SENSORAY CO., INC.

SX11
Software Development Kit
version 2.10

January 2002

7337 SW Tech Center Dr.
Tigard, OR 97223
Phone 503.684.8073 • Fax 503.684.8164
sales@sensoray.com
www.sensoray.com

Table of Contents

Introduction

The SX11 Software Development Kit (SDK) v.2.10 supports a range of Sensoray's software compatible frame grabbers:

- models 311 (PC/104+), 611 (PCI) and 711 (CompactPCI) (revisions A-C, Bt848 decoder);
- models 311, 611 and 711 (revision D, Fusion878 decoder);
- model 1101 (CPU/frame grabber).

The SDK v.2.10 supports Windows98/NT/2000/XP. Please contact Sensoray if you need support for other operating systems.

The API had not changed compared to the previous version of the SDK. Some of the data structures, however, were modified in order to implement certain enhancements, which requires re-compiling your source code with the new header files. Please read the "Building an application with sx11.dll" section for the details on application creation. It contains important information on the differences from the earlier SDK versions.

As in the previous SDK, Basic and Enhanced versions are available. The Enhanced SDK supports the following features (not supported in the Basic version):

- capture into video memory using DirectDraw;
- mapping a capture buffer on the external buffer;
- bi-modal capture (one field in color, the other in monochrome).

The following differences exist compared to SDK versions 1.x:

- the source code for all sample applications is now provided with the Basic SDK;
- the Enhanced SDK is now a complete package, rather than an addition to the Basic SDK. Only one of two needs to be installed. However, if the Enhanced SDK is purchased after the Basic SDK has been already installed, it is recommended to install it into the same directory to avoid unnecessary file duplication.

The sample applications provided with the SDK offer examples of using various features, and also could be used for fast hardware testing.

The following sample applications are included in the SDK:

- Sample1 – captures and displays the image in a window. Uses X11_Acquire function. Provides 15 frames per second capture. Executable is included.
- Sample2 - captures and displays the image in a window. Uses X11_StartAcquire function and a dual image buffer. Provides close to 30 frames per second capture. Executable is included.

- Sample3 - captures and displays the image in a window. Uses interrupts. Executable is included.
- Sample4 – captures the image directly into video card's memory and displays it using DirectDraw with optional overlays. Executable is included only in the Enhanced SDK.
- Sample5 – illustrates the use of bi-modal capture (one field is captured in color, the other in monochrome). Executable is included only in the Enhanced SDK.
- Sample6 – illustrates capture into the buffer allocated by another software package (e.g. Matrox Imaging Library). In order to compile this sample you need to have MIL v.4 or higher installed on your PC. The executable is not provided.
- Visual Basic sample application – illustrates the use of sx11.dll with VB.

Software Installation

It is recommended to install the SDK before installing the board(s) into your PC. To install the SDK run setup.exe from the installation disk.

Windows98

The first time you boot your system after model X11 frame grabber is plugged in, Windows will launch "Add new hardware Wizard". Select "Display a list of all drivers", click "Next", check "Floppy drive" and uncheck all other check boxes, and insert a SX11 SDK v.2.10 distribution disk into the floppy drive. Click "Next". Windows should automatically complete the rest of the installation procedure. You may see another "Found new hardware" message, but you should not be prompted any more.

WindowsNT

To install the SDK run setup.exe from the installation disk.

Windows2000/XP

The first time you boot your system after model X11 frame grabber is plugged in, Windows will launch "Found new hardware Wizard". Select "Install from a list...", click "Next", check “Include this location in the search”, click on “Browse”, and select a floppy drive. Insert a SX11 SDK v.2.10 distribution disk into the floppy drive and click “OK”. When a message appears saying that the driver does not have a Microsoft digital signature, click “Continue anyway”. Windows will display another "Found new hardware Wizard" immediately, repeat all steps same as above.

Note: under WindowsXP installing the driver before the SDK is installed may result in failure. In this case “Found new hardware...” message is not displayed any more. To install the driver properly go to Control Panel – System – Hardware – Device manager, locate the Sensoray X11 board under “Sound, video and game controllers” or “Other devices”, right click on it, select “Properties”, select “Driver” tab, and click on “Update driver”. Follow the steps described above.

Building an application with sx11.dll

Files to be included in the project

The following files are distributed with the SDK:

- `sx11.h` – contains data types and constants definitions;
- `sx11f.h` – contains exported functions declarations;
- `sx11app.c` – contains exported functions and helper functions definitions.

When building an application with `sx11.dll`, it is necessary to include `sx11app.c` in the project.

IMPORTANT:
If MFC is used, an option "Not using precompiled headers" must be set in the project settings for `sx11app.c`.

All modules containing calls to the `sx11.dll` functions must include `sx11f.h`. Please refer to the sample source code for an example of building an application with SX11 SDK v.2.10.

Differences compared to SDK v.2.00

The *PCIslot[i]* member of PCI structure now returns board type in the upper 2 bytes. Please refer to the description of the PCI structure.

Data types reference

System structure PCI

```
typedef struct {
    DWORD boards;
    DWORD PCIslot[SYS_GRABBERS];
} PCI;
```

The PCI structure contains information about the frame grabber boards identified by the system.

boards
Number of supported boards identified by the system.

PCIslot
Array of slot numbers. The lower 2 bytes of `PCIslot[i]` contain the slot number. The upper 2 bytes of `PCIslot[i]` contain board revision code:

All boards with Bt848 chip (revision C and earlier)	- 0;
311 rev.D ,E	- 0x10;
611 rev.D	- 0x20;
711 rev.D	- 0x30;

Revision codes equal or above 0x100 are reserved for other boards which may not be compatible with SX11 SDK.

The PCI structure is initialized by X11_InitSystem function. The system constant SYS_GRABBERS determines the maximum number of frame grabbers supported, and is defined in `sx11.h`. *PCIslot* member represents a PCI slot number for a given board. The PCI slot number is generated by Windows, and usually is not the same as the ordinal number of the slot. The PCI slot number is used as a parameter to X11_GetHFG function to obtain a frame grabber handle.

Image buffer structure BUFFER

```
typedef struct {
    HBUF hbuf;
    DWORD dwFrames;
    FRAME frame[SYS_FRAMES];
} BUFFER;
```

The BUFFER structure contains information about the image buffer allocated by the system. Image buffer may consist of one or more image frames. Acquisition functions always fill all the frames of an image buffer with the data corresponding to consecutively grabbed video frames.

hbuf
Image buffer handle.

dwFrames
Number of frames in the image buffer.

frame[SYS_FRAMES]
Array of FRAME structures.

The BUFFER structure is initialized by X11_AllocBuffer function.

Frame data structure FRAME

```
typedef struct {
    LPBITMAPINFO lpbmi;
    void *lpvbits;
} FRAME;
```

The FRAME structure contains information about each individual frame of the image buffer allocated by the system.

lpbmi
Pointer to the BITMAPINFO structure which defines the dimensions and color information for a device-independent bitmap (DIB) associated with the frame.

lpvbits
Pointer to the frame image data.

When an image buffer is allocated, every frame is associated with a Windows DIB, and a corresponding data structure is created. This simplifies the display of acquired images using Windows API. However, lpvbits could be used as a general purpose pointer to the frame data. See Windows API Help for the description of bitmaps, related data structures, and display functions.

Note: the BITMAPINFO structure is created even if the buffer is created as a "flat" buffer (see the description of *store* member in MODE structure). In such a case Windows API functions will display the image upside down.

Interrupt data structure INT_DATA

```
typedef struct {
    HFG hfg;
    DWORD mask;
    DWORD status;
    FPTR func;
    int priority;
    DWORD total;
    DWORD lost;
} INT_DATA;
```

Interrupt data structure contains the information necessary to facilitate interrupts use.

hfg
board handle.

mask
interrupt mask: one or more of the status constants defined in `sx11.h` (except for STATUS_READY_ALL). For example, setting *mask* to STATUS_READY | STATUS_GPINT results in the interrupt being generated on the acquisition completion or interrupt condition on GPIO interrupt pin. See the description of X11_InterruptOn function for the details.

status
interrupt status: shows the source of the interrupt.

func
a pointer to the user interrupt processing function that will be called once the interrupt has occurred (see `Sample3.c` for the example).

priority
interrupt processing thread priority. Can be one of the following: THREAD_PRIORITY_LOWEST, THREAD_PRIORITY_BELOW_NORMAL, THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_ABOVE_NORMAL, THREAD_PRIORITY_HIGHEST, THREAD_PRIORITY_TIME_CRITICAL.

total
total number of interrupts that occurred since X11_InterruptUnmask was called.

lost
total number of interrupts that were lost (due to processing delays).

Operation mode data structure MODE

```
typedef struct {
    DWORD scale;
    DWORD color;
    DWORD store;
    DWORD input;
    MODE_ADVANCED advanced;
} MODE;
```

The MODE structure contains information about the operation mode of the frame grabber. The settings that are not likely to be used frequently are hidden inside the *advanced* member.

scale
Defines image scale. Can be one of the following:

SCALE_ADVANCED	Image scale is defined by the settings in the MODE_ADVANCED structure.
SCALE8	Full size image.
SCALE6	3/4 size image.
SCALE4	1/2 size image.
SCALE2	1/4 size image.

color
Defines output color format of the image. Can be one of the following:

COLOR_MONO	Monochrome image, 1 byte per pixel.

COLOR_RGB	Color image, 3 bytes per pixel.
COLOR_YCRCB	YCrCb (4:2:2), 2 bytes/pixel;
COLOR_RGB32	RGB32 color format, 4 bytes/pixel;
COLOR_RGB16	RGB16 color format, 2 bytes/pixel;
COLOR_RGB15	RGB15 color format, 2 bytes/pixel;

store

Defines the way the image is stored in memory. Can be one of the following:

STORE_DIB	Image is stored as a Windows DIB (reversed lines order).
STORE_FLAT	Image is stored with normal lines order.

input

Controls the input multiplexor. Can be one of the following:

MUX_0	S-Video input.
MUX_1	Video 1 input.
MUX_2	Video 2 input.
MUX_3	Video 3 input (supported only on rev.D boards).
MUX_4	Video 4 input (supported only on rev.D boards, can only be used when S-video signal is not connected).

advanced

MODE_ADVANCED structure. Defines the advanced mode settings.

Advanced operation mode data type MODE_ADVANCED

```
typedef struct {
    DWORD interlace;
    DWORD xTotal;
    DWORD xActive;
    DWORD xDelay;
    float yFactor;
    DWORD yActive;
    DWORD yDelay;
    DWORD FORMAT;
    DWORD BRIGHT;
    DWORD CONTRAST;
    DWORD SAT_U;
    DWORD SAT_V;
    DWORD HUE;
    DWORD LNOTCH;
    DWORD LDEC;
    DWORD DEC_RAT;
    DWORD PEAK;
    DWORD CAGC;
    DWORD CKILL;
    DWORD HFILT;
    DWORD RANGE;
    DWORD CORE;
    DWORD YCOMB;
    DWORD CCOMB;
    DWORD ADELAY;
    DWORD BDELAY;
    DWORD SLEEP;
    DWORD CRUSH;
    DWORD VFILT;
    DWORD COLOR_BARS;
    DWORD GAMMA;
    DWORD PKTP;
    DWORD bimodal;
    DWORD colorkey;
    DWORD buffertype;
    DWORD gpintmode;
    DWORD reserved1;
    DWORD reserved2;
    DWORD reserved3;
    DWORD reserved4;
} MODE_ADVANCED;
```

The MODE_ADVANCED structure contains information about the advanced operation mode of the frame grabber.

interlace
Defines input image format. Can be one of the following:
IMG_INTERLACED Interlaced input image.

IMG_NONINTERLACED Noninterlaced input image.

This parameter affects how the vertical scaling is performed for scaling below ½ of the normal size: with IMG_INTERLACED the scaling is performed using both fields, with IMG_NONINTERLACED only one field is used. The recommended setting in this case is IMG_NONINTERLACED.

xTotal

Total number of output horizontal pixels (including horizontal blanking). Must be between 100 and 910 (NTSC), or 1135 (PAL). This is the number of pixels generated by the frame grabber internally.

xActive

Number of active output horizontal pixels. Must be between 80 and 900 (NTSC), or 1000 (PAL). This is the number of pixels in the output image.

xDelay

The horizontal offset of the start of the active area relative to the horizontal sync, pixels. The following condition must always be met: xDelay + xActive <= xTotal.

yFactor

Vertical scaling factor. The actual number of lines generated by the video source (525 for NTSC, 625 for PAL) is divided by yFactor to get the number of lines in the output image. Must be between 1.0 and 8.0.

yActive

Number of active output lines (before vertical scaling is applied). Must be between 60 and 480 (NTSC), or 576 (PAL). For example, to grab the whole NTSC image scaled down by the factor of 2, set yActive to 480, yFactor to 2.0. To grab the upper half of the NTSC image scaled down by the factor of 2, set yActive to 240, yFactor to 2.0.

yDelay

The vertical offset of the start of the active area relative to the vertical sync, lines (before vertical scaling is applied). The following condition must always be met:
yDelay + yActive <= (total lines, 525 or 625).

FORMAT

Input signal format. Can be one of the following:

FORMAT_NTSC	NTSC input signal.
FORMAT_NTSCJ	NTSC (Japan) input signal.
FORMAT_PAL	PAL input signal.
FORMAT_PALM	PAL(M) input signal.
FORMAT_PALN	PAL(N) input signal.
FORMAT_PALNC	PAL(N-combination) input signal¥.
FORMAT_SECAM	SECAM input signal.

BRIGHT

Controls the brightness (luminance) of the output signal. Takes the values between 0 and 0xFF which are treated as a signed offset, from -128 (0x80) to +127 (0x7F). The resolution of brightness change is one LSB (0.4% of the full range).

CONTRAST

This 9-bit value is multiplied by the luminance value to provide contrast (gain) adjustment. Takes values from 0 to 0x1FF (237%), with 0x0D8 corresponding to 100%.

SAT_U

A 9-bit value used to add a gain adjustment to the U component of the video signal. By adjusting U and V color components by the same incremental value, the saturation is adjusted. Takes values between 0 and 0x1FF (201%), with 0x0FE corresponding to 100%.

SAT_V

A 9-bit value used to add a gain adjustment to the V component of the video signal. By adjusting U and V color components by the same incremental value, the saturation is adjusted. Takes values between 0 and 0x1FF (284%), with 0x0B4 corresponding to 100%.

HUE

Controls the hue by adjusting the demodulating subcarrier phase. Takes values between 0 and 0xFF, which are treated as a signed offset with 0x80 corresponding to -90 degrees, and 0x7F corresponding to +89 degrees.

LNOTCH

Controls the internal luminance notch filter which attenuates the subcarrier in the output signal, removing the "chess board" pattern in the output image, in case a composite input is used. Can be one of the following:

LNOTCH_OFF	Filter disabled.
LNOTCH_ON	Filter enabled.

It is recommended to use LNOTCH in conjunction with CCOMB.

LDEC

Controls the luminance decimation filter used to reduce the high-frequency components of the luma signal. Useful when scaling down to lower resolutions. See HFILT for details. Can be one of the following:

LDEC_OFF	Filter disabled.
LDEC_ON	Filter enabled.

DEC_RAT

A 6-bit value corresponding to the number of frames dropped out of 60 (NTSC) or 50 (PAL/SECAM). A value of 0 disables decimation.

PEAK

Determines whether the normal or the peaking luma low pass filters are implemented via the HFILT. Can be one of the following:

PEAK_OFF	Normal low pass filters.
PEAK_ON	Peaking low pass filters.

CAGC

Controls the chroma AGC function. When enabled, will compensate for nonstandard chroma levels by multiplying the incoming chroma signal by a value in the range of 0.5 to 2.0. Can be one of the following:

CAGC_OFF	Chroma AGC off.
CAGC_ON	Chroma AGC on.

CKILL

Controls the low color detector and removal circuitry. Can be one of the following:

CKILL_OFF	Low color detection and removal disabled.
CKILL_ON	Low color detection and removal enabled.

HFILT

Controls the degree of horizontal low-pass filtering provided LDEC is set to LDEC_ON. Can be one of the following:

HFILT_AUTO	The filter is selected automatically depending on the scale setting. When horizontal scaling is between full and half resolution, no filtering is selected. When scaling between one-half and one-quarter resolution, the CIF filter is used. When scaling between one-quarter and one-eighth resolution, the QCIF filter is used. When scaling below one-eighth resolution, the ICON filter is used.
HFILT_CIF	CIF filter.
HFILT_QCIF	QCIF filter.
HFILT_ICON	ICON filter.

RANGE

Determines the range of the luminance output. Can be one of the following:

RANGE_NORM	Normal operation (luma range 16-253).
RANGE_FULL	Full range operation (luma range 0-255).

CORE

Controls the coring value. When coring is enabled, luminance levels below a certain value are truncated to 0. Can be one of the following:

CORE_OFF	Coring disabled.
CORE_8	Coring threshold is 8.
CORE_16	Coring threshold is 16.
CORE_24	Coring threshold is 24.

YCOMB

Controls the luminance comb filtering. Can be one of the following:

YCOMB_OFF	Vertical low-pass filtering and vertical interpolation.
YCOMB_ON	Vertical low-pass filtering only. The number of filter taps is determined by VFILT setting.

CCOMB

Controls the chrominance comb filtering. Can be one of the following:

CCOMB_OFF	Chroma filter disabled.
CCOMB_ON	Chroma filter enabled.

ADELAY

Back-porch sampling delay. The default values are 0x68 (NTSC), and 0x7F (PAL/SECAM).

BDELAY

Subcarrier sampling delay. The default values are 0x5D (NTSC), and 0x73 (PAL/SECAM).

SLEEP

Controls sleep mode of luma and chroma A/D's. Can take the following values:

SLEEP_OFF	Both A/D's operating.
Y_SLEEP	Luma A/D in sleep mode.
C_SLEEP	Chroma A/D in sleep mode. Y_SLEEP and C_SLEEP can be OR'ed, to disable both A/D's.

CRUSH

Controls the AGC mode. Can be one of the following:

CRUSH_OFF	Nonadaptive AGC.
CRUSH_ON	Adaptive AGC. Overflows in A/D's result in the input voltage range increase.

VFILT

Controls the number of taps in the vertical scaling filter. Can be one of the following:

If YCOMB is set to YCOMB_ON:

VFILT_0	2-tap filter.
VFILT_1	3-tap filter. Only available if scaling to less then 385 horizontal active pixels.
VFILT_2	4-tap filter. Only available if scaling to less then 193 horizontal active pixels.
VFILT_3	5-tap filter. Only available if scaling to less then 193 horizontal active pixels.

If YCOMB is set to YCOMB_OFF:

VFILT_0	2-tap interpolation only.
VFILT_1	2-tap filter and 2-tap interpolation. Only available if scaling to less then 385 horizontal active pixels.
VFILT_2	3-tap filter and 2-tap interpolation. Only available if scaling to less then 193 horizontal active pixels.
VFILT_3	4-tap filter and 2-tap interpolation. Only available if scaling to less then 193 horizontal active pixels.

COLOR_BARS

Controls a test color bar pattern. Can be one of the following:

COLORBARS_OFF	Color bars off.
COLORBARS_ON	Color bars on.

GAMMA

Controls gamma correction removal. Can be one of the following:

GAMMA_REMOVE_ON	Gamma correction removal on.
GAMMA_REMOVE_OFF	Gamma correction removal off.

PKTP

FIFO trigger point. May affect PCI transfer performance. Can be one of the following:

PKTP4	4 DWORDs.
PKTP8	8 DWORDs.
PKTP16	16 DWORDs.
PKTP32	32 DWORDs.

bimodal

Controls whether the acquisition mode is normal (both fields are captured in the same color format), or bimodal (second field is captured in monochrome). Can be one of the following:

BIMODAL_OFF	normal mode
BIMODAL_ON	bimodal mode.

Bimodal capture allows acquisition of 2 fields of the same interlaced frame in different color formats: the first (odd) being captured in any format specified by MODE.color setting, the second (even) being monochrome (1 byte/pixel). This capture mode could be beneficial for applications that have to locate an object within the image, which is easier to do in monochrome, and then analyze color information of the located object using the coordinates obtained during the first step. See Sample5 for implementation details.

colorkey

Color key value for overlays using DirectDraw. All pixels of this color on the primary surface are replaced with the corresponding pixels of the captured image. See X11_AllocBuffer function description for the details.

buffertype

Type of buffer to allocate. Can be one of the following:

BUF_MEM	regular memory buffer.
BUF_EXT	external buffer.
BUF_VIDEO	video memory buffer.

See X11_AllocBuffer function description for the details.

gpintmode

Controls the GPIO port interrupt pin operation mode. This parameter is a logical OR of two values that control whether the interrupt is edge or level sensitive (GPINT_LEVEL or GPINT_EDGE), and whether the signal coming from the interrupt pin is inverted or not (GPINT_INV or GPINT_NONINV).

reserved1(2,3,4)

Reserved. Do not modify.

Functions reference

The SX11 SDK is designed to provide the application developer with full control over the frame grabber, yet it is contains just 21 functions.

All special data types used by the DLL are defined in `sx11.h`. The sample applications provided with the SDK illustrate the use of most of the functions and allow building a custom application within minutes.

X11_InitSystem

```
ECODE X11_InitSystem (
    PCI *pPci         //pointer to a PCI structure
);
```

Parameters

pPci
Pointer to the structure of PCI type.

Return values

Returns 0 in case of success, or an error code (a list of error codes is included in sx11.h).

Notes

The function initializes the driver, searches for and initializes all boards supported by the SDK. This function has to be called **only once** per application. The system resources allocated by X11_InitSystem are released by a call to X11_CloseSystem.

The SDK supports multiple boards and allows multiple applications to access the boards. The SDK functions are “thread safe”, i.e. hardware accesses are protected from being interrupted by another thread. However care must be taken on the application level to prevent conflicting commands from being issued from different threads (processes) and to provide necessary synchronization. Generally it is not recommended to access one board from different processes.

X11_GetHFG

```
ECODE X11_GetHFG (
    HFG *pHfg,          //pointer to a frame grabber handle
    DWORD slot          //board's PCI slot number
);
```

Parameters

pHfg
Pointer to a variable receiving the handle value.

slot
Board's PCI slot number returned by X11_InitSystem.

Return values

Returns 0 in case of success, or an error code.

X11_AllocBuffer

```
ECODE X11_AllocBuffer (
    MODE *pMode,      //pointer to a MODE structure
    BUFFER *pBuffer,  //pointer to a BUFFER structure
    DWORD dwParameter //parameter depending on the buffer type
);
```

Parameters

pMode

Pointer to a variable of MODE type defining the current frame grabber operation mode. MODE has to be set up prior to calling X11_AllocBuffer to define the buffer size and properties. In case any changes are made to MODE affecting scaling, color format, or storage type, the image buffer has to be re-allocated. Other changes to MODE that do not affect the buffer size (e.g. input channel or various signal filtering options) do not require buffer re-allocation.

pBuffer

Pointer to a variable of BUFFER type receiving the allocated buffer data in case the call is successful.

dwParameter

Parameter which depends on MODE.advanced.buffertype setting.

If the buffer type is a regular memory buffer (MODE.advanced.buffertype=BUF_MEM), *dwParameter* specifies the number of frames in the image buffer (see the description of frames further in this section). The value of *dwParameter* must be between 1 and SYS_FRAMES.

If the buffer type is an external buffer (MODE.advanced.buffertype=BUF_EXT) (allocated by a function other than X11_AllocBuffer, for example, an image processing library), *dwParameter* is a pointer to this buffer. See Sample6 for implementation details.

If the buffer is allocated in the video memory (MODE.advanced.buffertype=BUF_VIDEO), *dwParameter* is a handle of the image display window. See Sample4 for implementation details.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_AllocBuffer allocates a buffer to which an image is captured. X11 software supports buffers with multiple frames. The difference between allocating multiple buffers and allocating a single buffer with multiple frames is that all frames of a buffer are being filled with image data from **consecutive** video frames by a **single** call to an image acquisition function. That means that by allocating a multiframe buffer one can guarantee that no input video frames are missed during the acquisition of a frame sequence. See `sx11.h` for BUFFER and FRAME types definitions.

Once a buffer is allocated, it is referenced by a handle that is returned by X11_AllocBuffer (as a member of BUFFER structure).

X11_FreeBuffer

```
void X11_FreeBuffer (
    HBUF hbuf         //buffer handle
)
```

Parameters

hbuf
A handle to the buffer being released.

Return values

None.

Notes

X11_FreeBuffer releases all memory resources associated with the buffer. All open buffers are closed automatically with a call to X11_CloseSystem. It is not necessary to re-allocate a buffer unless the MODE parameters affecting the buffer size are changed.

X11_CloseSystem

```
void X11_CloseSystem (
    void
)
```

Parameters

None.

Return values

None.

Notes

X11_CloseSystem releases all resources allocated by the calls to X11 functions. It is important that X11_CloseSystem is not called while image capture is in progress. Please make sure all acquisition functions have completed before calling this function.

In case of multiple processes having called X11_InitSystem, the last process to call X11_CloseSystem will actually release the resources.

X11_Acquire

```
ECODE X11_Acquire (
    HFG hfg,          //frame grabber handle
    HBUF hbuf,        //buffer handle
    float timeout,    //acquisition timeout value, seconds
    DWORD *pStatus    //pointer to status variable
)
```

Parameters

hfg
A handle to the frame grabber.

hbuf
A handle to the image buffer.

timeout
Acquisition timeout in seconds. The function returns if the acquisition is not completed after (*timeout*) seconds.

pStatus
Address of the variable receiving the status value. The individual bits are set if a corresponding condition occurs during the acquisition of any frame of the image buffer.

Return values

Returns 0 in case of success, or an error code.

Notes

The function returns **after** the acquisition is complete, or timeout expires. The function also sets the variable pointed to by *pStatus* with the value of the status word corresponding to the end of the acquisition of the last frame. Status bits are not reset automatically between the acquisition of the individual frames. See X11_GetStatus for the description of the constants used to select individual status bits.

X11_StartAcquire

```
ECODE X11_StartAcquire (
    HFG hfg,          //frame grabber handle
    HBUF hbuf,        //buffer handle
    DWORD dwAcqmode   //acquisition mode
)
```

Parameters

hfg
A handle to the frame grabber.

hbuf
A handle to the image buffer.

dwAcqmode
Acquisition mode: determines whether the image buffer is filled just once (AMODE_SINGLE), or continuously, until the acquisition is stopped by X11_StopAcquire (AMODE_CONT).

Return values

Returns 0 in case of success, or an error code.

Notes

X11_StartAcquire returns immediately after the acquisition is started. The application determines whether the acquisition is complete by polling status bits, or through the use of interrupts. If continuous mode is selected, the first frame of the image buffer starts being overwritten as soon as the last frame gets filled. The application determines the completion of each individual frame by polling (and resetting) the STATUS_READY bit, or through the use of the interrupts. The STATUS_READY_ALL bit is set upon the completion of the last frame of the image buffer. If AMODE_CONT is selected, the STATUS_READY_ALL bit can not be polled reliably, because it is being reset internally at the start of the first frame acquisition. There is no interrupt associated with the STATUS_READY_ALL bit.

X11_StopAcquire

```
ECODE X11_StopAcquire (
    HFG hfg           //frame grabber handle
)
```

Parameters

hfg
A handle to the frame grabber.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_StopAcquire function stops the image acquisition asynchronously. It is used only when acquisition in progress has to be interrupted (usually when continuous acquisition mode is used). There is no need to call X11_StopAcquire after the acquisition is complete.

X11_GetStatus

```
ECODE X11_GetStatus (
    HFG hfg,          //frame grabber handle
    DWORD *pStatus    //pointer to status variable
)
```

Parameters

hfg
A handle to the frame grabber.
pStatus
Address of the variable receiving the status value.

Return values

Returns 0 in case of success, or an error code.

Notes

The individual bits of the status word have the following meanings (corresponding to bit values of 1):

STATUS_READY	Frame acquisition complete.
STATUS_READY_ALL	Image buffer acquisition complete. Reset automatically at the start of the first frame acquisition.
STATUS_VIDEO	Video status changed at the input (e.g. present to absent).
STATUS_HLOCK	Horizontal lock condition changed at the input.
STATUS_OFLOW	Overflow detected.
STATUS_HSYNC	Start of new line.
STATUS_VSYNC	Start of new field.
STATUS_FMT	Video format change detected (e.g. NTSC to PAL).
STATUS_ERROR	Transfer error occurred. This is a combination of bits.

All status bits except STATUS_READY_ALL have to be reset by the application. See X11_ResetStatus.

X11_ResetStatus

```
ECODE X11_ResetStatus (
    HFG hfg,          //frame grabber handle
    DWORD dwMask      //reset mask
)
```

Parameters

hfg
A handle to the frame grabber.
dwMask
A bit value of 1 resets the corresponding status bit.

Return values

Returns 0 in case of success, or an error code.

Notes

Resets the bits of the status register.

X11_GetStatusEx

```
ECODE X11_GetStatusEx (
    HFG hfg,          //frame grabber handle
    STATUS *pStatus   //pointer to status variable
)
```

Parameters

hfg
A handle to the frame grabber.

pStatus
Address of the variable (of STATUS type) receiving the status value.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_GetStatusEx returns extended status information (it replaces the X11_GetDStatus function of the older SDK versions). The statusA member of STATUS structure is set to the same value as that returned by X11_GetStatus function. The statusB member is set to the value with the following meanings of the individual bits (corresponding to bit values of 1):

DSTATUS_PRES	Video present.
DSTATUS_HLOC	Device in horizontal lock.
DSTATUS_FIELD	Even field (0 corresponds to the odd field).
DSTATUS_NUML	625 line format (PAL/SECAM). The value of 0 corresponds to the 525 line format (NTSC/PAL-M).
DSTATUS_PLOCK	PLL out of lock. This bit has to be read and cleared until it is no longer set when switching between 525 and 625 line formats.
DSTATUS_LOF	Luma ADC overflow.
DSTATUS_COF	Chroma ADC overflow.

X11_ResetStatusEx

```
ECODE X11_ResetStatusEx (
    HFG hfg,          //frame grabber handle
    STATUS *pMask     //pointer to status mask
)
```

Parameters

hfg
A handle to the frame grabber.
pMask
Address of the variable (of STATUS type) defining the bits to be reset.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_ResetStatusEx resets selected bits of 2 status registers. The way the bits of 2 registers are affected by this function is different: the bits set to 1 in pStatus->statusA are reset to 0, the bit values of pStatus->statusB are written to the status register directly.

X11_SetMode

```
ECODE X11_SetMode (
    HFG hfg,          //frame grabber handle
    MODE *pMode       //pointer to MODE variable
)
```

Parameters

hfg
A handle to the frame grabber.

pMode
Address of the variable (of MODE type) defining the frame grabber operation mode.

Return values

Returns 0 in case of success, or an error code.

Notes

The value of the MODE variable pointed to by pMode can be modified as a result of the call to X11_SetMode. For example, if one of the predefined scale settings is used (SCALE8, etc.), the members of the advanced portion of MODE related to scaling are set accordingly.

X11_GetImageSize

```
ECODE X11_GetImageSize (
    MODE *pMode,      //pointer to the MODE variable
    DWORD *pxsize,    //pointer to DWORD variable
    DWORD *pysize     //pointer to DWORD variable
)
```

Parameters

pMode
Address of the variable (of MODE type) defining the frame grabber operation mode.
pxsize
Address of the variable receiving the horizontal image size, pixels.
pysize
Address of the variable receiving the vertical image size, pixels.

Return values

Returns 0 in case of success, or an error code.

Notes

The X11_GetImageSize function retrieves the dimensions of the image corresponding to the particular mode. It is convenient if some complex formatting options are used, resulting in nontrivial image dimensions. The retrieved values could be used to define the display window dimensions, for example.

X11_WritePort

```
ECODE X11_WritePort (
    HFG hfg,          //frame grabber handle
    DWORD value,      //value to write to port
    DWORD mask        //bitmask
)
```

Parameters

hfg
A handle to the frame grabber.

value
The value to be written to the output port.

mask
A value of 1 in a particular bit of *mask* allows modifying this bit.

Return values

Returns 0 in case of success, or an error code.

Notes

The X11_WritePort function writes to the 4-bit general purpose output port of the frame grabber. Only the lower 4 bits of *value* and *mask* are meaningful. Bits 0-3 correspond to the signals GPO0-GPO3, respectively.

X11_ReadPort

```
ECODE X11_ReadPort (
    HFG hfg,          //frame grabber handle
    DWORD *pValue     //pointer to DWORD variable
)
```

Parameters

hfg
A handle to the frame grabber.
pValue
An address of the variable receiving the value read from the input port.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_ReadPort reads a value from the 4-bit general purpose input port of the frame grabber. Bits 0-3 correspond to the signals GPI0-GPI3, respectively.

X11_InterruptOn

```
ECODE X11_InterruptOn (
    INT_DATA *pIntData      //pointer to INT_DATA variable
)
```

Parameters

pIntData
An address of the global INT_DATA variable defining interrupt parameters.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_InterruptOn enables interrupts handling. The following constants are used to enable/disable interrupt sources:

STATUS_READY	Frame acquisition complete.
STATUS_GPINT	GPIO interrupt.
STATUS_VIDEO	Video status changed at the input (e.g. present to absent).
STATUS_HLOCK	Horizontal lock condition changed at the input.
STATUS_OFLOW	Overflow detected.
STATUS_HSYNC	Start of new line.
STATUS_VSYNC	Start of new field.
STATUS_FMT	Video format change detected (e.g. NTSC to PAL).
STATUS_ERROR	Transfer error occurred. This is a combination of bits.

X11_InterruptOff

```
ECODE X11_InterruptOff (
    INT_DATA *pIntData      //pointer to INT_DATA variable
)
```

Parameters

pIntData
An address of the global INT_DATA variable defining interrupt parameters.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_InterruptOff disables the interrupt for a particular board and frees the resources. It has to be called prior to application termination in case X11_InterruptOn was called. Failing to call X11_InterruptOff may cause the system to hang.

X11_InterruptMask

```
ECODE X11_InterruptMask (
    INT_DATA *pIntData      //pointer to INT_DATA variable
)
```

Parameters

pIntData
An address of the global INT_DATA variable defining interrupt parameters.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_InterruptMask masks (disables) the interrupt sources set as 1's in mask member of INT_DATA structure.

X11_InterruptUnmask

```
ECODE X11_InterruptUnmask (
    INT_DATA *pIntData      //pointer to INT_DATA variable
)
```

Parameters

pIntData
An address of the global INT_DATA variable defining interrupt parameters.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_InterruptUnmask unmasks (enables) the interrupt sources set as 1's in mask member of INT_DATA structure.

X11_GetRC

```
ECODE X11_GetRC (
    HBUF hbuf,              //buffer handle
    DWORD frame,            //frame number within a buffer
    MODE *pmode,            //pointer to MODE structure
    DWORD rowcol,           //row or column flag
    DWORD rcnum,            //row or column number
    void *pArray            //pointer to array receiving data
)
```

Parameters

hbuf

A handle to the buffer from which the data is being retrieved.

frame

A number of a frame within a buffer being accessed (starting with 0).

pmode

A pointer to the variable of MODE type used to set the frame grabber mode.

rowcol

Access mode flag:

RMODE_ROW	Retrieves row data.
RMODE_COL	Retrieves column data.

rcnum

Row or column number (starting with 0).

pArray

Address of the external buffer (array) to copy the row (column) data into.

Return values

Returns 0 in case of success, or an error code.

Notes

X11_GetRC retrieves a row or a column from the image buffer and copies the data into the external buffer (array). The function is provided primarily for use with Visual Basic. In VB the pArray parameter that is passed to X11_GetRC has to be the address of the first element of an array, not an array itself (e.g. `dataarray (1)`, not just `dataarray`).