Shad-o-Snap Software Development Kit (SDK)

User's Manual

Rad-icon Imaging Corp.

P/N 1099 Rev. 01

1. Introduction

Thank you for purchasing the Shad-o-Snap Software Development Kit (SDK). This manual will guide you through the installation and use of this library. Our goal is to get you started on the image acquisition portion of your imaging project, and to help you develop a successful application.

Overview

The Shad-o-Snap SDK consists of a complete set of I/O function calls that allow you to communicate with your Shad-o-Snap camera. Functions are provided to send imaging commands to the camera, to access camera status information and to retrieve images from the Shad-o-Snap camera's on-board memory.

The Shad-o-Snap SDK does not contain image processing functions that are necessary to perform image correction or automated analysis. For a library of function calls to do offset, gain and pixel corrections please refer to our ShadoCam Imaging Library literature. Numerous third-party software packages are available to perform automated image analysis on images acquired with a Shad-o-Snap camera.

Technical Support

Although we have attempted to make this manual as complete as possible, we realize that there are always additional unanswered questions, as well as unique situations not covered in this booklet. Rad-icon is committed to providing excellent customer service and technical support for all of our products. After all, your success is our business.

For technical assistance with the Shad-o-Snap SDK or your Shad-o-Snap camera please e-mail your questions to support@rad-icon.com, or contact our customer service department (8 am to 5 pm Pacific Time) at 408-486-0886. Please be prepared to give a detailed description of your problem.

For the latest contact information, data sheets and application notes please visit our web site at http://www.rad-icon.com.

2. Installation

Installation of the Shad-o-Snap SDK is as simple as double-clicking on the self-extracting archive (SnapSDK_V1.x.exe) on the installation disk. The installation program will create a folder called "SnapSDK" (in your "C:\Program Files" directory by default). The following files will be copied to this folder:

SnapSDK.dll	Shad-o-Snap SDK run-time library
License.txt	End-user license agreement
Readme.txt	Installation and release information
SdkManual.pdf	This manual in PDF format
\Include\SnapSDK.h	Shad-o-Snap SDK C/C++ header file
\Lib\SnapSDK.lib	Shad-o-Snap SDK C/C++ link library

To start using the Shad-o-Snap SDK functions, make sure that the paths for the header file and link library are accessible to your compiler or development system. Copy the run-time library (.DLL) file into your application directory or to the "C:\WinNT\System32" (or equivalent) directory.

Make sure that your Shad-o-Snap camera is running and connected to your PC. The Shad-o-Snap camera installs itself as an external hard drive (e.g. D:\ or E:\) on your machine. To locate the camera, use the Windows Explorer program to look for a new drive that contains a small number of image files and a "setup" text file.

3. The Shad-o-Snap USB Interface

The Shad-o-Snap camera firmware emulates a Mass Storage Class (MSC) interface as defined in the USB standard. This interface is commonly used for mass storage devices (e.g. an external hard drive) and is automatically recognized by the Windows 2000, ME and XP operating systems, as well as the Mac OS X and other modern platforms. Windows includes built-in drivers that are automatically installed in order to communicate with the Shad-o-Snap camera. The camera then appears as an external drive (for example D:\ or E:\) and can be examined as such. However, while it is possible to read the files that appear on the "camera drive", any data written back to the camera will be ignored. Writing data to the camera will not cause any damage, but it may confuse the operating system and should therefore be avoided. Instead, the Shad-o-Snap SDK functions provide a safe and controlled way to send commands to the camera and read information back to the host PC.

Camera commands are sent by writing to the setup file ("SETUP.TXT") on the camera drive. Camera status information and images are retrieved by reading the setup and image files. There are five image files available on the camera drive: a "thumbnail" TIFF image, two full-size TIFF and RAW images, and two full-size TIFF and RAW offset images. These images can be retrieved at any time, even if there is no valid image information stored in the camera's memory.

One disadvantage of the Windows MSC interface is that the operating system automatically caches any information retrieved from the camera drive. The disk cache is not updated when the camera status changes (for example when a new image is acquired). The SDK functions have been developed to bypass the disk cache and allow the user to read the current status and image data directly into a memory buffer.

The Shad-o-Snap SDK calls consist of general I/O functions (OpenCamera, CloseCamera, ErrorMessage, LoadImage, ReadSetup) and specific commands (Acquire, GetStatus, SetStatus). General functions don't return to the calling program until the task has been completed (i.e. the entire image has been transferred), which may take several seconds.

Command functions return immediately after retrieving the status information or sending the command to the camera. There is no active feedback that the camera has successfully executed a command (although if the command was communicated successfully, there is no reason why it shouldn't be executed as well). If necessary, you can use the "GetStatus" commands to verify that the desired change in status has occurred. This especially applies to the "AcquireImage" and "AcquireOffset" commands, which require active polling of the camera status to determine if the image acquisition has been completed. We recommend a polling interval of several hundred milliseconds (e.g. 500 ms) to prevent overloading the camera interface with status requests.

A typical imaging sequence might look as follows:

1.	**Snap_OpenCamera();**	initialize SDK
2.	**SnapCmd_SetIntegrationTime(2000);**	camera µP sets up internal timing to provide periodic start pulses that trigger readout sequence and reset sensor
3.	**SnapCmd_AcquireOffset();**	µP grabs next available image from sensor and transfers to RAM; offset image is condensed to 8 bits/pixel
4.	**SnapCmd_GetOffsetState();**	poll until offset image is valid
5.	**SnapCmd_SetOffsetCorrection(1);**	turn Offset Correction on
6.	**SnapCmd_AcquireImage();**	µP grabs next available image from sensor, transfers to RAM and performs offset correction
7.	**SnapCmd_GetImageState();**	poll until image is valid
8.	**Snap_LoadThumbnail(pImgBuf, pBufSize);**	take quick look at image; if necessary adjust parameters and go back to (6)
9.	**Snap_LoadRawImage(pImgBuf, pBufSize);**	transfer entire image from camera to software buffer in host PC
10.	**Snap_CloseCamera();**	close SDK

4. Shad-o-Snap SDK Library

This section contains a complete function reference for the Shad-o-Snap SDK. The function calls have been developed for the Windows 2000 and Windows XP operating systems, and are intended to work with C or C++ applications with the provided header file.

Data Types

Image buffers used in this SDK are either of unsigned character type (one byte per pixel) for TIFF images, or unsigned short integer type (two bytes per pixel) for RAW integer files. All image formats contain row-sequential pixel data. Function return values (error codes) are of signed short integer type. Other parameters vary as indicated in the function definition.

The following table gives the sizes of the various data types that are used by the SIL:

Type	*Size*
char, unsigned char	8 bits
short, unsigned short	16 bits
unsigned long	32 bits
all pointers (char*, short* etc.)	32 bits

Constants

The SDK contains these defined constants:

SNAP_MAXERRORMESSAGE..... the maximum error message length (128 characters).

SNAP_SETUPFILESIZE ..... the size of the "SETUP.TXT" file (252 bytes).

SNAP_IMAGEBUFSIZE..... the size of the internal image buffer (16384 bytes).

Additional defined constants are listed on the following pages. Please refer to the C header file if you need to see the actual definitions.

Snap_OpenCamera

short **Snap_OpenCamera(**void**);**

Return Value

SNAPERROR_NOERROR if successful;
SNAPERROR_NOCAMERA or SNAPERROR_IOERROR on failure.

Parameters

None.

Description

Locates an attached Shad-o-Snap camera and initializes the SDK functions. Returns SNAPERROR_NOCAMERA if it can't find a Shad-o-Snap camera attached to the PC. Returns SNAPERROR_IOERROR if a camera was found but the function is unable to communicate with it.

This function MUST BE CALLED before attempting to communicate with the Shad-o-Snap camera through any of the other function calls.

Snap_CloseCamera

short **Snap_CloseCamera(**void**);**

Return Value

SNAPERROR_NOERROR.

Parameters

None.

Description

Releases any system resources used by the Shad-o-Snap SDK. Call this function at the end of your imaging session before closing your application program.

Snap_ErrorMessage

short **Snap_ErrorMessage(**short **errCode,** char ***errMessage);**

Return Value

SNAPERROR_NOERROR if successful; SNAPERROR_INVALIDERRCODE if error code is not valid.

Parameters

errCode the error code to be deciphered.

errMessage pointer to a character buffer to receive the error message text.

Description

Looks up the error code provided and returns the corresponding text error message as a null-terminated string. The maximum length of the error message is defined by the **SNAP_MAXERRORMESSAGE** parameter. See Appendix B for a listing of error messages.

Snap_LoadRawImage

short **Snap_LoadRawImage(**unsigned short ***imgBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_OPENIMAGEFILE on failure.

Parameters

imgBuf pointer to the raw image buffer

pBufSize pointer to a variable containing the size of the image buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads the 16-bit raw integer copy of the image currently stored in the camera memory into an image buffer supplied by the calling program. The image is accessed through the "IMAGE.RAW" file on the camera drive. If the function can not read the "IMAGE.RAW" file it returns SNAPERROR_OPENIMAGEFILE.

The variable pointed to by **pBufSize** should initially contain the maximum size of the raw image buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value).

Snap_LoadRawOffset

short **Snap_LoadRawOffset(**unsigned short ***imgBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_OPENIMAGEFILE on failure.

Parameters

imgBuf pointer to the raw image buffer

pBufSize pointer to a variable containing the size of the image buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads the 16-bit raw integer copy of the offset image currently stored in the camera memory into an image buffer supplied by the calling program. The image is accessed through the "OFFSET.RAW" file on the camera drive. If the function can not read the "OFFSET.RAW" file it returns SNAPERROR_OPENIMAGEFILE.

The variable pointed to by **pBufSize** should initially contain the maximum size of the raw image buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value).

Snap_LoadThumbnail

short **Snap_LoadThumbnail(**unsigned char ***imgBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_OPENTHMBNAILFILE on failure.

Parameters

imgBuf pointer to the thumbnail image buffer

pBufSize pointer to a variable containing the size of the image buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads a “thumbnail” copy (128x124 pixels) of the image currently stored in the camera memory into an image buffer supplied by the calling program. The image is accessed through the "THMBNAIL.TIF" file on the camera drive. If the function is unable to read the "THMBNAIL.TIF " file it returns SNAPERROR_OPENTHMBNAILFILE.

The variable pointed to by **pBufSize** should initially contain the maximum size of the thumbnail image buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value). The size of the thumbnail image is equal to SNAP_IMAGEBUFSIZE (16384 bytes), which includes a 512-byte TIFF header.

Snap_LoadTifImage

short **Snap_LoadTifImage(**unsigned char ***imgBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_OPENIMAGEFILE on failure.

Parameters

imgBuf pointer to the TIFF image buffer

pBufSize pointer to a variable containing the size of the image buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads the 8-bit TIFF copy of the image currently stored in the camera memory into an image buffer supplied by the calling program. The image is accessed through the "IMAGE.TIF" file on the camera drive. If the function can not read the "IMAGE.TIF" file it returns SNAPERROR_OPENIMAGEFILE.

The variable pointed to by **pBufSize** should initially contain the maximum size of the image buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value). Note that the TIFF image includes a 512-byte header at the beginning of the file.

Snap_LoadTifOffset

short **Snap_LoadTifOffset(**unsigned char ***imgBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_OPENIMAGEFILE on failure.

Parameters

imgBuf pointer to the TIFF image buffer

pBufSize pointer to a variable containing the size of the image buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads the 8-bit TIFF copy of the offset image currently stored in the camera memory into an image buffer supplied by the calling program. The image is accessed through the "OFFSET.TIF" file on the camera drive. If the function can not read the "OFFSET.TIF" file it returns SNAPERROR_OPENIMAGEFILE.

The variable pointed to by **pBufSize** should initially contain the maximum size of the image buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value). Note that the TIFF image includes a 512-byte header at the beginning of the file.

Snap_ReadSetupInfo

short **Snap_ReadSetupInfo(**unsigned char ***setupBuf,** unsigned long ***pBufSize);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

setupBuf pointer to the SetupInfo buffer

pBufSize pointer to a variable containing the size of the SetupInfo buffer (in bytes); returns the actual number of bytes read into the buffer

Description

Reads the current camera SetupInfo data into a text buffer supplied by the calling program. This information is accessed through the "SETUP.TXT" file on the camera drive. If the function can not read the "SETUP.TXT" file it returns SNAPERROR_IOERROR.

The variable pointed to by **pBufSize** should initially contain the maximum size of the SetupInfo buffer. After reading the information, the function replaces this value with the number of bytes actually transferred (less than or equal to the initial value). The minimum size of the buffer needed to read the entire SetupInfo file is given by the SNAP_SETUPFILESIZE constant.

See Appendix A for a listing and explanation of the "SETUP.TXT" file.

SnapCmd_AcquireImage

short **SnapCmd_AcquireImage(**void**);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

None.

Description

Sends a command to the Shad-o-Snap camera to acquire the next available image from the sensor and transfer it into the camera memory. The call returns immediately after sending the command, with a return value of SNAPERROR_NOERROR if the command was sent successfully, or with SNAPERROR_IOERROR if the function was unable to communicate with the camera. Use the **SnapCmd_GetImageState()** function to poll the camera in order to find out if the image acquisition process has been completed.

SnapCmd_AcquireOffset

short **SnapCmd_AcquireOffset(**void**);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

None.

Description

Sends a command to the Shad-o-Snap camera to acquire the next available image from the sensor and transfer it to the Offset Image storage in the camera memory. The call returns immediately after sending the command, with a return value of SNAPERROR_NOERROR if the command was sent successfully, or with SNAPERROR_IOERROR if the function was unable to communicate with the camera. Use the **SnapCmd_GetOffsetState()** function to poll the camera in order to find out if the image acquisition process has been completed.

SnapCmd_GetImageGain

short **SnapCmd_GetImageGain(**short ***pImageGain);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pImageGain pointer to a variable to accept the current ImageGain setting.

Description

Retrieves the current ImageGain setting from the camera. The ImageGain value is placed in the variable pointed to by **pImageGain**. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The ImageGain setting is used to scale the TIFF images retrieved from the camera. Possible values are between –2 and 4, a value of 4 meaning that the highest 8 bits of the 12-bit raw image (i.e. bits 4-11) are mapped into the 8-bit TIFF image, and a value of –2 meaning that only the lowest 6 bits are mapped. In other words, the 8-bit TIFF image is derived from the 12-bit raw image by scaling (multiplying) the raw image by 2^(-ImageGain).

See also **SnapCmd_SetImageGain()**.

SnapCmd_GetImageState

short **SnapCmd_GetImageState(**void**);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_NOVALIDIMAGE or SNAPERROR_IOERROR on failure.

Parameters

None.

Description

Checks to see if a valid image is stored in the camera memory. Use this function to poll the camera status after sending the **SnapCmd_AcquireImage()** command.

Returns SNAPERROR_NOERROR if a valid image is present. Returns SNAPERROR_ NOVALIDIMAGE if the image acquisition is still in progress, or if the camera memory has not been initialized. Returns SNAPERROR_IOERROR if the function is unable to communicate with the camera.

SnapCmd_GetIntegrationTime

short **SnapCmd_GetIntegrationTime(**unsigned short ***pIntTime);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pIntTime pointer to a variable to accept the current IntegrationTime setting.

Description

Retrieves the current IntegrationTime setting from the camera. The IntegrationTime value is placed in the variable pointed to by **pIntTime**. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The returned IntegrationTime value is the current setting of the on-board counter that sends periodic start pulses to the camera sensor (in milliseconds). Please refer to the **SnapCmd_GetTimingMode()** command for a discussion on camera timing.

See also **SnapCmd_SetIntegrationTime(), SnapCmd_SetTimingMode()**.

SnapCmd_GetOffsetCorrection

short **SnapCmd_GetOffsetCorrection(**char ***pOffsetCorr);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pOffsetCorr pointer to a variable to accept the current OffsetCorrection setting.

Description

Retrieves the current OffsetCorrection setting from the camera. The OffsetCorrection value is placed in the variable pointed to by **pOffsetCorr**. The function returns SNAPERROR_ NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The returned OffsetCorrection value is the current offset correction state of the camera. Possible values are 0 (offset correction disabled) or 1 (offset correction enabled).

See also **SnapCmd_SetOffsetCorrection().**

SnapCmd_GetOffsetGain

short **SnapCmd_GetOffsetGain(**short ***pOffsetGain);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pOffsetGain pointer to a variable to accept the current OffsetGain setting.

Description

Retrieves the current OffsetGain setting from the camera. The OffsetGain value is placed in the variable pointed to by **pOffsetGain**. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The OffsetGain setting is used to scale the offset images acquired from the sensor. Possible values are 0, 1 and 2, corresponding to multiplication factors of 1, ½ and ¼. For most applications the default value of 0 should be used.

OffsetGain control is provided because the camera stores the internal offset image using only the lowest 8 bits for each pixel. Normally this is sufficient since the average pixel value in an offset image should be well below 256. Any offset image pixel values above 255 will be truncated. In some rare cases when there is a large amount of dark signal present, it may be necessary to scale the offset image so that offsets larger than 255 can be handled. An OffsetGain value of 1 will allow offset values up to 512, and a value of 2 will handle offset values up to 1024. However, this occurs at the expense of losing the resolution of the least significant one or two bits.

Since the offset image is scaled at the time it is acquired, changing the OffsetGain setting renders the current offset image invalid. Offset correction will be disabled until a new offset image has been acquired.

See also **SnapCmd_SetOffsetGain()**.

SnapCmd_GetOffsetState

short **SnapCmd_GetOffsetState(**void**);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_NOVALIDOFFSET or SNAPERROR_IOERROR on failure.

Parameters

None.

Description

Checks to see if a valid offset image is stored in the camera memory. Use this function to poll the camera status after sending the **SnapCmd_AcquireOffset()** command.

Returns SNAPERROR_NOERROR if a valid image is present. Returns SNAPERROR_ NOVALIDOFFSET if the image acquisition is still in progress, or if the camera memory has not been initialized. Returns SNAPERROR_IOERROR if the function is unable to communicate with the camera.

SnapCmd_GetResetState

short **SnapCmd_GetResetState(**char ***pRESET);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pRESET pointer to a variable to accept the current RESET setting.

Description

Retrieves the current RESET setting from the camera. The RESET value is placed in the variable pointed to by **pRESET**. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The returned RESET value reflects the current status of the global reset input to the sensor. Possible values are 0 (RESET off) or 1 (RESET enabled).

See also **SnapCmd_SetResetState().**

SnapCmd_GetTimingMode

short **SnapCmd_GetTimingMode(**short ***pTimingMode);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

pTimingMode pointer to a variable to accept the current TimingMode setting.

Description

Retrieves the current TimingMode setting from the camera. The TimingMode value is placed in the variable pointed to by **pTimingMode**. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The returned TimingMode value reflects the current timing mode of the camera. There are a total of four different timing modes (0-3). In the default timing mode "0" all camera timing signals are generated internally. The detector integration time (period from one frame readout cycle to the next) is controlled by the on-board counter (see **SnapCmd_SetIntegrationTime**).

Timing Mode "1" enables the "Ext. Sync In" SMA connector on the camera front panel and allows the sensor integration time to be controlled from an external pulse generator.

Timing modes "2" and "3" implement an electronic shutter to work with timed x-ray exposures. In this case either a software trigger (via the **AcquireImage** command) or a hardware trigger (a rising edge on the "Ext. Sync In" input) starts a programmed timing sequence which first resets the photodiode array, then waits and integrates for a preselected period of time, and then reads out the image.

Please refer to the Shad-o-Snap Hardware Manual for additional information about the camera timing and timing modes.

See also **SnapCmd_SetTimingMode()**.

SnapCmd_SetImageGain

short **SnapCmd_SetImageGain(**short **nImageGain);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_INVALIDPARAMETER or SNAPERROR_IOERROR on failure.

Parameters

nImageGain variable containing the new ImageGain setting.

Description

Writes a new ImageGain setting to the camera. The function returns SNAPERROR_NOERROR if the value was written successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera. A return value of SNAPERROR_INVALIDPARAMETER indicates that the value passed by **nImageGain** was out of range (less than –2 or greater than 4).

See also **SnapCmd_GetImageGain()**.

SnapCmd_SetIntegrationTime

short **SnapCmd_SetIntegrationTime(**unsigned short **nIntTime);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_INVALIDPARAMETER or SNAPERROR_IOERROR on failure.

Parameters

nIntTime variable containing the new IntegrationTime setting, in milliseconds.

Description

Writes a new IntegrationTime setting to the camera. The function returns SNAPERROR_ NOERROR if the value was written successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera. A return value of SNAPERROR_ INVALIDPARAMETER indicates that the value passed by **nIntTime** was out of range (less than 20 or greater than 33500).

The new IntegrationTime value is used to program the on-board counter that sends periodic start pulses to the camera sensor. Please refer to the **SnapCmd_GetTimingMode()** command for a discussion on camera timing.

See also **SnapCmd_GetIntegrationTime(), SnapCmd_SetTimingMode()**.

SnapCmd_SetOffsetCorrection

short **SnapCmd_SetOffsetCorrection(**char **bOffsetCorr);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

bOffsetCorr variable containing the new OffsetCorrection setting.

Description

Writes a new OffsetCorrection setting to the camera. The function returns SNAPERROR_NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The offset correction state of the camera will be disabled if **bOffsetCorr** is equal to 0, or enabled if **bOffsetCorr** is equal to 1 (or any other non-zero value).

See also **SnapCmd_GetOffsetCorrection().**

SnapCmd_SetOffsetGain

short **SnapCmd_SetOffsetGain(**short **nOffsetGain);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_INVALIDPARAMETER or SNAPERROR_IOERROR on failure.

Parameters

nOffsetGain variable containing the new OffsetGain setting.

Description

Writes a new OffsetGain setting to the camera. The function returns SNAPERROR_ NOERROR if the value was written successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera. A return value of SNAPERROR_ INVALIDPARAMETER indicates that the value passed by **nOffsetGain** was out of range (less than 0 or greater than 2).

Since the offset image is scaled at the time it is acquired, changing the OffsetGain setting renders the current offset image invalid. Offset correction will be disabled until a new offset image has been acquired.

See also **SnapCmd_GetOffsetGain()**.

SnapCmd_SetResetState

short **SnapCmd_SetResetState(**char **bRESET);**

Return Value

SNAPERROR_NOERROR if successful or SNAPERROR_IOERROR on failure.

Parameters

bRESET variable containing the new RESET setting.

Description

Writes a new RESET setting to the camera. The function returns SNAPERROR_ NOERROR if the value was retrieved successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera.

The global reset input to the sensor will be disabled if **bRESET** is equal to 0, or enabled if **bRESET** is equal to 1 (or any other non-zero value).

See also **SnapCmd_GetResetState().**

SnapCmd_SetTimingMode

short **SnapCmd_SetTimingMode(**short **nTimingMode);**

Return Value

SNAPERROR_NOERROR if successful;

SNAPERROR_INVALIDPARAMETER or SNAPERROR_IOERROR on failure.

Parameters

nTimingMode variable containing the new TimingMode setting.

Description

Writes a new TimingMode setting to the camera. The function returns SNAPERROR_ NOERROR if the value was written successfully, or SNAPERROR_IOERROR if the function was unable to communicate with the camera. A return value of SNAPERROR_ INVALIDPARAMETER indicates that the value passed by **nTimingMode** was out of range (less than 0 or greater than 3).

See also **SnapCmd_GetTimingMode()**.

Appendices

A. SetupInfo Data

The "SETUP.TXT" file on the camera drive provides current information about the camera status. The default setup information looks as follows:

`ShadoSnap #0001` ..... serial number of attached camera
`Valid Image:  0` ..... valid image flag (0=no, 1=yes)
`Valid Offset:  0` ..... valid offset flag (0=no, 1=yes)
`Valid Pixmap:  0` ..... valid pixel map flag (for future expansion)
`Offset Corr.:  0` ..... offset correction flag (0=off, 1=on)
`Pixel Corr.:  0` ..... pixel correction flag (for future expansion)
`Image Scale:  2` ..... image gain (see **SnapCmd_GetImageGain**)
`Offset Scale:  0` ..... offset gain (see **SnapCmd_GetOffsetGain**)
`RESET:  0` ..... RESET state (0=disabled, 1=enabled)
`BIN:  0` ..... BIN state (for future expansion)
`NDR:  0` ..... NDR state (for future expansion)
`Int.Time: 00800h` ..... integration time counter setting (hex, 512µs/count)
`Timing Mode:  0` ..... timing mode (see **SnapCmd_GetTimingMode**)

B. Error Codes

The following error codes are defined in the Shad-o-Snap SDK:

SNAPERROR_NOERROR	No error has occurred.
SNAPERROR_NOCAMERA	The **Snap_OpenCamera()** function was unable to locate a Shad-o-Snap camera attached to the computer.
SNAPERROR_IOERROR	The called SDK function was unable to communicate with the attached camera.
SNAPERROR_OPENTHMBNAILFILE	The **Snap_LoadThumbnail()** function was unable to open the "THMBNAIL.TIF" file on the Shad-o-Snap camera drive.
SNAPERROR_OPENIMAGEFILE	The called **Snap_Load...()** function was unable to open the requested image on the Shad-o-Snap camera drive.
SNAPERROR_INVALIDPARAMETER	The parameter value passed to the called **Snap_Set...()** function is outside of the specified valid range.
SNAPERROR_NOVALIDIMAGE	The image memory in the Shad-o-Snap camera contains invalid data.
SNAPERROR_NOVALIDOFFSET	The offset image memory in the Shad-o-Snap camera contains invalid data.
SNAPERROR_NOTENOUGHMEMORY	The **Snap_OpenCamera()** function was unable to allocate sufficient memory resources to initialize the SDK functions.
SNAPERROR_CALLNOTSUPPORTED	The called SDK function is not supported in this version of the Shad-o-Snap SDK.
SNAPERROR_INVALIDERRCODE	The error code passed to the **Snap_ ErrorMessage()** function does not match one of the error codes listed in this appendix.

C. Sample Program Listing

The following program is a simple Win32 console application to test the Shad-o-Snap interface. The program initializes the Shad-o-Snap SDK, acquires the next available image from the camera, and saves the thumbnail TIFF image to disk.

```
#include <windows.h>
#include <sys/timeb.h>
#include "SnapSDK.h"

main()
{
     short errCode;
     char  errMessage[SNAP_MAXERRORMESSAGE];

     // Initialize the Shad-o-Snap SDK
     if ( errCode = Snap_OpenCamera() ) {
          Snap_ErrorMessage(errCode, errMessage);
          MessageBox(NULL, errMessage, "Shad-o-Snap SDK", MB_ICONERROR);
          return 0; }

     // Set up thumbnail image buffer
     unsigned long  nBufSize = 16384;
     unsigned char* pImgBuf = (unsigned char*)malloc(nBufSize);
     if ( pImgBuf == NULL ) {
          MessageBox(NULL, "Insufficient memory available.",
               "Shad-o-Snap SDK", MB_ICONERROR);
          Snap_CloseCamera();
          return 0; }

     // Wait for user input, then grab next available image
     MessageBox(NULL, "Ready to acquire image...",
               "Shad-o-Snap SDK", MB_ICONINFORMATION);
     if ( errCode = SnapCmd_AcquireImage() ) {
          Snap_ErrorMessage(errCode, errMessage);
          MessageBox(NULL, errMessage, "Shad-o-Snap SDK", MB_ICONERROR);
          Snap_CloseCamera();
          free(pImgBuf);
          return 0; }

     // Poll camera until image is valid
     _timeb time;
     double dtime1, dtime2;
     int nCount = 0;
```

```
    while ( errCode = SnapCmd_GetImageState() ) {
        // Check if I/O error occurred
        if ( errCode != SNAPERROR_NOVALIDIMAGE ) {
            Snap_ErrorMessage(errCode, errMessage);
            MessageBox(NULL, errMessage, "Shad-o-Snap SDK", MB_ICONERROR);
            break; }
        // Check if timout occurred
        if ( nCount++ == 10 ) {
            MessageBox(NULL, "Image acquisition timed out.",
                "Shad-o-Snap SDK", MB_ICONERROR);
            break; }
        // Wait 500 ms
        _ftime(&time);
        dtime1 = time.time + time.millitm / 1000.0;
        dtime2 = dtime1;
        while ( (dtime2 - dtime1) < 0.5 ) {
            _ftime(&time);
            dtime2 = time.time + time.millitm / 1000.0; }
        }

    // Retrieve thumbnail image and save to disk
    unsigned long nBytesWritten;
    if ( errCode = Snap_LoadThumbnail(pImgBuf, &nBufSize) ) {
        Snap_ErrorMessage(errCode, errMessage);
        MessageBox(NULL, errMessage, "Shad-o-Snap SDK", MB_ICONERROR); }
    else {
        HANDLE hFile = CreateFile("Thmbnail.tif", GENERIC_WRITE,
                                  FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                                  FILE_ATTRIBUTE_NORMAL, NULL);
        if ( hFile != INVALID_HANDLE_VALUE ) {
            WriteFile(hFile, pImgBuf, nBufSize, &nBytesWritten, NULL);
            CloseHandle(hFile); }
        }

    // Close SDK and release resources
    Snap_CloseCamera();
    free(pImgBuf);
    return 1;
}
```