLEEND	Cycle capture has been completed.
DCAM_EVENT_VVALIDBEGIN	Camera just starts data output.
DCAM_EVENT_CAPTUREEND	Camera stops capturing. This happens when dcam_idle() is called or automatically stops after capturing all images by DCAM_CAPTUREMODE_SNAP.

All DCAM modules support DCAM_EVENT_FRAMEEND, DCAM_EVENT_CYCLEEND and DCAM_EVENT_CAPTUREEND.

DCAM_UPDATE_xxx

This indicates changes in the camera settings. When the application has called a general function, there may be cases when changes are not recognized, such as with some extended functions. When dcam_precapture() is used, the camera status is assured. After that, the dcam_queryupdate() function can be used to check the changed settings.

DCAM_UPDATE_RESOLUTION	The resolution changed. Use dcam_getbinning() to confirm the binning value.
DCAM_UPDATE_AREA	The image size changed. Use dcam_getdatasize() and dcam_getbitssize() to confirm the data size.
DCAM_UPDATE_DATATYPE	The image data type changed. Use dcam_getdatatype() to confirm the image data type.
DCAM_UPDATE_BITSTYPE	The bitmap type changed. Use dcam_getbitstype() to confirm the bitmap type.
DCAM_UPDATE_EXPOSURE	The exposure time changed. Use dcam_getexposuretime() to confirm the exposure time.
DCAM_UPDATE_TRIGGER	The trigger status changed. Use dcam_gettriggermode()and dcam_gettriggerpolarity() to confirm the trigger status.

Functions

```
BOOL    dcam_allocframe( HDCAM h, int32 framecount );
BOOL    dcam_attachbuffer( HDCAM h, void** pTop, _DWORD size);
BOOL    dcam_capture( HDCAM h);
BOOL    dcam_close( HDCAM h);
BOOL    dcam_extended( HDCAM h, _ui32 iCmd, void* param, _DWORD size);
BOOL    dcam_firetrigger( HDCAM h);
BOOL    dcam_freeframe( HDCAM h);
BOOL    dcam_getstatus (HDCAM h, _DWORD pStatus *);
BOOL    dcam_getbinning( HDCAM h, int32* pBinning);
BOOL    dcam_getbitssize( HDCAM h, SIZE* pSize);
BOOL    dcam_getbitstype( HDCAM h, DCAM_BITSTYPE*pType)
BOOL    dcam_getcapability( HDCAM h, _DWORD* dwCapability, _DWORD dwCapTypeID);
BOOL    dcam_getdataframebytes( HDCAM h, _DWORD* pSize);
BOOL    dcam_getdatarange( HDCAM h, int32* pMax, int32* pMin);
BOOL    dcam_getdatasize( HDCAM h, SIZE* pSize);
BOOL    dcam_getdatatype( HDCAM h, DCAM_DATATYPE* pType);
BOOL    dcam_getexposuretime( HDCAM h, double* pSec);
BOOL    dcam_getframecount( HDCAM h, int32* pFrame);
int32   dcam_getlasterror( HDCAM h, char* buf, _DWORD bytesize);
BOOL    dcam_getmodelinfo( int32 index, int32 dwStringID, char* buf, _DWORD bytesize);
BOOL    dcam_getstring( HDCAM h, int32 dwStringID, char* buf, _DWORD bytesize);
BOOL    dcam_gettransferinfo( HDCAM h, int32* pNewestFrameIndex, int32* pFrameCount);
BOOL    dcam_gettriggermode( HDCAM h, int32* pMode);
BOOL    dcam_gettriggerpolarity( HDCAM h, int32* pPolarity);
BOOL    dcam_idle( HDCAM h);
BOOL    dcam_init(void* reserved1, int32* pCount, LPCSTR reserved2 );
BOOL    dcam_lockbits( HDCAM h, BYTE** pTop, int32* pRowbytes, int32 frame);
BOOL    dcam_lockdata( HDCAM h, void** pTop, int32* pRowbytes, int32 frame);
BOOL    dcam_open( HDCAM* ph, int32 index, LPCSTR reserved);
BOOL    dcam_precapture( HDCAM h, DCAM_CAPTUREMODE mode);
BOOL    dcam_queryupdate( HDCAM h, _DWORD* pFlag, _DWORD dwReserved);
BOOL    dcam_releasebuffer( HDCAM h);
BOOL    dcam_setbinning( HDCAM h, int32 binning);
BOOL    dcam_setbitstype( HDCAM h, DCAM_BITSTYPE type),
BOOL    dcam_setdatatype( HDCAM h, DCAM_DATATYPE type);
BOOL    dcam_setexposuretime( HDCAM h, double sec);
BOOL    dcam_setbitsinputlutrange( HDCAM h, int32 inMax, int32 inMin);
BOOL    dcam_setbitsoutputlutrange( HDCAM h, BYTE outMax, BYTE outMin);
BOOL    dcam_settriggermode( HDCAM h, int32 mode);
BOOL    dcam_settriggerpolarity( HDCAM h, int32 polarity);
BOOL    dcam_unlockbits( HDCAM h);
BOOL    dcam_unlockdata( HDCAM h);
BOOL    dcam_uninit(void* reserved1, LPCSTR reserved2 );
BOOL    dcam_wait( HDCAM h, _DWORD*pCode, _DWORD timeout, HDCAMSIGNAL abort );
```

dcam_allocframe()

Usage

DCAM-Module allocates image buffers for capturing.

Declaration

```
BOOL    dcam_allocframe( HDCAM h, int32 frame);
```

Argument(s)

HDCAM h;	specifies the camera.
int32 framecount;	is number of frames to allocate.

Error value

DCAMERR_BUSY	Camera is capturing now.
DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOMEMORY	Insufficient memory.
DCAMERR_NOTSTABLE	Camera is not stable state.

Explanation

When the application calls this function, the module allocates the necessary internal buffer for image acquisition. Capturing does not start at this time. To start acquisition, the application has to call the dcam_capture() function. If the buffer is no longer necessary, the application should call the dcam_freeframe() to release the internal buffer.

The application can call this function again before calling the dcam_freeframe() function. Any memory location received from dcam_lockdata() will be invalid.

If capturing has already started or preparation has not done, this function returns FALSE.

Reference

dcam_capture, dcam_idle, dcam_freeframe, dcam_wait, dcam_getstatus, dcam_getframecount

dcam_attachbuffer ()

Usage

DCAM Module assigns user specified memory as image capturing buffer

Declaration

```
BOOL    dcam_attachbuffer( HDCAM h, void** top, _DWORD bytesize);
```

Argument(s)

HDCAM h;	specifies the camera.
void** top;	is the array of pointer to buffer.
_DWORD bytesize;	is size of top parameter in bytes.

Error value

DCAMERR_BUSY	Camera is capturing now.
DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOMEMORY	Insufficient memory
DCAMERR_NOTSTABLE	Camera is not stable state.

Explanation

This function sets the application allocated memory as capturing buffer. DCAM-Module will capture directly from camera to these memory.

This function is available after calling the dcam_precapture() function and before calling the dcam_capture() function, and the dcam_allocbuffer() function is exclusive.

The application can get the required buffer size by using then dcam_getdataframebytes() function. DCAM does not verify if the frame buffer pointers are valid. System may hang up if a wrong address is used.

If the buffer is no longer necessary, the application should call the dcam_releasebuffer() to release the buffer from DCAM.

Reference

dcam_capture, dcam_idle, dcam_releasecapture, dcam_getdataframebytes

dcam_capture()

Usage
Start capturing images.

Declaration

```
BOOL    dcam_capture( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_BUSY	Camera is already capturing.
DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOTREADY	Camera is not ready state.

Explanation
This function should be called in ready state when the capture mode has been determined and frame memory has been allocated.

The capturing mode is specified at the dcam_precapture() function.

If the mode is the Sequence mode, the camera captures the images repeatedly. When more frames are captured than have been allocated, DCAM will loop back to the start of the buffer.

If the mode is the Snap mode, DCAM will go into the idle state after capturing the number of prepared buffer.

When this function is called in the idle status, capturing is resumed.

Reference
dcam_precapture, dcam_idle, dcam_freeframe, dcam_wait, dcam_getstatus

dcam_close()

Usage

Terminate and close a camera.

Declaration

```
BOOL    dcam_close(HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.

Explanation

This function terminates camera processing and closes camera handle. Once this function is called, the camera handle can no longer be used.

DCAM will forcibly terminate any active capture session and release any image buffers that have not been freed.

Reference

dcam_open

dcam_extended()

Usage
Use an extended function.

Declaration

```
BOOL    dcam_extended( HDCAM h, UINT iCmd, LPVOID param, _DWORD size);
```

Argument(s)

HDCAM h;	specifies the camera.
_ui32 iCmd;	specifies the extended function to be executed.
void* param;	is parameter for iCmd function, if necessary.
_DWORD size;	is size of the parameter, if param is necessary,

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	Invalid parameter
DCAMERR_UNKNOWNMSGID	This module does not support the command id.

Explanation
Camera functions that cannot be accessed with the general functions described in this manual can be accessed using this extended function. For detailed information, please refer to the separate Reference manual on DCAM-API extended functions.

_ui32 is unsigned int32.

Reference

dcam_firetrigger()

Usage

Fire software trigger.

Declaration

```
BOOL    dcam_firetrigger( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_NOTBUSY	Camera is not capturing.

Explanation

This function fires external trigger by software. This function is not available on all cameras. The application should check if this is available by using the dcam_getcapability() function.

This function can only be used while in busy state and trigger mode has been set to DCAM_TRIGMODE_SOFTWARE. If called while not in busy state, this function returns FALSE with an error value of DCAMERR_NOTBUSY.

Reference

dcam_getcapability, dcam_settriggermode, dcam_getstatus

dcam_freeframe()

Usage

Frees the image buffer.

Declaration

```
BOOL    dcam_freeframe( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_BUSY	Camera is capturing now.

Explanation

DCAM releases the image buffer allocated in the dcam_allocframe() function.

If capturing is in progress, this function returns FALSE with an error value of DCAMERR_BUSY to notify the user that the camera is in busy status.

Reference

dcam_precapture, dcam_capture, dcam_idle, dcam_wait, dcam_getstatus, dcam_getframecount

dcam_getbinning()

Usage

Get current binning mode.

Declaration

```
BOOL    dcam_getbinning( HDCAM h, int32* pBinning);
```

Argument(s)

HDCAM h;	specifies the camera.
int32* pBinning;	the pointer to receive the number of binnings.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOTSUPPORT	The camera does not support binning.

Explanation

DCAM returns the current binning mode of the camera.

Reference

dcam_setbinning

dcam_getbitssize()

Usage
Get the width and height of bitmap bits.

Declaration

```
BOOL    dcam_getbitssize( HDCAM h, SIZE* pSize);
```

Argument(s)

HDCAM h;	specifies the camera.
SIZE* pSize;	is the pointer to receive the bitmap width and height.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pSize is NULL

Explanation
DCAM has the function to make bitmap bits. This function returns the width and height of bitmap bits in pixels. Please refer to the dcam_lockbits() function.

Changing some camera parameters may affect the bitmap size. For example, the dcam_setbinning() function can change the bitmap size.

Reference
dcam_setbinning, dcam_getdatasize

dcam_getbitstype()

Usage

Get the current bitmap bits type.

Declaration

```
BOOL    dcam_getbitstype( HDCAM h, DCAM_BITSTYPE* pType);
```

Argument(s)s

HDCAM h;	specifies the camera.
DCAM_BITSTYPE* pType;	is the pointer to receive the bitmap bits type.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pType is NULL

Explanation

DCAM returns the current bitmap bits type of the camera.

Reference

dcam_setbitstype, dcam_getdatatype

dcam_getcapability()

Usage
Get the camera capability.

Declaration

```
BOOL    dcam_getcapability( HDCAM h, _DWORD* pdwCapability, _DWORD dwCapTypeID);
```

Argument(s)

HDCAM h;	specifies the camera.
_DWORD*pdwCapability;	is pointer to _DWORD for camera capability.
_DWORD dwCapTypeID;	specifies the type about which the camera is to be queried.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pdwCapability is NULL
DCAMERR_UNKNOWNPARAMID	dwCapTypeID is not supported by this module.

Explanation
You can choose one of following values for dwCapTypeID parameter to get specified information.

DCAM_QUERYCAPABILITY_FUNCTIONS	General capabilities of the camera
DCAM_QUERYCAPABILITY_DATATYPE	Data types that can be set to the camera
DCAM_QUERYCAPABILITY_BITSTYPE	Bitmap types that can be set to the camera

Reference
dcam_setbinning, dcam_getbinning, dcam_settriggermode, dcam_gettriggermode, dcam_gettriggerpolarity, dcam_settriggerpolarity

dcam_getdataframebytes()

Usage

Get the current frame buffer byte size.

Declaration

```
BOOL    dcam_getdataframebytes( HDCAM h, _DWORD* pSize);
```

Argument(s)s

HDCAM h;	specifies the camera.
_DWORD* pSize;	is pointer to _DWORD for byte per frame

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pSize is NULL

Explanation

This function returns the byte size per frame in the pSize parameter.

Reference

dcam_setbinning, dcam_getbitssize

dcam_getdatarange()

Usage

Get the current data range.

Declaration

```
BOOL    dcam_getdatarange( HDCAM h, int32* pMax, int32* pMin);
```

Argument(s)

HDCAM h;	specifies the camera.
int32* pMax;	is Pointer to int32 for maximum value of the camera output
int32* pMin;	is Pointer to int32 for the minimum value for the camera output

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pMax or pMin is NULL.

Explanation

This function returns the camera output range into the pMax and pMin parameters. The values returned the maximum and minimum possible values of the output data of the camera in the current settings. These values may not represent the maximum and minimum values of current image data.

Reference

dcam_setbitsinputrange

dcam_getdatasize()

Usage

Get the width and height of the image data.

Declaration

```
BOOL    dcam_getdatasize( HDCAM h, SIZE* pSize);
```

Argument(s)s

HDCAM h;	specifies the camera.
SIZE* pSize;	is the pointer to receive the width and height of the image data.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pSize is NULL

Explanation

This function returns the data size of the camera in the current settings in pixels.

Changing some camera parameters may affect the data size. For example, the dcam_setbinning() function can change the data size.

Reference

dcam_setbinning, dcam_getbitssize

dcam_getdatatype()

Usage

Get the current image data type.

Declaration

```
BOOL    dcam_getdatatype( HDCAM h, DCAM_DATATYPE* pType);
```

Argument(s)s

HDCAM h;	specifies the camera.
DCAM_DATATYPE* pType;	is the pointer to receive the image type.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pType is NULL

Explanation

This function returns the current data type of the camera.

Reference

dcam_setdatatype, dcam_getbitstype

dcam_getexposuretime()

Usage

Get the current exposure time.

Declaration

```
BOOL    dcam_getexposuretime( HDCAM h, double* pSec);
```

Argument(s)

HDCAM h;	specifies the camera.
double* pSec;	is the pointer to get current exposure time.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pSec is NULL
DCAMERR_NOTSUPPORT	The camera is not supported.

Explanation

This function returns the current exposure time of the camera.

Reference

dcam_setexposuretime

dcam_getframecount()

Usage
Get the number of prepared frames.

Declaration

```
BOOL    dcam_getframecount( HDCAM h, int32* pCount);
```

Argument(s)

HDCAM h;	specifies the camera.
int32* pCount;	is the pointer to get the number of prepared frames.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pCount is NULL
DCAMERR_NOMEMORY	No frames are reserved.

Explanation
This function returns the number of prepared frames. The prepared frames may refer to the frames allocated with dcam_allocframe() or the buffers attached with dcam_attachbuffer().

Reference
dcam_allocframe

dcam_getlasterror()

Usage

Get the last error code of the camera in current thread.

Declaration

```
int32      dcam_getlasterror( HDCAM h, char* buf = 0, _DWORD bytesize = 0);
```

Argument(s)

HDCAM h;	specifies the camera.
char* buf ;	is pointer of the buffer to receive the character string information. Option.
int32 bytesize;	is the size of the buffer to receive the character string information. Option.

Returned value

The last error code occurred in current thread.

Explanation

This function will return the error code of the last DCAM function in the thread that had failed. DCAM is capable of providing the last DCAM error of each thread used in the application.

dcam_getmodelinfo()

Usage

Get the camera information with camera index.

Declaration

```
BOOL    dcam_getmodelinfo( int32 index, int32 dwStringID, char* buf, _DWORD bytesize);
```

Argument(s)

int32 index;	specifies the camera by index number.
int32 dwStringID;	is the index of information.
char* buf;	is pointer to receive character strings.
_DWORD bytesize	is the size of the receive buffer.

Returned values

If the return value is FALSE, this function is failed cause of:

index is wrong

dwStringID value is unsupported.

buf is NULL.

Explanation

This function provides camera information before opening the camera. You can choose following values for dwStringID.

DCAM_IDSTR_VENDOR	Vendor information
DCAM_IDSTR_MODEL	Product name
DCAM_IDSTR_BUS	Name of bus being used by camera
DCAM_IDSTR_CAMERAID	Name identifying the camera
DCAM_IDSTR_CAMERAVERSION	Camera version
DCAM_IDSTR_DRIVERVERSION	Driver version

If the camera is already opened, the dcam_getstring() function can be used instead.

“\0” is appended at the end of the character string even if the character string information is longer than specified buffer size.

Reference

dcam_open, dcam_getstring

dcam_getstatus()

Usage

Get the camera status.

Declaration

```
BOOL dcam_getstatus( HDCAM h, _DWORD* pStatus);
```

Argument(s)

HDCAM h;	specifies the camera.
_DWORD* pStatus;	is pointer to receive camera status.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pStatus is NULL

Explanation

Get the current status of the camera.

Status can be one of following.

DCAM_STATUS_BUSY	Image capturing is in progress.
DCAM_STATUS_READY	Image capturing is enabled.
DCAM_STATUS_STABLE	Camera settings have been entered.
DCAM_STATUS_UNSTABLE	Camera settings have not been entered.

Reference

dcam_precapture, dcam_capture, dcam_wait, dcam_idle

dcam_getstring()

Usage

Get the camera information with camera handle.

Declaration

```
BOOL    dcam_getstring( HDCAM h, int32 dwStringID, char* buf, _DWORD bytesize);
```

Argument(s)s

HDCAM h;	specifies the camera.
int32 dwStringID;	is the index of information.
char* buf;	is pointer to receive character strings.
_DWORD bytesize	is the size of the receive buffer.

Returned values

If the return value is FALSE, the dwStringID value is unsupported.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	buf is NULL
DCAMERR_UNKNOWNSTRID	An invalid character string number has been specified.

Explanation

This function is similar to dcam_getmodelinfo(), however this function provides camera information only after the camera has been opened. You can choose following values for dwStringID.

DCAM_IDSTR_VENDOR	Vendor information
DCAM_IDSTR_MODEL	Product name
DCAM_IDSTR_BUS	Name of bus being used by camera
DCAM_IDSTR_CAMERAID	Name identifying the camera
DCAM_IDSTR_CAMERAVERSION	Camera version
DCAM_IDSTR_DRIVERVERSION	Driver version

To get this information before opening the camera, use the dcam_getmodelinfo() function.

“\0” is appended at the end of the character string even if the character string information is longer than specified buffer size.

Reference

dcam_open, dcam_getmodelinfo

dcam_gettransferinfo()

Usage

Get the information of capturing.

Declaration

```
BOOL dcam_gettransferinfo( HDCAM h, int32* pNewestFrameIndex, int32* pFrameCount = 0);
```

Argument(s)

HDCAM h;	specifies the camera.
int32* pNewestFrameIndex;	is pointer to receive the number of the frame in which the most recent data is stored.
int32* pFrameCount;	is pointer to receive the number of frames captured since the capture operation was begun. If no frames have been captured, a value of –1 is returned.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pNewestFrameIndex is NULL

Explanation

This function will return the index of the newest available frame and the current frame count.

Reference

dcam_precapture, dcam_capture, dcam_wait, dcam_idle, dcam_getstatus

dcam_gettriggermode()

Usage
Get the current synchronization mode.

Declaration

```
BOOL    dcam_gettriggermode( HDCAM h, int32* pMode);
```

Argument(s)s

HDCAM h;	specifies the camera.
int32* pMode;	is pointer to get the synchronization mode.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pMode is NULL
DCAMERR_NOTSUPPORT	The camera is not supported.

Explanation
DCAM Module returns the current synchronization mode. The value is one of following:

DCAM_TRIGMODE_INTERNAL	means the internal synchronization mode.
DCAM_TRIGMODE_EDGE	means the external synchronization edge mode.
DCAM_TRIGMODE_LEVEL	means the external synchronization level mode.
DCAM_TRIGMODE_SOFTWARE	means the software trigger mode. Trigger can be fire by dcam_firetrigger() function.
DCAM_TRIGMODE_TDI	means the camera captures images with TDI mode and trigger will shift image vertically one line.
DCAM_TRIGMODE_TDIINTERNAL	means the camera captures images with TDI mode without external trigger.
DCAM_TRIGMODE_START	means the trigger changes the camera mode from external trigger to internal.
DCAM_TRIGMODE_SYNCREADOUT	means the trigger starts reading out. The exposure time is the period between two triggers.

Reference
dcam_settriggermode, dcam_firetrigger

dcam_gettriggerpolarity()

Usage
Get the current external trigger polarity.

Declaration

```
BOOL    dcam_gettriggerpolarity( HDCAM h, int32* pPolarity);
```

Argument(s)s

HDCAM h;	specifies the camera.
int32* pPolarity;	is pointer to receives the polarity of the external trigger.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pPolarity is NULL
DCAMERR_NOTSUPPORT	The camera is not supported.

Explanation
This function returns the polarity of the external trigger. The value can be one of following:

DCAM_TRIGPOL_NEGATIVE	negative polarity.
DCAM_TRIGPOL_POSITIVE	positive polarity.

Reference
dcam_settriggerpolarity

dcam_idle()

Usage

Stop image capturing.

Declaration

```
BOOL    dcam_idle( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.

Explanation

This function stops image capturing. If the camera is in the middle of capturing an image, the capturing process is aborted and the image is invalid. If capturing is not in progress, nothing happens.

If the capture mode is DCAM_CAPTUREMODE_SNAP and capturing has already been completed, it is not necessary to use this function.

Reference

dcam_precapture, dcam_capture, dcam_freeframe, dcam_wait, dcam_getstatus

dcam_init()

Usage

Initializes DCAM-Manager and Modules.

Declaration

```
BOOL    dcam_init(void* reserved1 = 0, int32* pCount = 0, LPCSTR reserved2 = 0);
```

Argument(s)

void* reserved1;	is rserved to NULL.
int32* pCount;	is pointer to get the total number of available cameras.
LPCSTR reserved2;	is reserved to NULL.

Explanation

This function initializes DCAM-Manager and Modules. This function can only be called once per instance of DCAM-API. If this function is called while an instance of DCAM-API exists, this function will return FALSE. This function will return the total number of supported cameras found on the system.

Reference

dcam_open, dcam_close

dcam_lockbits ()

Usage
Lock the bitmap data.

Declaration

```
BOOL    dcam_lockbits( HDCAM h, BYTE** top, int32* rowbytes, int32 frame);
```

Argument(s)

HDCAM h;	specifies the camera.
BYTE** top;	is the pointer for the variable that receives the top address of the bitmap data buffer.
int32* rowbytes;	is the pointer to get offset byte value between a line and next line. A negative value may be returned in some cases.
int32 frame;	is the number of the frame for which the bitmap data is to be locked. if this value is -1, that means the latest captured frame.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	top is NULL
DCAMERR_NOTREADY	camera status is not ready. You need call the dcam_allocframe() function.

Explanation
This function returns the bitmap bits pointer that the application can access to. When access has been completed, the lock should immediately be canceled using the dcam_unlockbits() function.

The format of the bitmap bits is specified with the dcam_setbitstype() function.

In Windows, the locked data can be used with SetDIBits() and other functions, as bitmap data independent of any camera.

Reference
dcam_getsize, dcam_setbmptype, dcam_unlockbits, dcam_lockdata

dcam_lockdata()

Usage
Lock image data.

Declaration

```
BOOL    dcam_lockdata( HDCAM h, void** top, int32* rowbytes, int32 frame);
```

Argument(s)

HDCAM h;	specifies the camera.
void** top;	is the pointer for the variable that receives the address of the first line of the image data buffer.
int32* rowbytes;	is the pointer to get offset byte value between a line and next line. A negative value may be returned in some cases.
int32 frame;	is the number of the frame for which the image data is to be locked. if this value is -1, that means the latest captured frame.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	top is NULL
DCAMERR_NOTREADY	camera status is not ready. You need call the dcam_allocframe() function.

Explanation
This function returns a pointer that the application can use to access the image data. When access has been completed, the lock should immediately be canceled with the dcam_unlockdata() function.

The format of the data is specified with the dcam_setdatatype() function.

Reference
dcam_getsize, dcam_setdatatype, dcam_unlockdata, dcam_lockbits

dcam_open()

Usage

Open the camera and return the camera handle.

Declaration

```
BOOL    dcam_open( HDCAM* ph, int32 index, LPCSTR reserved = 0);
```

Argument(s)s

HDCAM* ph;	is the pointer to get the camera handle
int32 index;	specifies the index of camera.
LPCSTR reserved;	is reserved to NULL.

Explanation

This function initializes the camera at the specified index and returns a camera handle.

Reference

dcam_init, dcam_close, dcam_extended

dcam_precapture()

Usage
Prepare for capturing images.

Declaration

```
BOOL    dcam_precapture( HDCAM h, DCAM_CAPTUREMODE mode);
```

Argument(s)

HDCAM h;	specifies the camera.
DCAM_CAPTUREMODE mode;	specifies the capture mode.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOMEMORY	Insufficient memory
DCAMERR_NOTSTABLE	Camera is not stable state.

Explanation
You can choose a value from following for mode.

DCAM_CAPTUREMODE_SNAP	One cycle of images are captured in this mode. Data takes priority in this mode.
DCAM_CAPTUREMODE_SEQUENCE	Images are captured continuously in this mode, and data takes priority.

This function prepares the camera for image capturing and sets the camera to STABLE state. This function only sets parameters necessary for image capturing, but does not actually initiate capturing. Capturing is initialized with the dcam_capture() function.

Reference
dcam_capture, dcam_idle, dcam_wait, dcam_getstatus

dcam_queryupdate()

Usage

Check the changing of the camera settings.

Declaration

```
BOOL    dcam_queryupdate( HDCAM h, _DWORD* pFlag, _DWORD dwReserved= 0);
```

Argument(s)s

HDCAM h;	specifies the camera.
_DWORD* pFlag;	is pointer to receive the status change.
_DWORD dwReserved;	is reserved to 0.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	pFlag is NULL

Explanation

DCAM keeps track when certain parameters have changed through the setting of other functions. For example, the exposure time may be altered when a new binning mode is set. This function returns information on parameters that have changed.

DCAM_UPDATE_RESOLUTION	Resolution has changed
DCAM_UPDATE_AREA	Image size has changed
DCAM_UPDATE_DATATYPE	Image data type has changed
DCAM_UPDATE_BITSTYPE	Bitmap data type has changed
DCAM_UPDATE_EXPOSURE	Exposure time has changed
DCAM_UPDATE_TRIGGER	Trigger setting has changed

All update flags will reset to 0 when this function is called.

Reference

dcam_setbinning, dcam_getdatasize, dcam_wait

dcam_releasebuffer ()

Usage

Releases the attached buffer.

Declaration

```
BOOL    dcam_releasebuffer( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_BUSY	Camera is capturing now.
DCAMERR_INVALIDHANDLE	Invalid camera handle.

Explanation

This function is used to release the memory that had been previously attached to DCAM with dcam_attachbuffer(). This function does not destroy that memory, but only prevents DCAM from being able to access it.

Reference

dcam_attachbuffer

dcam_setbinning()

Usage

Change the binning mode.

Declaration

```
BOOL    dcam_setbinning( HDCAM h, int32 binning);
```

Argument(s)

HDCAM h;	specifies the camera.
int32 binning;	specifies the number of binnings.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	Camera is supported but binning value is wrong.
DCAMERR_NOTSTABLE	Camera is not stable state.
DCAMERR_NOTSUPPORT	Camera is not supported.

Explanation

This function changes the binning mode of the camera.

If the binning mode is changed, the image size changes. When transferring data, the image size must be checked, using the dcam_getsize() function.

Reference

dcam_getsize, dcam_getbinning

dcam_setbitstype()

Usage

Change the bitmap bits type.

Declaration

```
BOOL    dcam_setbitstype( HDCAM h, DCAM_BITSTYPE type);
```

Argument(s)s

HDCAM h;	specifies the camera.
DCAM_BITSTYPE type;	specifies the bitmap bits type.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOTSTABLE	Camera is not stable state.
DCAMERR_UNKNOWNBITSTYPE	The specified bitmap type is not supported.

Explanation

When the application calls the dcam_lockbits() function, the bitmap bits type is same as that specifies in this function.

You can choose one of following value for type.

DCAM_BITSTYPE_INDEX8	256-color index color
DCAM_BITSTYPE_INDEX24	24-bit full color

Reference

dcam_setbitstype, dcam_getdatatype, dcam_lockbits

dcam_setdatatype()

Usage
Change the data type for the image.

Declaration

```
BOOL    dcam_setdatatype( HDCAM h, DCAM_DATATYPE type);
```

Argument(s)s

HDCAM h;	specifies the camera.
DCAM_DATATYPE type;	specifies the image type.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOTSTABLE	Camera is not stable state.
DCAMERR_UNKNOWNDATATYPE	The specified data type is not supported.

Explanation
This function set the camera mode to the data type specified. Here is a list of possible choices, however not all are available to all cameras.

DCAM_DATATYPE_UINT8	8-bit integer, no sign
DCAM_DATATYPE_UINT16	16-bit integer, no sign
DCAM_DATATYPE_RGB24	24-bit RGB color
DCAM_DATATYPE_RGB48	48-bit RGB color

Reference
dcam_setdatatype, dcam_getbitstype, dcam_lockdata

dcam_setexposuretime()

Usage

Set the exposure time.

Declaration

```
BOOL    dcam_setexposuretime( HDCAM h, double sec);
```

Argument(s)

HDCAM h;	specifies the camera.
double sec;	specifies the exposure time, in seconds.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_NOTSUPPORT	The camera is not supported.

Explanation

This function sets a new exposure time in seconds. For example, one millisecond is specified as 0.001.

Depending on the camera, the exposure time that is being specified will not be the actually exposure time set. The camera may round up to the next valid value. Use the dcam_getexposuretime() function to get the actual exposure time that was set in the camera.

Reference

dcam_getexposuretime

dcam_setbitsinputlutrange()

Usage

Set the input range of the LUT to make the bitmap bits.

Declaration

```
BOOL    dcam_setbitsinputlutrange( HDCAM h, int32 inMax, int32 inMin = 0);
```

Argument(s)

HDCAM h;	specifies the camera.
int32 inMax;	specifies the maximum value for the input range.
int32 inMin;	specifies the minimum value for the input range.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	Invalid parameter

Explanation

DCAM Module uses as the input range of the LUT when it makes bitmap data. To specify the range for 8-bit data, the dcam_setbitsoutputlutrange() function is used.

Reference

dcam_getbits, dcam_setbitsoutputlutrange

dcam_setbitsoutputlutrange()

Usage

Set the output range of the LUT to make the bitmap bits.

Declaration

```
BOOL    dcam_setbitsoutputlutrange( HDCAM h, BYTE outMax, BYTE outMin = 0);
```

Argument(s)

HDCAM h;	specifies the camera.
BYTE outMax;	specifies the maximum value for the output range.
BYTE outMin;	specifies the minimum value for the output range.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	Invalid parameter.

Explanation

DCAM Module uses as the output range of the LUT when it makes bitmap data. To specify the input range, the dcam_setbitsinputindexrange() function is used.

Reference

dcam_getbits, dcam_setbitsinputlutrange

dcam_settriggermode()

Usage
Change the synchronization (trigger) mode.

Declaration

```
BOOL    dcam_settriggermode( HDCAM h, int32 mode);
```

Argument(s)

HDCAM h;	specifies the camera.
int32 mode;	specifies the synchronization (trigger) mode.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	The parameter is invalid.
DCAMERR_NOTSUPPORT	The camera does not support the specified trigger mode.

Explanation
This function sets the synchronization (trigger) mode of the camera. You can choose a of trigger mode from following values. However not all values are available for all cameras.

DCAM_TRIGMODE_INTERNAL	internal synchronization mode.
DCAM_TRIGMODE_EDGE	exposure begins with an external trigger source.
DCAM_TRIGMODE_LEVEL	exposure begins with an external trigger and exposure length is controlled by the length of the pulse.
DCAM_TRIGMODE_SOFTWARE	exposure begins when dcam_firetrigger() is called.
DCAM_TRIGMODE_TDI	external trigger shifts the image by one line and one line is read out.
DCAM_TRIGMODE_TDIINTERNAL	similar to TDI mode, but the camera controls the timing.
DCAM_TRIGMODE_START	the first image waits for an external trigger then changes to internal trigger mode.
DCAM_TRIGMODE_SYNCREADOUT	external trigger simultaneously reads out the current image and starts a new image.

The dcam_settriggerpolarity() function is used to switch between the rising and falling edge in Edge mode, and between High and Low for the effective level in Level mode.

Reference
dcam_gettriggermode, dcam_settriggerpolarity, dcam_firetrigger()

dcam_settriggerpolarity()

Usage

Change the polarity of the external trigger mode.

Declaration

```
BOOL    dcam_settriggerpolarity( HDCAM h, int32 polarity);
```

Argument(s)

HDCAM h;	specifies the camera.
int32 polarity;	A variable range is conveyed for obtaining the logic of the external trigger mode.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_INVALIDPARAM	The parameter is invalid.
DCAMERR_NOTSUPPORT	The camera is not supported.

Explanation

This function sets the logic of the external trigger. You can select the trigger polarity from following values.

DCAM_TRIGPOL_NEGATIVE	Negative logic.
DCAM_TRIGPOL_POSITIVE	Positive logic.

In Edge mode, imaging begins at the rising edge with positive logic and at the falling edge with negative logic. In Level mode, the exposure time is high level with positive logic and low level with negative logic.

Reference

dcam_gettriggerpolarity

dcam_uninit()

Usage

Terminates DCAM-API.

Declaration

```
BOOL    dcam_uninit(void* reserved1 = 0, LPCSTR reserved2 = 0);
```

Argument(s)

void* reserved1;	is reserved to NULL.
LPCSTR reserved2;	is reserved to NULL.

Explanation

This function will cleanup all resources and objects used by this DCAM. All open cameras will be forcefully closed. Any resources used by the individual cameras will also be freed. No new cameras can be opened.

Reference

dcam_init

dcam_unlockbits()

Usage

Cancel locking bitmap bits.

Declaration

```
BOOL    dcam_unlockbits( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.

Explanation

This function cancels the locking bitmap bits by using the dcam_lockbits() function. If locked bits is no longer necessary, the application should call this function as soon as possible.

Reference

dcam_allocframe, dcam_lockbits, dcam_unlockdata

dcam_unlockdata()

Usage

Cancel locking image data.

Declaration

```
BOOL    dcam_unlockdata( HDCAM h);
```

Argument(s)

HDCAM h;	specifies the camera.

Error value

DCAMERR_INVALIDHANDLE	Invalid camera handle.

Explanation

This function unlocks the data buffer that was locked by the dcam_lockdata() function. If locked data is no longer necessary, the application may call this function to allow DCAM to use that data buffer for image capturing. A previously locked frame is automatically unlocked when a new frame has been locked.

Reference

dcam_allocframe, dcam_lockdata, dcam_unlockbits

dcam_wait()

Usage
The system waits for an event to be generated.

Declaration

```
BOOL    dcam_wait( HDCAM h, _DWORD* pCode, _DWORD timeout = 0
                   , HDCAMSIGNAL abort = 0);
```

Argument(s)

HDCAM h;	specifies the camera.
_DWORD*pCode;	specifies the wait status. One of the following should be specified.
_DWORD timeout;	specifies the waiting time.
HDCAMSIGNAL abort;	specifies an event handle, used to notify of an abort request from an external source while waiting.

Error value

DCAMERR_ABORT	Aborted by means of an event handle.
DCAMERR_BUSY	There are too many wait .
DCAMERR_INVALIDHANDLE	Invalid camera handle.
DCAMERR_TIMEOUT	No event occur.

Explanation
This function causes the thread to wait until the camera reaches a certain status during capture.

You can choose following events.

DCAM_EVENT_FRAMEBEGIN	Waits for beginning of data recording.
DCAM_EVENT_FRAMEEND	Waits for a frame to be recorded
DCAM_EVENT_CYCLEEND	Waits for a cycle to be captured
DCAM_EVENT_VVALIDBEGIN	Waits for beginning of camera output.
DCAM_EVENT_CAPTUREEND	Waits for dcam_idle() to be called or after capturing all images by DCAM_CAPTUREMODE_SNAP.

All DCAM modules support DCAM_EVENT_FRAMEEND, DCAM_EVENT_CYCLEEND and DCAM_EVENT_CAPTUREEND.

DCAM_WAIT_INFINITE can be used for infinite timeout.

HDCAMSIGNAL is event HANDLE on Windows, MPEventID on MacOSX.

Reference
dcam_precapture, dcam_capture, dcam_idle, dcam_freeframe, dcam_getstatus

APPENDIX

A. Function Validation

Initialize and Termination

pre-init	pre-open	unstable,stable,ready,busy	Function
ERROR	ERROR	OK	dcam_getlasterror()
>> pre-open	ERROR	ERROR	dcam_init()
ERROR	OK	OK	dcam_getmodelinfo()
ERROR	>> unstable	ERROR	dcam_open()
ERROR	ERROR	>> pre-open	dcam_close()
OK	>> pre-init	>> pre-init	dcam_uninit()

Prepare capturing

unstable	stable	ready	busy	Function
OK	OK	OK	OK	dcam_getstring()
OK	OK	OK	OK	dcam_getcapability()
OK*1	OK	OK	OK	dcam_getbinning()
OK	>> unstable	not stable	not stable	dcam_setbinning()
unstable	stable	ready	busy	Function
OK*1	OK	OK	OK	dcam_getdatatype()
OK*1	OK	OK	OK	dcam_getbitstype()
OK	>> unstable	not stable	not stable	dcam_setdatatype()
OK	>> unstable	not stable	not stable	dcam_setbitstype()
OK*1	OK	OK	OK	dcam_getdatasize()
OK*1	OK	OK	OK	dcam_getbitssize()
OK	OK	OK	OK	dcam_getexposuretime()
OK	OK	OK	OK	dcam_gettriggermode()
OK	OK	OK	OK	dcam_gettriggerpolarity()
OK	OK	OK	OK	dcam_setexposuretime()
OK	OK	OK	OK*2	dcam_settriggermode()
OK	OK	OK	OK*2	dcam_settriggerpolarity()
unstable	stable	ready	busy	Function
OK	OK	OK	OK	dcam_queryupdate()
occur	occur	never	never	DCAM_UPDATE_RESOLUTION
occur	occur	never	never	DCAM_UPDATE_AREA
occur	occur	never	never	DCAM_UPDATE_DATATYPE
occur	occur	never	never	DCAM_UPDATE_BITSTYPE
occur	occur	occur	occur	DCAM_UPDATE_EXPOSURE
occur	occur	occur	occur*2	DCAM_UPDATE_TRIGGER

Capturing and other

unstable	stable	ready	busy	Function
OK	OK	not stable	not stable	dcam_precapture()
OK*1	OK	OK	OK	dcam_getdatarange()
OK*1	OK	OK	OK	dcam_getbuffersize()
not stable	>> ready	OK*3	busy	dcam_attachbuffer()
not stable	>> ready	OK*4	busy	dcam_allocframe()
OK	OK	OK	OK	dcam_getframecount()
not ready	not ready	>> busy	busy	dcam_capture()
OK	OK	OK	>> ready	dcam_idle()
OK	OK	OK	OK	dcam_wait()
not busy	not busy	not busy	OK	dcam_firetrigger()
OK	OK	OK	OK	dcam_gettransferinfo()
OK	OK	OK	busy	dcam_freeframe()
OK	OK	OK	busy	dcam_releasebuffer()
unstable	stable	ready	busy	Function
not ready	not ready	OK	OK	dcam_lockdata()
not ready	not ready	OK	OK	dcam_lockbits()
OK	OK	OK	OK	dcam_unlockdata()
OK	OK	OK	OK	dcam_unlockbits()
OK	OK	OK	OK	dcam_setbitsinputlutrange()
OK	OK	OK	OK	dcam_setbitsoutputlutrange()
unstable	stable	ready	busy	Function
not support	not support	not support	not support	dcam_showpanel()
OK	OK	OK	OK	dcam_extended()

HAMAMATSU

Homepage Address http://www.hamamatsu.com

HAMAMATSU PHOTONICS K.K., Systems Division
812 Joko-cho, Hamamatsu City, 431-3196, Japan, Telephone: (81)53-431-0124, Fax: (81)53-435-1574, E-mail:export@sys.hpk.co.jp

U.S.A. and Canada: Hamamatsu Photonic Systems: 360 Foothill Road, Bridgewater, N.J. 08807-0910, U.S.A., Telephone: (1)908-231-1116, Fax: (1)908-231-0852, E-mail:usa@hamamatsu.com
Germany: Hamamatsu Photonics Deutschland GmbH: Arzbergerstr. 10, D-82211 Herrsching am Ammersee, Germany, Telephone: (49)8152-375-0, Fax: (49)8152-2658E-mail:info@hamamatsu.de
France: Hamamatsu Photonics France S.A.R.L.: 8, Rue du Saule Trapu, Parc du Moulin de Massy, 91882 Massy Cedex, France, Telephone: (33) 1 69 53 71 00, Fax: (33)1 69 53 71 10E-mail:france@hamamatsu.com
United Kingdom: Hamamatsu Photonics UK Limited: Lough Point, 2 Gladbeck Way, Windmill Hill, Enfield, Middlesex EN2 7JA, United Kingdom, Telephone: (44)208-367-3560, Fax: (44)208-367-6384, E-mai:info@hamamatsu.co.uk
North Europe: Hamamatsu Photonics Norden AB: Smidesvagen 12, SE-171-41 Solna, Sweden, Telephone: (46)8-509-031-00, Fax: (46)8-509-031-01, mail:system@hamamatsu.se
Italy: Hamamatsu Photonics Italia S.R.L.: Strada della Moia, 1/E20020Arese(Milano), Italy, Telephone: (39) 02-935 81 733, Fax: (39) 02-935 81 741E-mail:info@hamamatsu.it