

**SX11 Frame Grabber Family
Software Reference**

Revised June 13, 2000

Table of contents

DATA TYPES	3
HFG	3
HBUF	3
ECODE	3
PCI	3
FRAME	3
BUFFER	4
MODE	4
MODE_ADVANCED	5
INT_DATA	10
FUNCTIONS	11
X11_InitSystem	11
X11_CloseSystem	11
X11_AllocBuffer	12
X11_FreeBuffer	13
X11_Acquire	14
X11_StartAcquire	15
X11_StopAcquire	16
X11_GetStatus	16
X11_ResetStatus	17
X11_GetDStatus	17
X11_SetDStatus	18
X11_SetMode	18
X11_InterruptOn	19
X11_InterruptOff	21
X11_InterruptMask	21
X11_InterruptUnmask	21
X11_GetHFG	22
X11_GetRC	22
X11_WritePort	23
X11_ReadPort	23
X11_GetImageSize	24
X11_GetApps	24

Data Types

HFG

A 32-bit value used as a handle to a frame grabber. All functions use the handle to access a particular board.

HBUF

A 32-bit value used as a handle to an image buffer. Acquisition functions use the handle to access a particular buffer.

ECODE

A 32-bit value used as an error code.

PCI

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

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

Member	Description
boards	Number of supported boards identified by the system.
PCISlot	Array of slot numbers.

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 PCI BIOS, and may not be the same as the ordinal number of the slot.

FRAME

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

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

Member	Description
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 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.

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.

Member	Description
hbuf	Handle to the image buffer.
dwFrames	Number of frames in the image buffer.
frame[SYS_FRAMES]	Array of FRAME structures.

The **BUFFER** structure is initialized by **X11_AllocBuffer** function.

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.

Member	Description												
scale	Defines image scale. Can be one of the following: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>SCALE_ADVANCED</td><td>Image scale is defined by the settings in the MODE_ADVANCED structure.</td></tr><tr><td>SCALE8</td><td>Full size image.</td></tr><tr><td>SCALE6</td><td>3/4 size image.</td></tr><tr><td>SCALE4</td><td>1/2 size image.</td></tr><tr><td>SCALE2</td><td>1/4 size image.</td></tr></tbody></table>	Value	Description	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.
Value	Description												
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: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>COLOR_MONO</td><td>Monochrome image, 1 byte per pixel.</td></tr><tr><td>COLOR_RGB</td><td>Color image, 3 bytes per pixel.</td></tr></tbody></table>	Value	Description	COLOR_MONO	Monochrome image, 1 byte per pixel.	COLOR_RGB	Color image, 3 bytes per pixel.						
Value	Description												
COLOR_MONO	Monochrome image, 1 byte per pixel.												
COLOR_RGB	Color image, 3 bytes per pixel.												
store	Defines the way the image is stored in memory. Can be one of the following: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>STORE_DIB</td><td>Image is stored as a Windows DIB</td></tr></tbody></table>	Value	Description	STORE_DIB	Image is stored as a Windows DIB								
Value	Description												
STORE_DIB	Image is stored as a Windows DIB												

STORE_FLAT (reversed lines order).
Image is stored with normal lines order.

input

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

<u>Value</u>	<u>Description</u>
MUX_0	S-Video or Video 0 input.
MUX_1	Video 1 input.
MUX_2	Video 2 input.
MUX_3	Video 3 input.

Note: **MUX_3** setting is valid only for model 1101, in which case a camera can be connected to either S-Video or Video 0 input, not both. See hardware reference for details.

advanced

MODE_ADVANCED structure. Defines the advanced mode settings.

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 reserved1;
    DWORD reserved2;
    DWORD reserved3;
    DWORD reserved4;
} MODE_ADVANCED;
```

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

Member	Description																
interlace	Defines input image format. Can be one of the following: <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>IMG_INTERLACED</td> <td>Interlaced input image.</td> </tr> <tr> <td>IMG_NONINTERLACED</td> <td>Noninterlaced input image.</td> </tr> </tbody> </table>	Value	Description	IMG_INTERLACED	Interlaced input image.	IMG_NONINTERLACED	Noninterlaced input image.										
Value	Description																
IMG_INTERLACED	Interlaced input image.																
IMG_NONINTERLACED	Noninterlaced input image.																
xTotal	Total number of output horizontal pixels (including horizontal blanking). Should 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. Should 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 horizontal sync, pixels. The following condition should 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. Should be between 1.0 and 8.0.																
yActive	Number of active output lines (<u>before vertical scaling is applied</u>). Should be between 60 and 525 (NTSC), or 625 (PAL). For example, to grab the whole NTSC image scaled down by the factor of 2, set yActive to 525, yFactor to 2.0. To grab the upper half of the NTSC image scaled down by the factor of 2, set yActive to 262, yFactor to 2.0.																
yDelay	The vertical offset of the start of the active area relative to vertical sync, lines (<u>before vertical scaling is applied</u>). The following condition should always be met: yDelay + yActive <= (total lines, 525 or 625) .																
FORMAT	Input signal format. Can be one of the following: <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>FORMAT_NTSC</td> <td>NTSC input signal.</td> </tr> <tr> <td>FORMAT_NTSCJ</td> <td>NTSC (Japan) input signal.</td> </tr> <tr> <td>FORMAT_PAL</td> <td>PAL input signal.</td> </tr> <tr> <td>FORMAT_PALM</td> <td>PAL(M) input signal.</td> </tr> <tr> <td>FORMAT_PALN</td> <td>PAL(N) input signal.</td> </tr> <tr> <td>FORMAT_PALNC</td> <td>PAL(N-combination) input signal.</td> </tr> <tr> <td>FORMAT_SECAM</td> <td>SECAM input signal.</td> </tr> </tbody> </table>	Value	Description	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.
Value	Description																
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 "checkboard" pattern in the output image, in case a composite input is used. Can be one of the following:

<u>Value</u>	<u>Description</u>
LNOTCH_OFF	Filter disabled.
LNOTCH_ON	Filter enabled.

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:

<u>Value</u>	<u>Description</u>
LDEC_OFF	Filter disabled.
LDEC_ON	Filter enabled.

DEC_RAT

A 6-bit value corresponding to the number of fields or 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:

<u>Value</u>	<u>Description</u>
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:

<u>Value</u>	<u>Description</u>
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:

<u>Value</u>	<u>Description</u>
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:

<u>Value</u>	<u>Description</u>
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:	
	Value	Description
	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:	
	Value	Description
	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:	
	Value	Description
	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:	
	Value	Description
	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:	
	Value	Description
	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 ORed, to disable both A/D's.
CRUSH	Controls the AGC mode. Can be one of the following:	
	Value	Description
	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:	
	Value	Description
	If YCOMB is set to YCOMB_ON :	
	VFILT_0	2-tap filter.
	VFILT_1	3-tap filter. Only available if scaling to less than 385 horizontal active pixels.
	VFILT_2	4-tap filter. Only available if scaling to less than 193 horizontal active pixels.
	VFILT_3	5-tap filter. Only available if scaling to less than 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 than 385 horizontal active pixels.

VFILT_2	3-tap filter and 2-tap interpolation. Only available if scaling to less than 193 horizontal active pixels.										
VFILT_3	4-tap filter and 2-tap interpolation. Only available if scaling to less than 193 horizontal active pixels.										
COLOR_BARS	Controls a test color bar pattern. Can be one of the following: <table border="0"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>COLORBARS_OFF</td> <td>Color bars off.</td> </tr> <tr> <td>COLORBARS_ON</td> <td>Color bars on.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	COLORBARS_OFF	Color bars off.	COLORBARS_ON	Color bars on.				
<u>Value</u>	<u>Description</u>										
COLORBARS_OFF	Color bars off.										
COLORBARS_ON	Color bars on.										
GAMMA	Controls gamma correction removal. Can be one of the following: <table border="0"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>GAMMA_REMOVE_ON</td> <td>Gamma correction removal on.</td> </tr> <tr> <td>GAMMA_REMOVE_OFF</td> <td>Gamma correction removal off.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	GAMMA_REMOVE_ON	Gamma correction removal on.	GAMMA_REMOVE_OFF	Gamma correction removal off.				
<u>Value</u>	<u>Description</u>										
GAMMA_REMOVE_ON	Gamma correction removal on.										
GAMMA_REMOVE_OFF	Gamma correction removal off.										
PKTP	FIFO trigger point. Can be one of the following: <table border="0"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>PKTP4</td> <td>4 DWORDs.</td> </tr> <tr> <td>PKTP8</td> <td>8 DWORDs.</td> </tr> <tr> <td>PKTP16</td> <td>16 DWORDs.</td> </tr> <tr> <td>PKTP32</td> <td>32 DWORDs.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	PKTP4	4 DWORDs.	PKTP8	8 DWORDs.	PKTP16	16 DWORDs.	PKTP32	32 DWORDs.
<u>Value</u>	<u>Description</u>										
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). 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 the corresponding sample application for the details. Note: Available only in SX11 Enhanced SDK. <table border="0"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>BIMODAL_OFF</td> <td>normal mode</td> </tr> <tr> <td>BIMODAL_ON</td> <td>bimodal mode.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	BIMODAL_OFF	normal mode	BIMODAL_ON	bimodal mode.				
<u>Value</u>	<u>Description</u>										
BIMODAL_OFF	normal mode										
BIMODAL_ON	bimodal mode.										
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 details.										
buffertype	Type of buffer to allocate. Can be one of the following: <table border="0"> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>BUF_MEM</td> <td>regular memory buffer</td> </tr> <tr> <td>BUF_EXT</td> <td>external buffer. Note: Available only in SX11 Enhanced SDK.</td> </tr> <tr> <td>BUF_VIDEO</td> <td>video memory buffer. Note: Available only in SX11 Enhanced SDK.</td> </tr> </tbody> </table> See X11_AllocBuffer function description for details.	<u>Value</u>	<u>Description</u>	BUF_MEM	regular memory buffer	BUF_EXT	external buffer. Note: Available only in SX11 Enhanced SDK.	BUF_VIDEO	video memory buffer. Note: Available only in SX11 Enhanced SDK.		
<u>Value</u>	<u>Description</u>										
BUF_MEM	regular memory buffer										
BUF_EXT	external buffer. Note: Available only in SX11 Enhanced SDK.										
BUF_VIDEO	video memory buffer. Note: Available only in SX11 Enhanced SDK.										
reserved1(2,3,4)	Reserved. Do not modify.										

INT_DATA

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

The **INT_DATA** structure contains information required for interrupts support.

Member	Description																		
hfg	Frame grabber handle. Selects the frame grabber.																		
mask	Interrupt mask. Setting mask to 0 disables interrupts. Setting mask to the following values (which can be ORed) allows specific interrupts: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>STATUS_READY</td><td>Frame acquisition complete.</td></tr><tr><td>STATUS_VIDEO</td><td>Video status changed at the input (e.g. present to absent).</td></tr><tr><td>STATUS_HLOCK</td><td>Horizontal lock condition changed at the input.</td></tr><tr><td>STATUS_OFLOW</td><td>Overflow detected.</td></tr><tr><td>STATUS_HSYNC</td><td>Start of new line.</td></tr><tr><td>STATUS_VSYNC</td><td>Start of new field.</td></tr><tr><td>STATUS_FMT</td><td>Video format change detected (e.g. NTSC to PAL).</td></tr><tr><td>STATUS_ERROR</td><td>Transfer error occurred. This is a combination of bits.</td></tr></tbody></table>	Value	Description	STATUS_READY	Frame acquisition complete.	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.
Value	Description																		
STATUS_READY	Frame acquisition complete.																		
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.																		
status	Frame grabber status. The meaning of the individual bits corresponds to that of the interrupt mask. The value of status is set when the interrupt occurs.																		
func	Pointer to the user interrupt handling function. The function should return a DWORD, and take no arguments.																		
priority	An integer specifying the interrupt handling thread priority level. Takes the following values: 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_InterruptOn was called.																		
lost	The number of interrupts not processed.																		

Note

The bits of the interrupt mask correspond to those of the status word, so the same constants can be used for selecting individual status conditions.

Functions

X11_InitSystem

`__declspec(dllimport) ECODE __stdcall X11_InitSystem (pPCldata)`

`PCI * pPCldata;` `/* address of the PCI structure */`

The **X11_InitSystem** function performs the system initialization. It identifies the supported boards present in the system, and initializes internal data structures.

Parameter	Description
<code>pPCldata</code>	Points to the variable of PCI type.

Returns

The function returns 0 in case of successful initialization, or an error code. It also modifies the **PCI** type structure: **boards** contains the number of supported boards identified and initialized, **PCIslot** array contains the slot numbers of identified boards.

In case the system was already initialized by another process, a warning code (**WNG_INITIALIZED**) is returned, and the boards are not reset. All the data structures still are modified after the call to reflect the actual system.

The frame grabber handles for each board can be obtained with the help of **X11_GetHFG** function.

Example

```
PCI pci;
ECODE ecode;
int i;

ecode = X11_InitSystem (&pci);
if (ecode && (ecode < WNG_INITIALIZED)) {
    return ecode;
} else {
    printf ("Boards %d\n", pci.boards);
    for (i = 0; i < pci.boards; i++) {
        printf ("Slot %04X\n", pci.PCIslot[i]);
    }
    if (ecode) {
        printf ("Initialized by other process\n");
    }
}
```

X11_CloseSystem

`__declspec(dllimport) void __stdcall X11_InitSystem (void)`

The **X11_CloseSystem** function frees any allocated system resources. It has to be called before terminating the application.

Example

```
PCI pci;
ECODE ecode;

if (!(ecode = X11_InitSystem (&pci))) {
    /*
     * application code here*
     */
}
```

```

        X11_CloseSystem;
        return 0;
    } else {
        return ecode;
    }
}

```

X11_AllocBuffer

__declspec(dllimport) ECODE __stdcall X11_AllocBuffer (pMode, pBuffer, dwParameter)

MODE * pMode; /* address of the **MODE** structure */
BUFFER * pBuffer; /* address of the **BUFFER** structure */
DWORD dwParameter; /* context dependent parameter */

The **X11_AllocBuffer** function allocates an image buffer.

Parameter	Description								
<i>pMode</i>	Points to the variable of MODE type. 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 settings affecting scaling, color format, or storage type, the image buffer has to be re-allocated (see MODE , MODE_ADVANCED , and X11_FreeBuffer).								
<i>pBuffer</i>	Points to the variable of BUFFER type. The members of BUFFER structure are set by X11_AllocBuffer , if the call is successful.								
<i>dwParameter</i>	Parameter which depends on MODE.advanced.buffertype setting. <table border="1"> <thead> <tr> <th>buffertype</th> <th>dwParameter</th> </tr> </thead> <tbody> <tr> <td>BUF_MEM</td> <td>Number of frames in the image buffer. Must be between 1 and SYS_FRAMES.</td> </tr> <tr> <td>BUF_EXT</td> <td>A pointer to the external buffer.</td> </tr> <tr> <td>BUF_VIDEO</td> <td>A handle of the image display window.</td> </tr> </tbody> </table>	buffertype	dwParameter	BUF_MEM	Number of frames in the image buffer. Must be between 1 and SYS_FRAMES .	BUF_EXT	A pointer to the external buffer.	BUF_VIDEO	A handle of the image display window.
buffertype	dwParameter								
BUF_MEM	Number of frames in the image buffer. Must be between 1 and SYS_FRAMES .								
BUF_EXT	A pointer to the external buffer.								
BUF_VIDEO	A handle of the image display window.								

Returns

The function returns 0 in case of success, or an error code. It also modifies the **BUFFER** type structure: **hbuf** contains the allocated image buffer handle, **dwFrames** - the number of frames in the image buffer, and **frame** array - pointers to the the individual frames' data and associated bitmap information.

Notes

The **X11_AllocBuffer** could be used to allocate buffers of three types: regular buffer, external buffer, and video memory buffer. The type of the allocated buffer is controlled by **MODE.advanced.buffertype** member.

Regular buffer is allocated in the PC memory when **X11_AllocBuffer** is called. This option is the most common.

External buffer must be allocated by an application prior to the call to **X11_AllocBuffer**. In this case a pointer to this buffer is passed to **X11_AllocBuffer**, which insures image capture into this pre-allocated buffer. The size of the buffer should match the image format settings. This option is useful when interfacing to a different application, or library, e.g. an image processing package. Creating an image buffer externally, and then "mapping" an acquisition buffer onto it provides an easy interface to the third party software, eliminating the need for image copying. See the sample application that illustrates interfacing to Matrox Imaging Library (MIL) for the details. **Note:** Available only in SX11 Enhanced SDK.

Video memory buffer is allocated in the graphics display (video card) memory. Capturing directly into video memory provides the fastest capture-and-display operation. In this mode the **SX11 SDK** makes

use of DirectDraw, allowing overlays of text and/or graphics over the captured image in real time. The application creates an image display window, and passes its handle to **X11_AllocBuffer**. All information that has to show on top of the image is output to the window client area the way it is usually done for Windows applications. A "key color" has to be defined prior to the call to **X11_AllocBuffer**, that is the color that will be replaced by the captured image. The key color is defined by **MODE.advanced.keycolor**. For example, if the key color is white (RGB(255,255,255)), the captured image will show through all white pixels of the window's client area, while all pixels that are not white will show on top of the captured image. The captured image format in this mode has to be YCrCb (mode.color=**COLOR_YCRCB**). The video card should be capable of DirectDraw and YUV overlay surfaces support. Multiple frame buffers are not supported in this mode. Only one overlay window can be created at a time. See the corresponding sample application for the details. **Note:** Available only in SX11 Enhanced SDK.

Example

```
//mode.advanced.buffertype = BUF_MEM
//dwFrames = number of frames to allocate
ecode = X11_AllocBuffer (&mode, &buffer, dwFrames);

//mode.advanced.buffertype = BUF_EXT
//p_extbuf = pointer to the external buffer
ecode = X11_AllocBuffer (&mode, &buffer, (DWORD) p_extbuf);

//mode.advanced.buffertype = BUF_VIDEO
//mode.advanced.keycolor = RGB (255, 255, 255) (for example)
//hwnd = image display window handle
ecode = X11_AllocBuffer (&mode, &buffer, (DWORD) hwnd);
```

X11_FreeBuffer

__declspec(dllexport) void __stdcall X11_FreeBuffer (hbuf)

HBUF hbuf, /* image buffer handle */

The **X11_FreeBuffer** function releases the resources associated with an image buffer.

Parameter	Description
<i>hbuf</i>	Handle of the image buffer to free.

Notes

X11_FreeBuffer has to be called in case the image buffer is reallocated (e.g. if the image format is changed). It is not necessary to call **X11_FreeBuffer** when the application is terminated, because **X11_CloseSystem** releases the buffer resources.

Example

```
PCI pci;
ECODE ecode;
MODE mode = {DEF_MODE_NTSC};
BUFFER buffer;

if (!(ecode = X11_InitSystem (&pci))) {
    if (!(ecode = X11_AllocBuffer (&mode, &buffer, 1))) {
        /* application code here */
        /* now we change the scaling */
        mode.scale = SCALE2;
        X11_FreeBuffer (buffer.hbuf);
        /* re-allocate the buffer */
        if (!(ecode = X11_AllocBuffer (&mode, &buffer, 1))) {
            /* application code here */
        } else {
```

```

        return ecode;
    }
    } else {
        return ecode;
    }
} else {
    return ecode;
}
}

```

X11_Acquire

__declspec(dllexport) ECODE __stdcall X11_Acquire (hfg, hbuf, timeout, pStatus)

HFG *hfg*; /* frame grabber handle */
HBUF *hbuf*; /* image buffer handle */
float *timeout*; /* acquisition timeout, sec */
DWORD * *pStatus*; /* address of the variable receiving status value */

The **X11_Acquire** function grabs the number of frames corresponding to that of the image buffer.

Parameter	Description
<i>hfg</i> <i>hbuf</i>	Frame grabber handle. Selects the frame grabber. Image buffer handle. Selects the image buffer. All frames of the image buffer are filled with data following one call to X11_Acquire . See BUFFER and X11_AllocBuffer .
<i>timeout</i>	Acquisition timeout in seconds. The function returns if the acquisition is not completed after <i>timeout</i> seconds.
<i>pStatus</i>	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.

Returns

The function returns after the acquisition is complete, or timeout expires. The return value is 0 in case of success, or an error code. 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.

Example

```

PCI pci;
ECODE ecode;
MODE mode = {DEF_MODE_NTSC};
BUFFER buffer;
DWORD frames = THAT_MANY;
HFG hfg;
float timeout = .5;
DWORD status;

if (!(ecode = X11_InitSystem (&pci))) {
    /* assume we need the 1st frame grabber */
    if (!(ecode = X11_GetHFG (&hfg, pci.PCIslot[0]))) {
        if (!(ecode = X11_AllocBuffer (&mode, &buffer, frames))) {
            if (!(ecode = X11_Acquire (hfg, buffer.hbuf,
                timeout, &status))) {
                /* application code here */
            } else {
                return ecode;
            }
        }
    }
}

```

```

        } else {
            return ecode;
        }
    } else {
        return ecode;
    }
} else {
    return ecode;
}
}

```

X11_StartAcquire

__declspec(dllimport) ECODE __stdcall X11_SrartAcquire (hfg, hbuf, acqmode)

HFG *hfg*; /* frame grabber handle */
HBUF *hbuf*; /* image buffer handle */
DWORD *acqmode*; /* acquisition mode */

The **X11_StartAcquire** function starts the acquisition of the number of frames corresponding to that of the image buffer, and returns immediately.

Parameter	Description						
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.						
<i>hbuf</i>	Image buffer handle. Selects the image buffer. All frames of the image buffer are filled with data following one call to X11_StartAcquire . See BUFFER and X11_AllocBuffer .						
<i>acqmode</i>	Acquisition mode switch. Can be one of the following: <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>AMODE_SINGLE</td> <td>Image buffer is filled once.</td> </tr> <tr> <td>AMODE_CONT</td> <td>Image buffer is filled continuously, until the acquisition is stopped.</td> </tr> </tbody> </table>	Value	Description	AMODE_SINGLE	Image buffer is filled once.	AMODE_CONT	Image buffer is filled continuously, until the acquisition is stopped.
Value	Description						
AMODE_SINGLE	Image buffer is filled once.						
AMODE_CONT	Image buffer is filled continuously, until the acquisition is stopped.						

Returns

The function returns 0 in case of success, or an error code. It returns immediately after the acquisition is started. The application determines if 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. If **AMODE_SINGLE** is selected, 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 at the start of the first frame acquisition. There is no interrupt associated with the **STATUS_READY_ALL** bit.

Example

```

PCI pci;
ECODE ecode;
MODE mode = {DEF_MODE_NTSC};
BUFFER buffer;
DWORD frames = THAT_MANY;
HFG hfg;
DWORD status;

if (!(ecode = X11_InitSystem (&pci))) {
    /* assume we need the 1st frame grabber */
    if (!(ecode = X11_GetHFG (&hfg, pci.PCIslot[0]))) {
        if (!(ecode = X11_AllocBuffer
            (&mode, &buffer, frames))) {
            if (!(ecode = X11_StartAcquire

```

```

        (hfg, buffer.hbuf, timeout, &status)) {
    /* wait until acquisition complete */
    while (!(ecode = X11_GetStatus (hfg, &status)) &&
           !(status & STATUS_READY_ALL)) {
    }
    if (!ecode) {
        /* application code here */
    } else {
        return ecode;
    }
    } else {
        return ecode;
    }
    } else {
        return ecode;
    }
    } else {
        return ecode;
    }
    } else {
        return ecode;
    }
}

```

X11_StopAcquire

__declspec(dllexport) ECODE __stdcall X11_StopAcquire (hfg)

HFG *hfg*; /* frame grabber handle */

Parameter	Description
-----------	-------------

<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
------------	--

The **X11_StopAcquire** function stops the image acquisition. It has to be used only if acquisition was started by calling **X11_StartAcquire**.

Returns

The function returns 0 in case of success, or an error code.

X11_GetStatus

__declspec(dllexport) ECODE __stdcall X11_GetStatus (hfg, pStatus)

HFG *hfg*; /* frame grabber handle */
DWORD **pStatus*; /* address of the variable receiving status value */

The **X11_GetStatus** function retrieves the value of the frame grabber status word.

Parameter	Description
-----------	-------------

<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>pStatus</i>	Address of the variable receiving the status value.

Returns

The function returns 0 in case of success, or an error code. The individual bits of the status word have the following meanings:

Value	Description
STATUS_READY	Frame acquisition complete. Has to be reset by the application. See X11_ResetStatus .
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.

X11_ResetStatus

`__declspec(dllexport) ECODE __stdcall X11_ResetStatus (hfg, mask)`

HFG *hfg*; /* frame grabber handle */
DWORD *mask*; /* reset mask */

The **X11_ResetStatus** function resets individual bits of the frame grabber status register.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>mask</i>	A value of 1 resets the corresponding bit of the status register.

Returns

The function returns 0 in case of success, or an error code. For the meanings of the status word individual bits see **X11_GetStatus**.

X11_GetDStatus

`__declspec(dllexport) ECODE __stdcall X11_GetDStatus (hfg, pStatus)`

HFG *hfg*; /* frame grabber handle */
DWORD **pStatus*; /* address of the variable receiving status value */

The **X11_GetDStatus** function retrieves additional status information.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>pStatus</i>	Address of the variable receiving the status value.

Returns

The function returns 0 in case of success, or an error code. The individual bits of the status word have the following meanings:

Value	Description
DSTATUS_PRES	Video present. Reset to 0 when input sync is not detected in 31 consecutive line periods.
DSTATUS_HLOCK	Device in horizontal lock.
DSTATUS_FIELD	Reflects whether an odd or even field is being captured (0 - odd field).
DSTATUS_NUML	Number of lines found in input video signal (0 - 525, 1 - 625).

DSTATUS_CSEL Identifies which crystal is selected.
DSTATUS_LOF Luma ADC overflow. Set to 0 at the reset, set to 1 if an overflow occurs. Has to be written to to be reset.
DSTATUS_COF Chroma ADC overflow. Set to 0 at the reset, set to 1 if an overflow occurs. Has to be written to to be reset.

Note

Use **X11_SetDStatus** to reset status bits.

X11_SetDStatus

`__declspec(dllexport) ECODE __stdcall X11_SetDStatus (hfg, status, mask)`

HFG *hfg*; /* frame grabber handle */
DWORD *status*; /* a value to write */
DWORD *mask*; /* write mask */

The **X11_SetDStatus** function modifies individual bits of the frame grabber **DSTATUS** register.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>status</i>	Status value to be written.
<i>mask</i>	Write mask. Bits set to 1 allow modification of the corresponding bit of DSTATUS register.

Returns

The function returns 0 in case of success, or an error code. For the meanings of the status word individual bits see **X11_GetDStatus**.

X11_SetMode

`__declspec(dllexport) ECODE __stdcall X11_SetMode (hfg, pMode)`

HFG *hfg*; /* frame grabber handle */
MODE * *pMode*; /* address of the **MODE** type variable */

The **X11_SetMode** function sets the required frame grabber mode.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>pMode</i>	Points to the MODE type variable containing mode settings.

Returns

The function returns 0 in case of success, or an error code.

Note

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** are set accordingly.

X11_InterruptOn

`__declspec(dllexport) ECODE __stdcall X11_InterruptOn (pIntData)`

`INT_DATA * pIntData; /* address of the variable of INT_DATA type */`

The **X11_InterruptOn** function enables the interrupts for the particular frame grabber.

Parameter	Description
<i>pIntData</i>	Pointer to the <u>global</u> variable containing the interrupt settings.

Returns

The function returns 0 in case of success, or an error code.

Notes

The **SX11.dll** handles the interrupts by creating a parallel thread, which wakes up when an allowed interrupt occurs. The system part of the interrupt handling procedure copies the necessary data to the global variable of **INT_DATA** type pointed to by *pIntData*, and calls the user function pointed to by the **func** member of **INT_DATA**. The **INT_DATA** structure contains the following members:

Member	Description																		
hfg	Frame grabber handle. Selects the frame grabber.																		
mask	Interrupt mask. Setting mask to 0 disables interrupts. Setting mask to the following values (which can be ORed) allows specific interrupts: <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>STATUS_READY</td> <td>Frame acquisition complete.</td> </tr> <tr> <td>STATUS_VIDEO</td> <td>Video status changed at the input (e.g. present to absent).</td> </tr> <tr> <td>STATUS_HLOCK</td> <td>Horizontal lock condition changed at the input.</td> </tr> <tr> <td>STATUS_OFLOW</td> <td>Overflow detected.</td> </tr> <tr> <td>STATUS_HSYNC</td> <td>Start of new line.</td> </tr> <tr> <td>STATUS_VSYNC</td> <td>Start of new field.</td> </tr> <tr> <td>STATUS_FMT</td> <td>Video format change detected (e.g. NTSC to PAL).</td> </tr> <tr> <td>STATUS_ERROR</td> <td>Transfer error occurred. This is a combination of bits.</td> </tr> </tbody> </table>	Value	Description	STATUS_READY	Frame acquisition complete.	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.
Value	Description																		
STATUS_READY	Frame acquisition complete.																		
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.																		
status	Frame grabber status. The meaning of the individual bits corresponds to that of the interrupt mask. The value of status is set when the interrupt occurs.																		
func	Pointer to the user interrupt handling function. The function should return a DWORD, and take no arguments.																		
priority	An integer specifying the interrupt handling thread priority level. Takes the following values: 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_InterruptOn was called.																		
lost	The number of interrupts not processed. The interrupts that occur while																		

the user function is executing.

The data can be exchanged between the main application and user function via a global variable, like shown in the example below. By default all interrupts are masked, that is turned off, at the reset. To unmask (allow) the interrupt, **X11_InterruptUnmask** function has to be called. Also, this function has to be called after the interrupt occurs, because the system part of the interrupt handling procedure masks the interrupts.

Example

```
/* global section */
/* define USER type to pass data to/from the user interrupt
   handling function. Assume components' types are defined */
struct USER {
    SOMETYPE user1;
    ANOTHERTYPE user2;
    YETANOTHERTYPE user3;
};
struct USER user;
INT_DATA intdata;
/* end of global section */

ECODE ecode;
HFG hfg;
BUFFER buffer;
BOOL enough;

/* Initialize the system, allocate buffer(s), get handles here */

/* Set up INT_DATA */
intdata.hfg = hfg;
intdata.mask = STATUS_READY;
intdata.func = UserFunc;          //see below;
intdata.priority = THREAD_PRIORITY_NORMAL;

/* Set up the necessary user data */
user.user2 = NOT_READY;          //some user flag;

/* Enable interrupt */
if (!(ecode = X11_InterruptOn (&intdata))) {
    if (!(ecode = X11_InterruptUnmask (hfg, STATUS_READY))) {
        /* start acquisition */
        X11_StartAcquire (hfg, buffer.hbuf, AMODE_SINGLE);
        /* do whatever is necessary here,
           for example: */
        while (!enough) {
            if (user.user2 == READY) {
                user.user2 = NOT_READY;
                /* start next acquisition */
                X11_InterruptUnmask (hfg, STATUS_READY);
                X11_StartAcquire (hfg, buffer.hbuf, AMODE_SINGLE);
                /* image is ready, do something with it*/
            }
        }
    } else {
        return ecode;
    }
} else {
    return ecode;
}

DWORD UserFunc (void)
```

```

{
    /* do what you need to here,
       pass the data via USER */
    user.user1 = 1;
    user.user2 = READY;           // etc.
    return 0;
}

```

X11_InterruptOff

__declspec(dllexport) ECODE __stdcall X11_InterruptOff (pIntData)

INT_DATA * pIntData; /* address of the variable of **INT_DATA** type */

The **X11_InterruptOff** function disables the interrupts for the particular frame grabber.

Parameter	Description
-----------	-------------

<i>pIntData</i>	Pointer to the <u>global</u> variable containing the interrupt settings.
-----------------	--

Returns

The function returns 0 in case of success, or an error code.

Notes

The argument of **X11_InterruptOff** function should be the same as used in the call to **X11_InterruptOn**.

X11_InterruptMask

__declspec(dllexport) ECODE __stdcall X11_InterruptMask (pIntData)

INT_DATA * pIntData; /* address of the variable of **INT_DATA** type */

The **X11_InterruptMask** function masks the interrupts for the particular frame grabber.

Parameter	Description
-----------	-------------

<i>pIntData</i>	Pointer to the <u>global</u> variable containing the interrupt settings.
-----------------	--

Returns

The function returns 0 in case of success, or an error code.

Notes

X11_InterruptMask disables selected interrupt(s) on a hardware level. The interrupt handling procedures remain active. The argument of **X11_InterruptMask** function should be the same as used in the call to **X11_InterruptOn**.

X11_InterruptUnmask

__declspec(dllexport) ECODE __stdcall X11_InterruptUnmask (pIntData)

INT_DATA * pIntData; /* address of the variable of **INT_DATA** type */

The **X11_InterruptUnmask** function unmask the interrupts for the particular frame grabber.

Parameter	Description
<i>pIntData</i>	Pointer to the <u>global</u> variable containing the interrupt settings.

Returns

The function returns 0 in case of success, or an error code.

Notes

X11_InterruptUnmask enables selected interrupt(s) on a hardware level. The argument of the function should be the same as used in the call to **X11_InterruptOn**.

X11_GetHFG

__declspec(dllexport) ECODE __stdcall X11_GetHFG (phfg, slot)

HFG * *phfg*; /* address of the frame grabber handle */
DWORD *slot*; /* PCI slot number */

The **X11_GetHFG** retrieves the value of the frame grabber handle given the PCI slot number of the board.

Parameter	Description
<i>phfg</i>	Points to the variable of HFG type.
<i>slot</i>	PCI slot number.

Returns

The function returns 0 in case of success, or an error code. It sets the variable pointed to by *phfg* to the value of the frame grabber handle.

Example

```

PCI pci;
ECODE ecode;
HFG hfg[SYS_GRABBERS];
int i;

if (!(ecode = X11_InitSystem (&pci))) {
    for (i = 0; i < pci.boards; i++) {
        if ((ecode = X11_GetHFG (&hfg[i], pci.PCIslot[i]))) {
            return ecode;
        }
    }
    /* application code here */
} else {
    return ecode;
}

```

X11_GetRC

__declspec(dllexport) ECODE __stdcall X11_GetRC (hbuf, frame, pmode, rowcol, rnum, pArray)

HBUF *hbuf*; /* image buffer handle */
DWORD *frame*; /* frame number */
MODE * *pmode*; /* address of **MODE** structure */
DWORD *rowcol*; /* access mode flag */
DWORD *rnum*; /* row or column number */
void * *pArray*; /* address of receiving array */

The **X11_GetRC** retrieves the data corresponding to one row or column of an image buffer frame into an external buffer (array).

Parameter	Description						
<i>hbuf</i>	Image buffer handle.						
<i>frame</i>	Frame number (starting with 0).						
<i>pmode</i>	Points to the variable of MODE type.						
<i>rowcol</i>	Access mode flag. Can be one of the following:						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>RMODE_ROW</td> <td>Retrieves row data</td> </tr> <tr> <td>RMODE_COL</td> <td>Retrieves column data</td> </tr> </tbody> </table>	Value	Description	RMODE_ROW	Retrieves row data	RMODE_COL	Retrieves column data
Value	Description						
RMODE_ROW	Retrieves row data						
RMODE_COL	Retrieves column data						
<i>rcnum</i>	Row or column number (starting with 0).						
<i>pArray</i>	Address of the external buffer (array) to copy the row (column) data into.						

Returns

The function returns 0 in case of success, or an error code.

Notes

The function could be useful when writing applications with the tools that can not handle pointers directly (for example, Visual Basic). It allows to get access to the pixel data by retrieving rows or columns from an individual frame.

X11_WritePort

__declspec(dllexport) ECODE __stdcall X11_WritePort (hfg, data, mask)

HFG *hfg*; /* frame grabber handle */
DWORD *data*; /* data to write to the output port */
DWORD *mask*; /* write mask */

The **X11_WritePort** function writes to the 4-bit output port of the frame grabber.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>data</i>	Data to write to the output port. Only lower 4 bits are significant.
<i>mask</i>	A value of 1 allows modification of a corresponding bit. Facilitates modifications of individual bit(s) without affecting the other.

Returns

The function returns 0 in case of success, or an error code.

X11_ReadPort

__declspec(dllexport) ECODE __stdcall X11_ReadPort (hfg, pData)

HFG *hfg*; /* frame grabber handle */
DWORD * *pData*; /* address of the data variable */

The **X11_ReadPort** function reads from the 4-bit input port of the frame grabber.

Parameter	Description
<i>hfg</i>	Frame grabber handle. Selects the frame grabber.
<i>pData</i>	Points to the DWORD variable receiving the data into

the lower 4 bits.

Returns

The function returns 0 in case of success, or an error code.

X11_GetImageSize

`__declspec(dllexport) ECODE __stdcall X11_GetImageSize (pMode, pXsize, pYsize)`

MODE * pMode; /* address of the **MODE** type variable */
DWORD * pXsize; /* address of the variable receiving horizontal size */
DWORD * pYsize; /* address of the variable receiving vertical size */

The **X11_GetImageSize** function retrieves the dimensions of the image corresponding to the particular mode. It is convenient in cases 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.

Parameter	Description
<i>pMode</i>	Points to the MODE type variable containing mode settings.
<i>pXsize</i>	Points to the DWORD receiving the horizontal size of the image, in pixels.
<i>pYsize</i>	Points to the DWORD receiving the vertical size of the image, in pixels.

Returns

The function returns 0 in case of success, or an error code.

X11_GetApps

`__declspec(dllexport) DWORD __stdcall X11_GetApps (void)`

The **X11_GetApps** returns the number of applications currently connected to **sx11.dll**.

Note

The **sx11.dll** shares some of the internal data between applications. This is done to allow multiple applications access different boards without interfering with each other. (Of course, it is not recommended to access the same board from different applications). For example, suppose there are 2 frame grabbers in the system, and each of them has to be accessed from a separate application. The first application to start finds 2 frame grabbers and initializes them. The second application should not initialize the frame grabbers when it starts, because by that time the first one might have already set up a particular operation mode. By sharing the data **sx11.dll** allows the applications that are started later to skip initialization procedures. See **X11_InitSystem** for more details.