sharpVISION™ SDK

Reference Manual

Software Development Kit

SDK Release
2.07

Document Revision
October 2006

Products Information
http://www.idtvision.com

North America
1804 Miccosukee Commons, suite 208
TALLAHASSE FL 32308
USA
P: (+1) (850) 222-5939
F: (+1) (850) 222-4591
llourenco@idtpiv.com

Europe
via Pennella, 94
I-38057 - Pergine Valsugana (TN)
ITALY
P: (+39) 0461- 53 21 12
F: (+39) 0461- 53 21 04
pgallorosso@idtpiv.com

Table of Contents

1. Overview

The on-line documentation of the sharpVISION™ Software Development Kit and its components is divided into the following parts:

Using sharpVISION™ SDK

This section describes how to start using the sharpVISION™ SDK.

sharpVISION™ SDK Reference

This section contains a detailed description of the sharpVISION™ SDK functions.

sharpVISION™ ActiveX Control Reference

This section contains a detailed description of the sharpVISION™ ActiveX functions.

sharpVISION™ TWAIN™ Driver Reference

This section contains a detailed description of the sharpVISION™ TWAIN™ Driver.

sharpVISION™ LabVIEW™ Interface Reference

This section contains a detailed description of the sharpVISION™ LabVIEW™ VIs.

sharpVISION™ MATLAB™ Interface Reference

This section contains a detailed description of the sharpVISION™ MATLAB™ Drivers.

Appendix

This section provides additional information about data structures, parameters and functions return codes.

Important note: ActiveX and LabVIEW plug-ins are not supported in sharpVISION SDK MAC OS X version, but only in Windows version.

1.1. Directories structure

The default installation directory of the SDK is "**C:\Program Files\IDT\sharpVISION**". Under this directory a set of sub-directories is created:

BIN: it contains the files (drivers, INF, DLLs) that can be re-distributed with the camera and your application.

DOCS: it contains the SDK documentation and the camera manuals.

IMAGES: it's the default directory where the acquired images may be stored.

INCLUDE: it contains the SDK header files (H and BAS).

LABVIEW: it contains the LabVIEW™ example Virtual Instruments.

LIB: it contains the SDK lib file.

MATLAB: it contains the MATLAB™ drivers and examples.

SOURCE: it contains the Visual C++ SDK examples.

2. Using sharpVISION™ SDK

2.1. Programming Language

A C/C++ header file is included in the SDK (SharpAPI.h file in the Include sub-directory).

Most compiled languages can call functions; you will need to write your own header/import/unit equivalent based on the C header file.

In Windows version a Visual Basic module is included in the SDK (**SharpAPI.bas** file in the Include sub-directory). VB cannot use **SvQueueOneFrame** or **SvQueueCameraSettings** or related functions, because these functions have callbacks which occur on a different thread. If you want to use VB, you might need to write some C code depending on your application's requirements. The same issue with asynchronous callbacks, above, also applies to Java.

The Windows driver is a DLL (SharpDrv.dll) that resides in the system32 directory. It may be found also in the Bin sub-directory.

MS Visual C++™: A Visual C++ 6.0 stub COFF library is provided (**SharpDrv.lib** in the Lib sub-directory); if you are using Visual C++, link to SharpDrv.lib. The DLL uses Windows standard calling conventions (_stdcall).

Borland C++ Builder™: the SharpDrv.lib file is in COFF format. Borland C++ Builder requires the OMF format. To convert the library into to OMF format, use the IMPLIB Borland tool with the following syntax: “IMPLIB SharpDrv.lib SharpDrv.dll”.

Other compilers: the Most other compilers can create a stub library for DLLs. The DLL uses Windows standard calling conventions (_stdcall).

MAC OS Project Builder™: the driver is a Framework that resides in the **/Library/Frameworks** folder. If you use Apple Project Builder 2.1, add the sharpVISION.framework file to your project.

2.2. Load/Unload the Driver

The first call into the sharpVISION driver must be **SvLoadDriver**. Call **SvUnloadDriver** when you are finished.

2.3. Enumerate/Open a camera

To get the list of available cameras, call **SvEnumCameras**. Use the *cameraId* field of the camera list in your call to **SvOpenCamera**. Here is a simple example of opening the first available camera:

```
SV_ENUMITEM svList[10];
unsigned long nListLen = sizeof(szList)/sizeof(SV_ENUMITEM);
SvLoadDriver();
// nListLen is the length of your SV_ENUMITEM array
SvEnumCameras( &svList[0], &nListLen );
// nListLen is now the number of cameras available. It may be
// larger than your SV_ENUMITEM array length!
if (( nListLen > 0 ) && ( svList[0].isOpen == FALSE ))
{
      SV_HANDLE hCamera;
      // Open the first camera in the list.
      SvOpenCamera( svList[0].cameraId, &hCamera );
      // Do something...
      ...
      // Close the camera.
      SvCloseCamera( hCamera );
}
// Unload the driver
SvUnloadDriver();
```

The camera list contains a unique ID which identifies each particular camera. Many developers use the unique ID to recall previous settings, or associate a meaningful name string with a camera.

2.4. Configuring a camera

The camera state is represented by the opaque SV_SETTINGS structure. You can read the default state, read the state from the camera, or send the state to the camera. Parameters are read and written to a SV_SETTINGS structure with functions **SvGetParameter** and **SvSetParameter**. The functions **SvGetParameterMin** and **SvGetParameterMax** provide information on a parameter's range. Here is an example of setting the camera to 100 ms exposure time:

```
SV_SETTINGS svCfg;
svCfg.size = sizeof(SV_SETTINGS );    // Don't forget this!
// Read default settings from the camera.
SvReadDefaultSettings( hCamera, &svCfg );
// Change svCfg: set exposure to 10 ms.
SvSetParameter( &svCfg, SVP_EXPOSURE, 10000 );
// Send settings to the camera
SvSendCameraSettings( hCamera, &svCfg );
```

Not all parameters are supported by all cameras. When you query or set a parameter (or get the parameter maximum/minimum) and that parameter is not supported, the error code *SVC_NOTSUPPORTED* is returned.

Not all parameters are valid in a particular state. A good example of this is readout speed (see Readout Speed subsection below). When you call **SvSendCameraSettings** or **SvValidateSettings**, the parameters in your SV_SETTINGS structure are adjusted to valid values if necessary.

2.5. Synchronous Streaming Data Grab

To grab an image from steaming data, you need to allocate an image buffer of sufficient size, fill out a SV_FRAME structure, then call a grab function. Here is an example of a simple, synchronous frame grab:

```
SV_FRAME frame;
unsigned long nSizeInBytes;

// Image size depends on the current ROI & image format.
SvGetCameraInfo( hCamera, SVI_IMG_SIZE, &nSizeInBytes );

// Fill out fields in SV_FRAME structure.
frame.pBuffer = malloc(nSizeInBytes*sizeof(BYTE));
frame.bufferSize = nSizeInBytes;

// Do synchronous image grab with a 5 sec time out
SvGrabOneFrame( hCamera, &frame );

// Process the data
...

// free the buffer
free(frame.pBuffer);
```

Most applications will use the asynchronous grab function. The asynchronous function is required for capturing images at full speed; it also provides the ability to abort image capture. The asynchronous function is covered below.

2.6. Asynchronous Streaming Data Grab

An asynchronous grab function is available: **SvQueueOneFrame**. Many frames can be queued up at a time. The maximum number of frames that can be queued at a time is **100**. When an image arrives from the camera and a frame is complete, your application can receive a callback. The completed frame is removed from the queue, and if available the next frame takes its place.

Using **SvQueueOneFrame** has the following advantages over a synchronous grab:

- Your thread is not blocked.
- Queued frames are easily aborted (SvAbortGrab).
- Software trigger only works with queued frames.
- You can capture frames at full speed.

To capture frames at full speed, your application should be designed to keep at least two frames queued.

Image processing should be done on a separate thread, not during the frame-done callback. Process the image on your main thread, or pass the frame to a worker thread if necessary.

Most application programs will use the asynchronous **SvQueueOneFrame** rather than the synchronous **SvGrabOneFrame** to capture frames. Applications with a GUI will likely choose to control the camera and queue frames from their main GUI thread, and perform lengthy processing on a worker thread.

If the camera is in double exposure mode, the acquisition must be done using the asynchronous grab function.

2.7. Queue Camera Configuration

There is an asynchronous function used to change the camera state: **SvQueueCameraSettings**. The settings are placed on the same queue as your frames queued by **SvQueueOneFrame**. Actions are guaranteed to occur in the order they are queued. If you want to clear the queue, call **SvAbortGrab**. As with **SvQueueOneFrame**, you can receive a callback when the settings have been changed.

2.8. Binning

Binning is the process of combining adjacent pixels of the sensor during readout. The two primary benefits of binning are improved signal to noise ratio (SNR) and the ability to increase frame rate, albeit at the expense of reduced spatial resolution.

The maximum image size for each binning is not an even division of the 1x1 maximum image size. When you switch from a value of binning to another and update the configuration to the camera, your binning values will be adjusted to fit the new binning mode. When binning, ROI is specified in “super-pixels”.

A common mistake occurs when switching from higher binning, such as 4x4, to lower binning, such as 1x1. If the caller forgets to adjust the region, they will end up with the old 4x4 size. When switching binning modes, you might want to select the largest possible region as follows:

```
unsigned long nMaxWidth,nMaxHeight;

// Get the current maximum image size
SvGetCameraInfo( hCamera, SVI_CCD_WIDTH, &nMaxWidth );
SvGetCameraInfo( hCamera, SVI_CCD_HEIGHT, &nMaxHeight );

// Reset ROI to the new maximum values
SvSetParameter( &svCfg, SVP_ROIX, 0 );
SvSetParameter( &svCfg, SVP_ROIY, 0 );
SvSetParameter( &svCfg, SVP_ROIWIDTH, maxWidth );
SvSetParameter( &svCfg, SVP_ROIHEIGHT, maxHeight );
```

2.9. Read-out Speed and Pixel Format

The maximum readout speed depends on bit depth, although this restriction may not exist in future cameras. Only 8 bit formats can run at 20 MHz; 16 bit formats are limited to 10 MHz or lower. There is no advantage to using a lower readout speed. When you call **SvValidateSettings** or **SvSendCameraSettings**, the readout speed is lowered if your image format is 16 bits. A common mistake occurs when switching from a 16 bit format to 8 bit format. If you want to run at the highest speed, you need to explicitly switch the readout speed to 20 MHz. We suggest this approach: every time you change the format, change the readout speed to the maximum. The driver will automatically lower the speed if necessary.

```
// Restore the read out speed to 20 MHz
SvSetParameter( &svCfg, SVP_IMGFORMAT, nFormat );
SvSetParameter( &svCfg, SVP_READOUT, SV_ROS_20MHZ );
```

2.10. Region of Interest (ROI)

Different camera models have different behaviors:

1300-DE: ROI boundaries occur on multiplies of 16, although the maximum region for a particular binning mode is not restricted to this rule. Minimum ROI is 16x16.

1400-DE and **1500-EX**: they can do ROI on 4-pixel boundaries in width and 1-pixel in height.

On 1300-DE cameras, your ROI is adjusted in **SvSendCameraSettings** before the camera state is changed. The ROI is rounded out to include your original region, but with boundaries which occur on multiples of 16. If the ROI spills over the maximum-image boundary, the ROI will shrink to that boundary. Call **SvValidateSettings** to see how the driver will alter your ROI, and use the new ROI size suggested by the driver. In some cases, for example an automatic-exposure algorithm, you may be forced to crop the images you capture. If you are running in a binning mode, your region and image size is specified in super-pixels rather than CCD pixels.

2.11. Processes and Threads

sharpVISION is process safe. A camera may only be opened by one process at a time, and camera handles may not be passed across process boundaries. (To be clear: a process may open any number of cameras at a time.) When you call SvEnumCameras, you can check the *isOpen* field to determine if a particular camera has already been opened.

sharpVISION is not thread safe. If you are using more than one thread, you must serialize calls into sharpVISION.

3. sharpVISION™ SDK Reference

3.1. Initialization Functions

3.1.1. Overview: Initialization functions

Initialization functions allow the user to initialize the sharpVISION camera, enumerate the available cameras, open and close them.

SvGetVersion returns the DLL version numbers and the demo flag.

SvLoadDriver loads the driver and initializes it.

SvUnloadDriver unloads the driver.

SvEnumCameras enumerates the sharpVISION cameras connected to the computer.

SvOpenCamera opens a camera.

SvCloseCamera closes a camera previously open.

3.1.2. SvGetVersion

SV_ERROR SvGetVersion(unsigned short **pVerMajor*, unsigned short **pVerMinor* , int* *pIsDemo*)

Return values

SVC_SUCCESS if successful, otherwise

SVC_DRIVERFAULT if the version numbers could not be extracted from the driver.

Parameters

pVerMajor

Specifies the pointer to the variable that receives the major version number.

pVerMinor

Specifies the pointer to the variable that receives the minor version number.

pIsDemo

Specifies the pointer to the variable that receives the demo flag. If 1, the driver is demo, if 0 it isn't.

Remarks

This function must be called to retrieve the sharpVISION DLL version number and demo flag.

See also:

3.1.3. SvLoadDriver

SV_ERROR SvLoadDriver(void)

Return values

SVC_SUCCESS if successful, otherwise

SVC_DRIVERALREADYLOADED if the driver is already loaded.

SVC_OUTOFMEMORY if not enough memory could be allocated for the driver.

SVC_NOFIREWIREDRIVER, if the firewire driver could not be found.

SVC_FIREWIREFAULT, if an error occurred while calling the firewire driver.

Parameters

None

Remarks

The routine loads the sharpVISION driver DLL and initializes it. It must be called before any other routine, except SvGetVersion.

See also: SvUnloadDriver

3.1.4. SvUnloadDriver

void SvUnloadDriver(void)

Return values

None

Parameters

None

Remarks

This function must be called before terminating the application. This function frees any memory and resource allocated by the camera driver and unloads it.

See also: SvLoadDriver

3.1.5. SvEnumCameras

SV_ERROR SvEnumCameras(PSV_ENUMITEM *pItemList***, unsigned long ****pItemNr* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_FIREWIREFAULT, if an error occurred while calling the firewire driver.

Parameters

pItemList

Specifies the pointer to an array of SV_ENUMITEM structures.

pItemNr

Specifies the pointer to the variable that receives the number of detected cameras.

Remarks

The routine enumerates the cameras connected to the computer and fills the SV_ENUMITEM structures with information about the cameras detected. This routine must be called before SvOpenCamera to find out which cameras are available. The pItemNumber variable must specify the number of structures in the pItemList array and receives the number of detected cameras.

See also: SvOpenCamera

3.1.6. SvOpenCamera

SV_ERROR SvOpenCamera(unsigned long *cameraId*, SV_HANDLE* *pHandle*)

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDCAMERAID, if the camera ID is not valid.

SVC_UNKNOWNCAMERA, if the camera model is unknown to this version of the driver.

SVC_FIREWIREFAULT, if an error occurred while calling the firewire driver.

Parameters

cameraId

Specifies the ID of the camera to be opened.

pHandle

Specifies the pointer to the variable that receives the camera handle.

Remarks

The routine opens the camera whose ID is in the variable *cameraId*. The value can be retrieved calling the SvEnumCameras (see SV_ENUMITEM structure).

See also: SvCloseCamera

3.1.7. SvCloseCamera

SV_ERROR SvCloseCamera(SV_HANDLE *hCamera* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

Closes and open Camera.

See also: SvOpenCamera

3.2. Configuration Functions

3.2.1. Overview: Configuration functions

The configuration functions allow the user to control the status of the sharpVISION camera.

SvGetCameraInfo gets independent information from the camera, such as serial number, CCD type, etc.

SvReadDefaultSettings reads default settings in the SV_SETTINGS opaque structure.

SvReadCameraSettings reads camera settings in the SV_SETTINGS opaque structure.

SvSendCameraSettings sends settings to the camera.

SvValidateSettings validates and updates a camera state.

SvQueueCameraSettings queues camera settings.

SvSetParameter sets the camera parameters in the SV_SETTINGS opaque structure.

SvGetParameter gets the parameters from the SV_SETTINGS opaque structure.

SvGetParameterMin gets a parameter's minimum value.

SvGetParameterMax gets a parameter's maximum value.

SvEnableStreaming enables/disables data streaming.

3.2.2. SvGetCameraInfo

SV_ERROR SvGetCameraInfo(SV_HANDLE *hCamera***, SV_INFO** *infoKey***, unsigned long ****pValue* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_NOTSUPPORTED, if the infoKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera.

infoKey

Specifies which parameter the funcion returns.

pValue

Specifies the pointer to the variable that receives the parameter value

Remarks

This function returns camera specific informations, such as CCD type or version numbers, generally state-independent informations. See the Appendix for a list of all the available infoKey values.

See also: SvGetParameter

3.2.3. SvReadDefaultSettings

SV_ERROR SvReadDefaultSettings(SV_HANDLE *hCamera*, **PSV_SETTINGS** *pSettings* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid

Parameters

hCamera

Specifies the handle to an open camera.

pSettings

Specifies the pointer to the structure to be filled with the camera settings.

Remarks

This function reads the default settings of the specified camera and fills the SV_SETTINGS structure. The structure is opaque and can be accessed only through the SvGetParameter and SvSetParameter functions. To change a parameter on the camera, the entire structure must be sent to the driver, using the SvSendCameraSettings function. The default state is specific to each individual camera; some parameter defaults are factory calibrated.

See also: SvSendCameraSettings

3.2.4. SvReadCameraSettings

SV_ERROR SvReadCameraSettings (SV_HANDLE *hCamera*, **PSV_SETTINGS** *pSettings* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid

Parameters

hCamera

Specifies the handle to an open camera.

pSettings

Specifies the pointer to the structure to be filled with the camera settings.

Remarks

This function reads the current settings of the specified camera and fills the SV_SETTINGS structure. The structure is opaque and can be accessed only through the SvGetParameter and SvSetParameter functions. To change a parameter on the camera, the entire structure must be sent to the driver, using the SvSendCameraSettings function.

See also: SvSendCameraSettings

3.2.5. SvSendCameraSettings

SV_ERROR SvSendCameraSettings (SV_HANDLE *hCamera*, **PSV_SETTINGS** *pSettings* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

SVC_OUTOFMEMORY, if not enough memory is available.

Parameters

hCamera

Specifies the handle to an open camera.

pSettings

Specifies the pointer to the structure that contains the camera settings.

Remarks

The state contained in the SV_SETTINGS structure is validated, modified if necessary, then sent to the camera. The structure is opaque and can be accessed only through the SvGetParameter and SvSetParameter functions.

See also: SvReadDefaultSettings, SvReadCameraSettings

3.2.6. SvQueueCameraSettings

SV_ERROR SvQueueCameraSettings (SV_HANDLE *hCamera*, **PSV_SETTINGS** *pSettings*, **SV_AsyncCallback** *pfnCallback*, **unsigned long** *nFlags*, **void ****pUser*, **unsigned long** *nUserData*)

Return values

SVC_SUCCESS if successful, otherwise

Parameters

hCamera

Specifies the handle to an open camera.

pSettings

Specifies the pointer to the structure that contains the camera settings.

pfnCallback

Specifies the pointer to the callback routine. The routine is called by the driver when the settings are changed. See SC_AsynchCallback.

nFlags

Specifies the flags. See Appendix.

pUser

Specifies the pointer to user data. The pointer is passed to the callback rouitne when it's called.

nUserData

Specifies the value of user data.

Remarks

This function queues up a change to the camera state. This function returns immediately. When the camera state has changed, you will receive a callback if desired.

See also: SVSendCameraSettings

3.2.7. SvSetParameter

SV_ERROR SvSetParameter (PSV_SETTINGS *pSettings*, **SV_PARAM** *paramKey*, **unsigned long** *nValue*)

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_NOTSUPPORTED, if the paramKey is not supported.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

Parameters

pSettings

Specifies the pointer to the SV_SETTINGS structure the parameter is written to.

infoKey

Specifies which parameter the function sets.

nValue

Specifies the parameter's value.

Remarks

This function write a parameter to the opaque SV_SETTINGS structure. The parameter will not change on the camera until the entire structure is sent to the driver calling the SvSendCameraSettings or SvQueueCameraSettings functions.

See also: SvGetParameter, SvSendCameraSettings, SvQueueCameraSettings

3.2.8. SvGetParameter

SV_ERROR SvGetParameter(PSV_SETTINGS *pSettings*, **SV_PARAM** *paramKey*, **unsigned long ****pValue***)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_NOTSUPPORTED, if the paramKey is not supported.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

Parameters

pSettings

Specifies the pointer to the SV_SETTINGS structure the parameter is read from.

infoKey

Specifies which parameter the function returns.

pValue

Specifies the pointer to the parameter's value.

Remarks

This function reads a parameter from the opaque SV_SETTINGS structure.

See also: SvSetParameter

3.2.9. SvGetParameterMin

SV_ERROR SvGetParameterMin(PSV_SETTINGS *pSettings*, **SV_PARAM** *paramKey*, **unsigned long ****pValue***)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_NOTSUPPORTED, if the paramKey is not supported.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

Parameters

pSettings

Specifies the pointer to the SV_SETTINGS structure the parameter is read from.

infoKey

Specifies which parameter the function returns.

pValue

Specifies the pointer to the parameter's value.

Remarks

This function reads the minimum value of a parameter.

See also: SvGetParameterMax, SvGetParameter

3.2.10. SvGetParameterMax

SV_ERROR SvGetParameterMax(PSV_SETTINGS *pSettings*, **SV_PARAM** *paramKey*, **unsigned long ****pValue***)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_NOTSUPPORTED, if the paramKey is not supported.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

Parameters

pSettings

Specifies the pointer to the SV_SETTINGS structure the parameter is read from.

infoKey

Specifies which parameter the function returns.

pValue

Specifies the pointer to the parameter's value.

Remarks

This function reads the minimum value of a parameter.

See also: SvGetParameterMin, SvGetParameter

3.2.11. SvEnableStreaming

SV_ERROR SvEnableStreaming(SV_HANDLE *hCamera***, unsigned long** *bEnable* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BADSETTINGS, if the SV_SETTINGS structure is not valid.

SVC_OUTOFMEMORY, if not enough memory is available.

Parameters

hCamera

Specifies the handle to an open camera.

bEnable

Specifies whether the streaming is to be enabled or not. If the parameter is non zero, the streaming is enabled. If the parameter is zero, the streaming is disabled.

Remarks

This function enables or disables the camera streaming. When a camera is streaming, the frame rate is higher and the frames can be grabbed faster. When a camera is not streaming, the driver will automatically start and stop the firewire stream for each frame grab, so the rate is slower.

See also:

3.3. Grab Functions

3.3.1. Overview: Grab functions

Grab functions allow the user to capture streamed data from the digital camera.

The grab process may be performed in two ways:

- **Synchronous**: calling SvGrabFrame function.
- **Asynchronous**: calling SvQueuFrame function.

Both methods use the SV_FRAME structure to grab the data.

SvGrabOneFrame grabs one frame synchronously.

SvQueueOneFrame grabs one frame asynchronously.

SvAbortGrab aborts any pending asynchronous grab.

SvTrigger triggers a camera exposure.

3.3.2. SvGrabOneFrame

SV_ERROR SvGrabOneFrame(SV_HANDLE *hCamera*, **PSV_FRAME** *pFrame* **)**

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BUFFERTOOSMALL, if the frame buffer is too small for the image.

SVC_FIREWIREFAULT, if an internal error occurred while calling the firewire driver.

Parameters

hCamera

Specifies the handle to an open camera.

pFrame

Specifies the pointer to a SV_FRAME structure. The structure is used to acquire the frame.

Remarks

This function grabs synchronously a frame from the camera. It returns when the frame has been acquired.

See also: SvQueueOneFrame

3.3.3. SvQueueOneFrame

SV_ERROR SvQueueOneFrame(SV_HANDLE *hCamera*, **PSV_FRAME** *pFrame*, **SV_AsyncCallback** *pfnCallback*, **unsigned long** *nFlags*, **void *pUser**, **unsigned long** *nUserData*)

Return values

SVC_SUCCESS if successful, otherwise

SVC_INVALIDHANDLE, if the camera handle is not valid.

SVC_BUFFERTOOSMALL, if the frame buffer is too small for the image.

SVC_FIREWIREFAULT, if an internal error occurred while calling the firewire driver.

Parameters

hCamera

Specifies the handle to an open camera.

pFrame

Specifies the pointer to the frame structure.

pfnCallback

Specifies the pointer to the callback routine. The routine is called by the driver when the settings are changed. See SC_AsynchCallback.

nFlags

Specifies the flags. See Appendix.

pUser

Specifies the pointer to user data. The pointer is passed to the callback rouitne when it's called.

nUserData

Specifies the value of user data.

Remarks

This functions queues a frame buffer and returns immediately. It's used for asynchronous acquisitions. When the frame has been captured the *pfnCallback* routine is called. The frame structure and the associated data buffer must persist until the frame has been grabbed.

See also: SvGrabOneFrame

3.3.4. SvAbortGrab

SV_ERROR SvAbortGrab(SV_HANDLE *hCamera* **)**

Return values

SVC_SUCCESS if successful, otherwise

Parameters

hCamera

Specifies the handle to an open camera.

Remarks

This function stops all the pending grab operations and clears the queue. After the function has returned no more SvQueueOneFrame or SvQueueCameraSettings callbacks occur.

See also: SvQueueCameraSettings, SvQueueOneFrame

3.3.5. SvTrigger

SV_ERROR SvTrigger(SV_HANDLE *hCamera*)

Return values

SVC_SUCCESS if successful, otherwise

Parameters

hCamera

Specifies the handle to an open camera.

Remarks

This function triggers a camera exposure (software trigger). The camera trigger mode must be set to hardware or software trigger (see Appendix B). Firewire streaming must be on.

See also: SvQueueOneFrame

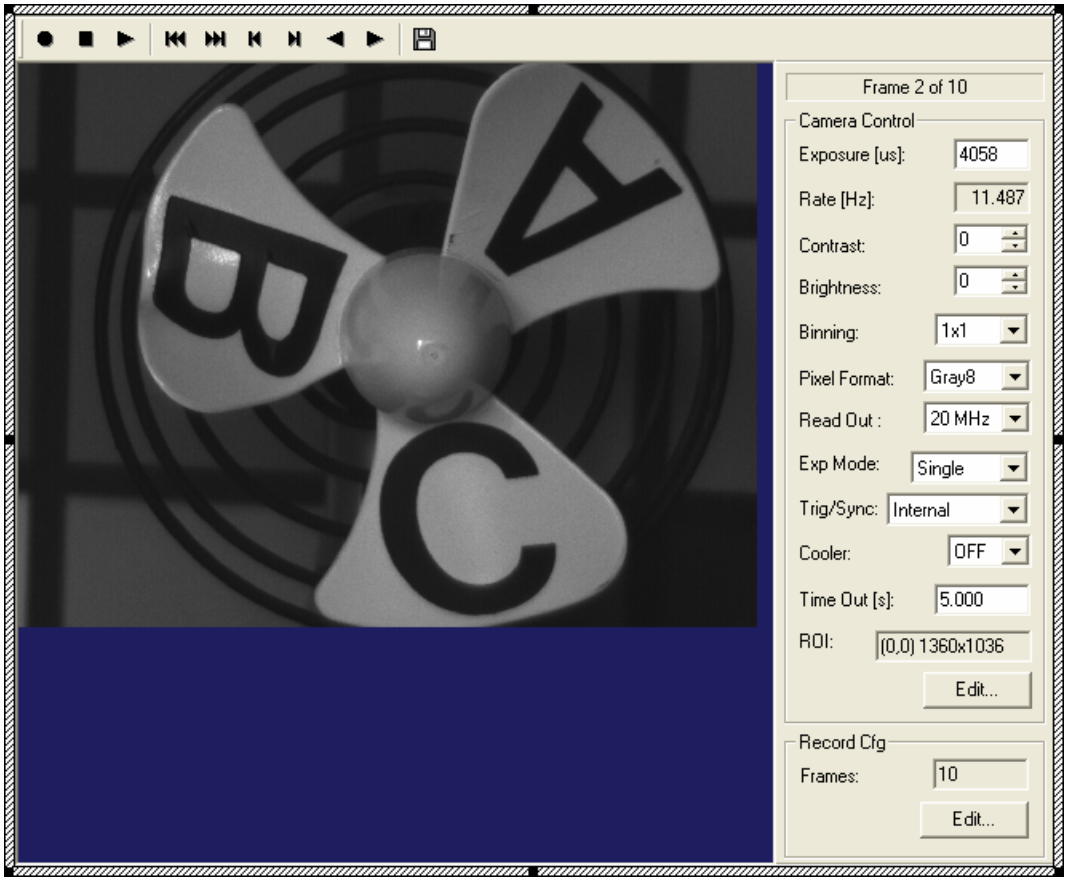
4. sharpXvision™ ActiveX Control Reference

4.1. Overview

ActiveX is a set of technologies that enable software components to interact with one another in a networked environment, regardless of the language in which the components were created. An ActiveX control is a user interface element created using ActiveX technology. ActiveX controls are small, fast, and powerful, and make it easy to integrate and reuse software components.

The sharpXvision ActiveX control includes all the capabilities of the sharpVISION camera in a simple control that can be inserted in any application. The ActiveX technology is supported only in Windows OS.

The figure below shows the sharpXvision ActiveX control user interface. The control may be inserted in any application and may be controlled using the software interface described below.



4.2. Camera Control Functions

4.2.1. Overview: Camera Control functions

Camera Control functions allows the user to control the status of the camera toolbar buttons.

CmdShowCameraToolbar toggles the camera toolbar.

CmdShowCameraControl shows and hides the camera settings dialog bar.

CmdShowRecordDialogBox is no more supported (deprecated)

CmdShowSaveSequenceDialogBox displays the save acquisition dialog box.

CmdRecord starts acquiring.

CmdPlay set the camera in play mode or it stops the camera if already in play mode (toggle).

CmdStop stops the camera, if in play mode. If in idle mode, it snaps a single frame or two frames if the camera mode is double exposure.

CmdRewind displays the sequence of the acquired frames in reverse order.

CmdFastForward displays the sequence of the acquired frames.

CmdShowPreviousFrame displays the previous image frame from the list of the acquired frames.

CmdShowNextFrame displays the next image frame from the list of the acquired frames.

GetImageDataPointer gets the pointer to the acquired image buffer.

GetImageFramesNumber gets the number of acquired frames.

GetImageFrameWidth gets the width of the image frame.

GetImageFrameHeight gets the height of the image frame.

4.2.2. CmdShowMainToolbar

short CmdShowMainToolbar(short *nShow* **)**

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

nShow

Specifies if the main toolbar is visible or not. It must be one of the following values:

FALSE: hides the toolbar.
TRUE: shows the toolbar.

Remarks

This function makes the main toolbar visible or not.

See also:

4.2.3. CmdShowCameraBar

void CmdShowCameraBar(short *nShow* **)**

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

nShow

Specifies if the camera bar is visible or not. It must be one of the following values:

FALSE: hides the bar.
TRUE: shows the bar.

Remarks

This function makes the camera bar visible or not.

See also:

4.2.4. CmdShowSaveSequenceDialogBox

void CmdShowSaveSequenceDialogBox (void)

Return values

None

Parameters

None

Remarks

The function displays the "save acquired sequence" modal dialog box.

See also:

4.2.5. CmdRecord

short CmdRecord(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function starts the acquisition of a sequence of images. The number of images can be set (**SetFramesNumber**) and read (**GetFramesNumber**).

See also: SetFramesNumber, GetFramesNumber

4.2.6. CmdPlay

short CmdPlay(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

If the camera is in idle mode, this function set the camera in play mode, otherwise it stops the camera.

See also: CmdStop

4.2.7. CmdStop

short CmdStop(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

If the camera is playing, the function stops the camera. If the camera is in idle mode, the function snaps a frame.

See also: CmdPlay

4.2.8. CmdRewind

short CmdRewind(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the sequence of the acquired frames in reverse order.

See also: CmdFastForward

4.2.9. CmdFastForward

short CmdFastForward(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the sequence of the acquired frames.

See also: CmdRewind

4.2.10. CmdShowPreviousFrame

short CmdShowPreviousFrame(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the previous image from the list of the acquired frames.

See also: CmdShowNextFrame

4.2.11. CmdShowNextFrame

short CmdShowNextFrame(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the next image from the list of the acquired frames.

See also: CmdShowPreviousFrame

4.2.12. CmdShowFirstFrame

short CmdShowFirstFrame(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the first frame of a recorded sequence.

See also: CmdShowLastFrame

4.2.13. CmdShowLastFrame

short CmdShowLastFrame(void)

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

None

Remarks

This function displays the last frame of a recorded sequence.

See also: CmdShowFirstFrame

4.2.14. GetImageWidth

long GetImageWidth()

Return values

The image width, in pixels

Parameters

None

Remarks

This function returns the current value in pixels of the image width.

See also: GetImageHeight

4.2.15. GetImageHeight

long GetImageHeight()

Return values

The image height, in pixels

Parameters

None

Remarks

This function returns the current value in pixels of the image height.

See also: GetImageWidth

4.2.16. GetImagePixelDepth

long GetImagePixelDepth( )

Return values

The image pixel depth (8 or 10 or 12)

Parameters

None

Remarks

This function returns the current value of the pixel depth of the image.

See also: GetImageWidth, GetImageHeight

4.2.17. GetExposure

long GetExposure()

Return values

The camera exposure, in microseconds

Parameters

None

Remarks

This function returns the current value in microseconds of camera exposure.

See also:

4.2.18. GetFramesNumber

long GetFramesNumber()

Return values

The number of allocate frames

Parameters

None

Remarks

This function returns the current value of the number of frames that can be acquired by calling the CmdRecord () routine. After the recording the data may be read by calling the GetFrames routine.

See also: CmdRecord, GetFrames

4.2.19. GetFrames

long GetFrames (long *nFrames***, long** *nStartIndex***, long** *pDataBuff* **)**

Return values

SVC_SUCCESS, if successful, otherwise

SVC_GENERICERROR, if an error occurs.

Parameters

nFrames

Specifies the number of frames to read.

nStartIndex

Specifies the index of the first frame to read.

pDataBuff

Specifies the pointer to data buffer.

Remarks

This function reads one or more frames from the controls memory to a data buffer specified by the pDataBuff parameter. The values of **nStartIndex** and **nStartIndex + nFrames** must be lower than the number of total frames that can be read by **GetFramesNumber**.

See also: GetFramesNumber

4.2.20. IsCameraOK

long IsCameraOK (void)

Return values

Returns 1 if the camera is initialized and open, 0 otherwise

Parameters

None

Remarks

This function returns the camera current status. Call this function before using any other routine, to check if the camera is initialized and open.

See also:

4.2.21. IsCameraRecording

long IsCameraRecording (void)

Return values

Returns 1 if the camera is recording frames, 0 otherwise

Parameters

None

Remarks

This function returns the camera current recording status. Call this function to check if the camera is recording frames or not.

See also:

4.2.22. SetExposure

void SetExposure (long *nExposure* **)**

Return values

None

Parameters

nExposure

Specifies the new camera exposure value in microseconds.

Remarks

This function sets a new value of exposure in the camera.

See also:

4.2.23. SetFramesNumber

void SetFramesNumber (long *nFrames* **)**

Return values

None

Parameters

nFrames

Specifies the new number of frames to set.

Remarks

This function sets the number of frames that will be acquired in the next acquisition. The value may be read by calling the **GetFramesNumber** routine.

See also: GetFramesNumber

5. sharpVISION™ TWAIN™ Driver Reference

5.1. Overview

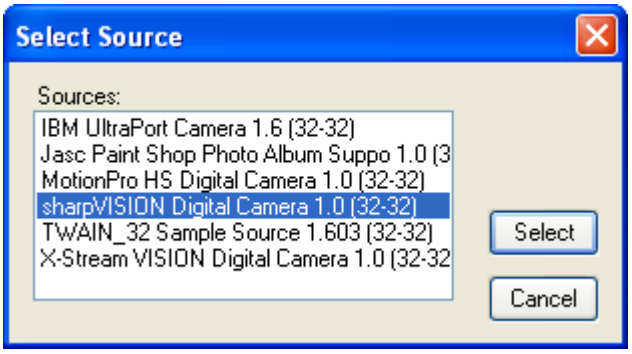
TWAIN defines a standard software protocol and API (application programming interface) for communication between software applications and image acquisition devices (the data source). The sharpVISION™ TWAIN driver complies with these specifications.

The sharpVISION™ TWAIN driver allows to acquire images from the sharpVISION digital camera and import them into many applications, such as Adobe Photoshop®, Adobe Image Ready®, Adobe Acrobat®, Microsoft Imaging®, Microsoft Office® and so on.

The applications that comply with the TWAIN specifications have added two options to their menu (Select Source and Acquire). The options allow the user to select the sharpVISION camera and import acquired images in the application.

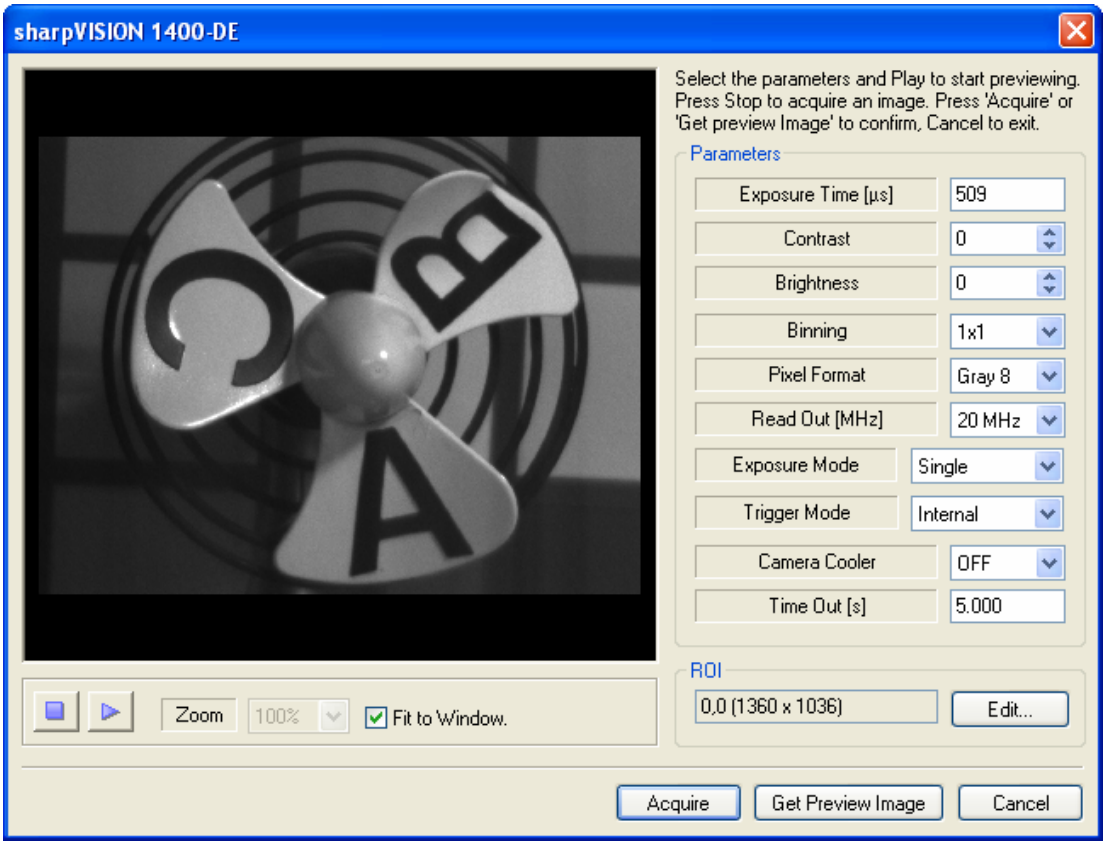
5.2. Select source

When the user chooses the Select Source option, the application requests that a Select Source dialog box is displayed. A list of all the available devices appears. The user may highlight and select the sharpVISION camera and click OK. See the picture below.



5.3. Acquire images

If the user selects the Acquire option, the sharp VISION User Interface is displayed.



The user may select the camera parameters and operate the camera by pressing one of the buttons described below. Then the currently displayed image may be imported in the host application by pressing the "**Acquire**" button or the "**Get Preview Image**" button.

5.3.1. Parameters

The camera configuration parameters may be interactively changed.

Exposure Time: the camera exposure may be adjusted. The value is in microseconds with increments of 1 microsecond.

Contrast and Brightness: the contrast and brightness of the image produced by the camera may be adjusted. The values range from –100 to 100.

Binning: the user may select the binning mode from 1x1 to 4x4. In this fashion pixels may be grouped to form a larger pixel, which results in added SNR and sensitivity. When this parameter is changed, the Region of Interest (ROI) is reset. The control is disabled when the camera is playing.

Pixel Format: the camera digitizer is a 12-bit digitizer. The output can be 8-bit (Gray8) or 10-bit (Gray16) (12-bit for 1500-EX camera model). The control is disabled when the camera is playing.

Readout Frequency: this parameter controls the speed at which the image data is read from the CCD. The supported values are: 20 MHz, 10 MHz, 5 MHz and 2MHz.

Exposure Mode: the camera exposure may be single or double.

Trigger Mode: the default state of the camera at start-up is in the Internal/continuous mode. In this mode the camera does not require a trigger input signal. The other camera modes (External Edge High, Edge Low, Pulse High and Pulse Low) require an external trigger input. In these modes the camera waits for a trigger signal to initiate the acquisition of a frame. There is a delay (latency) between the trigger event and the start of an acquisition of 12 microseconds.

Camera Cooler: the camera cooler feature may be activated. The parameter enables and disables the thermoelectric-cooler.

Acquisition Time Out: the acquisition and snap time out may be configured. The value is in seconds.

Region of interest: the camera ROI may be adjusted. The grayed edit boxes in the ROI group show the current ROI settings. The user may change them by pressing the "Edit…" button (see below). The button is disabled when the camera is playing.

5.3.2. Zoom Group

The way the preview window image is displayed may be adjusted.

Zoom: the image zoom may be adjusted. If the image is bigger than the preview window, the user may drag to pan over the image.

Fit to Window: if the check box is checked the image is automatically zoomed so that it fits on the current display window. The zoom combo box is disabled.

5.3.3. Camera Control Group

Play: it's the standard Play button of any camera interface. If the button is pressed, the camera starts previewing.

Stop: press this button to stop the camera play or to refresh the preview window with a new camera snapshot or stop the current acquisition if the record button has been previously pressed.

6. sharpVISION™ LabVIEW™ Interface Reference

6.1. Overview

The sharpVISION™ LabVIEW® Interface allows acquiring images and controlling the sharpVISION digital cameras from inside National Instruments LabVIEW application. It works with LabVIEW 6 and greater, on Windows 2000/XP. Windows NT is not supported.

The sharpVISION™ LabVIEW™ Interface includes the **VIs (Virtual Instruments)** for controlling the camera and a few example VIs to show how to use the interface: the camera Vis are packaged in a library called **IDT.LLB**) located in the **Idt** directory in the **user.lib** subdirectory of the LabVIEW folder. The examples are located in the **LabVIEW** subdirectory of the installation folder (C:\Program Files\IDT\sharpVISION).

The sharpVISION Vis may be accessed by selecting the “Show Functions Palette” menu item from the Window” menu, then by clicking the “User Libraries” button and the “IDT sharpVISION VIs” button.

The VI interface and examples are listed below.

6.2. Initialization VIs

6.2.1. Overview: Initialization VIs

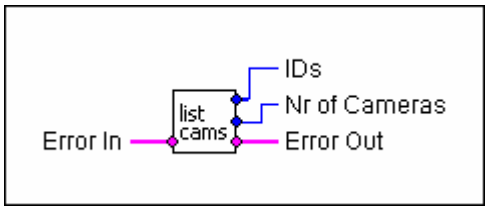
Initialization VIs allow the user to initialize the sharpVISION cameras, enumerate the available cameras, open and close them.

IDT List Cameras: it enumerates the sharpVISION cameras currently connected to the computer. Then it outputs a list of the IDs of the enumerated cameras.

IDT Open: it opens a sharpVISION camera. The user may open a specific camera supplying its unique ID (retrieved using the List Cameras VI) or open the first available camera supplying 0 as ID.

IDT Close: it closes a sharpVISION camera, previously open.

6.2.2. IDT List Cameras



Inputs

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

IDs

Specifies the array containing the IDs of the detected cameras

Nr of Cameras

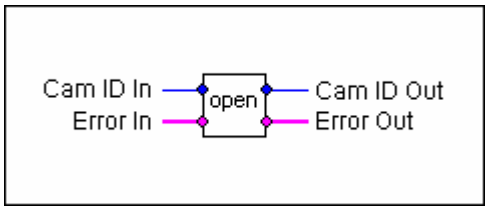
Specifies the number of detected cameras

Remarks

The VI enumerates the active cameras and returns a list of the detected cameras IDs. This VI must be set before **"IDT Open"** to find out which cameras are available. The "Nr of cameras" output contains the number of detected cameras. If any error occurs during the cameras enumeration, the Error Out terminal signals the error condition.

See also: "IDT Open"

6.2.3. IDT Open



Inputs

Camera ID

Specifies the ID of the camera to be opened, or 0 for the first available camera

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error condition

Camera ID

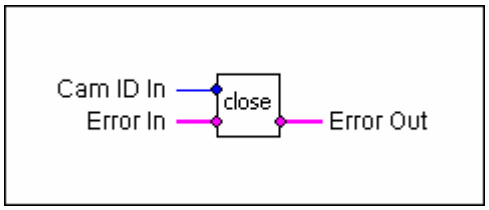
Specifies the ID of the opened camera

Remarks

The VI opens the camera with a specific ID. The value can be retrieved calling the **"IDT List Cameras"** VI. The user may supply a specific camera ID or 0: in this case the first available camera is opened. If any error occurs during the camera opening, the Error Out terminal signals this error. The VI also returns the ID of the open camera.

See also: "IDT Close"

6.2.4. IDT Close



Inputs

Camera ID

Specifies the ID of the camera to be closed

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error condition

Remarks

This VI closes a camera previously open. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Open”

6.3. Configuration VIs

6.3.1. Overview: Configuration VIs

Configuration VIs allow the user to read information from the camera, read configuration parameters from the camera and write configuration parameters to the camera.

IDT Get Info: it reads information from the camera, such as camera model, firmware version, sensor type, sensor model, etc. The info key is one of the input parameters.

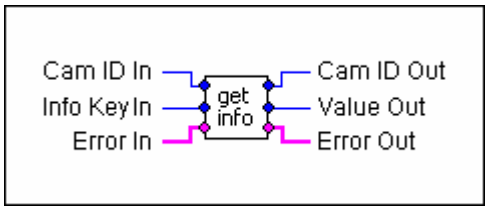
IDT Setup: it sets the configuration parameters to the camera: contrast, brightness, exposure time, binning and ROI.

IDT Get Param: it reads a specific parameter from the camera. The parameter key is one of the input parameters.

IDT Set Param: it writes a specific parameter to the camera. The parameter key is one of the input parameters.

IDT Trig Setup: it allows the user to configure the trigger mode.

6.3.2. IDT Get Info



Inputs

Camera ID

Specifies a valid camera ID

Info Key

Specifies which parameter has to be returned by the VI

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Value

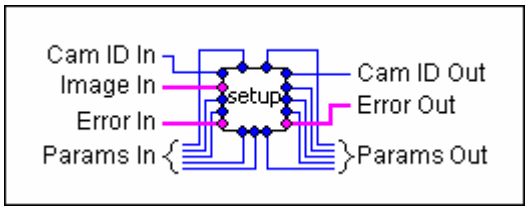
Specifies the value of the info parameter

Remarks

This VI returns camera specific information, such as sensor type or version numbers, generally state-independent information. See the Appendix B for a list of all the available Info Key values. If any error occurs during the operation, the Error Out terminal signals this error.

See also: "IDT Get Param"

6.3.3. IDT Setup



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Parameters

Each input is the value of a specific parameter: exposure time (in microseconds), contrast [0,4095], brightness [0,4095], binning (1x1, 2x2, 3x3 or 4x4), ROI (X, Y, width and height). Note that set 0 to exposure, binning, ROI Width or ROI height, means set it to the default value. For ex. Setting 0 to exposure you obtain an exposure value of 16384 microseconds.

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Parameters Out

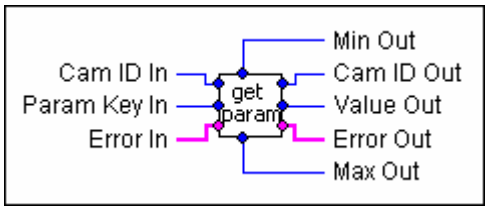
Specifies the values of the parameters

Remarks

This VI writes all the configuration parameters to the camera. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Get Param”, “IDT Set Param”

6.3.4. IDT Get Param



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Param Key

Specifies the index of the parameter

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Value

Specifies the current value of the parameter

Min

Specifies the minimum value of the parameter

Max

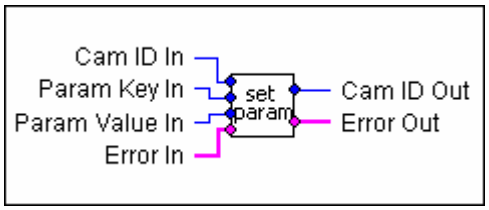
Specifies the maximum value of the parameter

Remarks

This VI reads a specific configuration parameter from the camera and returns the parameter value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Set Param”

6.3.5. IDT Set Param



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Param Key

Specifies the index of the parameter

Value

Specifies the value of the parameter

Outputs

Camera ID

Specifies the camera ID

Error

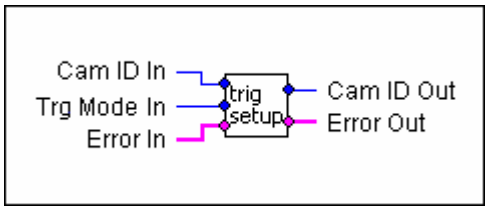
Specifies the return error condition

Remarks

This VI writes a specific configuration parameter to the camera. The parameter key is one of the input parameters. A list of the parameters indexes is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Get Param”

6.3.6. IDT Trig Setup



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Trigger Mode

Specifies the trigger mode parameter.

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Remarks

This VI configures the trigger mode to the camera (internal, external edge-low, edge-high, pulse-low and pulse-high). If any error occurs during the operation, the Error Out terminal signals this error.

See also: "IDT Get Param"

6.4. Grab VIs

6.4.1. Overview: Grab VIs

Grab VIs allow the user to grab images from the camera.

IDT Snap: this VI executes a complete acquisition sequence. It opens the first available camera, captures an image and closes the camera.

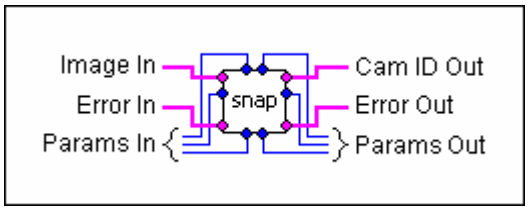
IDT Grab Setup: this VI prepares the camera to acquire images. Use this VI before IDT Grab Acquire.

IDT Grab: this VI acquires an image. It waits for a new image to be available and then transforms that image into an IMAQ Image. To abort the acquisition, set the IDT Grab Abort global variable to TRUE. The image grab is synchronous and the VI exits when the frame has been grabbed or a time out occurs.

IDT Grab Stop: this VI stops capturing images. Use this VI after IDT Grab.

IDT Grab Abort: this is a global variable. Set this to TRUE to abort the IDT Grab. Note that after the VI has aborted, this variable is reset to FALSE.

6.4.2. IDT Snap



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Parameters

Specifies the parameters to configure the camera (exposure, binning and ROI).

Image

Specifies the image object input

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Image

Specifies the image object output

Parameters

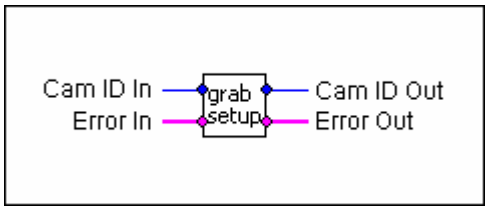
Specifies the parameters values

Remarks

This VI snaps an image from the camera. A complete acquisition sequence is executed. The camera is opened and configured, then one image is snapped and the camera is closed. The image grab is synchronous and the function exits when the frame has been snapped. The image format depends on the image size (ROI) and pixel depth.

See also:

6.4.3. IDT Grab Setup



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

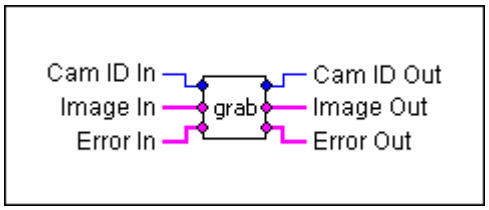
Specifies the return error condition

Remarks

This VI prepares a camera to acquire images. The VI must be called before any call to the “IDT Grab” VI. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Grab“, “IDT Grab Stop“

6.4.4. IDT Grab



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Image

Specifies the image object input

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Image

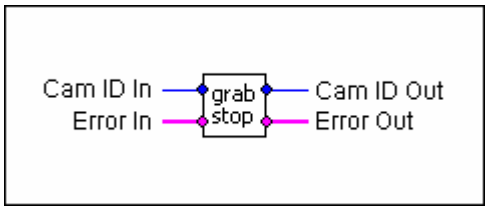
Specifies the image object output

Remarks

this VI acquires an image. It waits for a new image to be available and then transforms that image into an IMAQ Image. To abort the acquisition, set the IDT Grab Abort global variable to TRUE. The image grab is synchronous and the VI exits when the frame has been grabbed or a time out occurs.

See also: “IDT Grab setup“, “IDT Stop Grab“

6.4.5. IDT Grab Stop



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

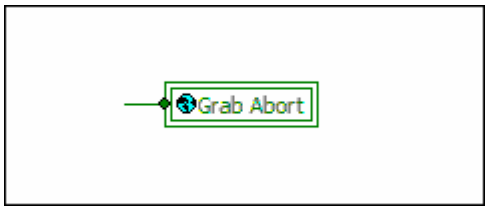
Specifies the return error condition

Remarks

This VI stops any camera acquisition previously started. If any error occurs during the operation, the Error Out terminal signals this error.

See also: “IDT Grab Setup“

6.4.6. IDT Grab Abort



Inputs

Variable value

Specifies the value to set the variable

Outputs

None

Remarks

this is a global variable. Set this to TRUE to abort the IDT Grab. Note that after the VI has aborted, this variable is reset to FALSE.

See also: “IDT Grab“, “IDT Stop Grab“

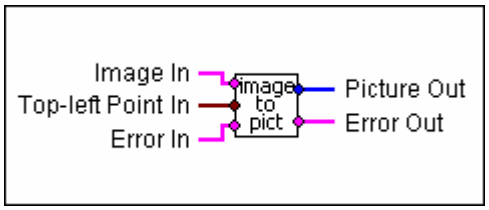
6.5. Miscellaneous VIs

6.5.1. Overview: Miscellaneous VIs

Miscellaneous VIs allow the user to convert image formats and manage the error conditions in the sharpVISION VIs.

IDT Image To Picture converts an IMAQ image to a LabVIEW picture.

6.5.2. IDT Image To Picture



Inputs

Error

Specifies a standard error cluster input terminal

Top-Left point

Specifies the coordinates of the top-left point

Image

Specifies the image to convert

Outputs

Error

Specifies the return error condition

Picture

Specifies the output LabVIEW picture object

Remarks

This VI converts an IMAQ image object into a LabVIEW picture object. If any error occurs, the Error Out terminal signals this error.

See also:

6.6. How to use the VIs

6.6.1. Snapping an image

The easiest way to acquire an image is to use the IDT Snap VI. It opens the first available camera, configures it, acquires an image and then closes the camera. Optionally you may set the exposure, binning and ROI parameters.

6.6.2. Opening and closing a camera

A camera must be opened before using its functions and then it must be closed. To open a specific camera you have to supply to the Open VI the unique ID of that camera. You can also supply 0 to open the first available camera. To obtain the list of all available cameras you can use the “IDT List Cameras” VI.

6.6.3. Configuring a camera

The camera configuration parameters may be set using a single VI which configures all the camera parameters (IDT Setup). The parameters that may be written by this VI are: exposure time; contrast and brightness [-100,100]; binning; ROI (X, Y, width, height) as an array of four unsigned long. If you want to read or write a single parameter you may respectively use the “IDT Get Param” VI or the “IDT Set Param”.

6.6.4. Acquiring images

The correct sequence to grab an image is to setup the driver to capture the image (IDT Grab Setup VI), acquire the image (IDT Grab) and stop the acquisition (IDT Grab Stop). You may obtain better performance in acquiring multiple images if you run the IDT Grab Setup VI once, then a loop of IDT Grab Acquire, followed by one IDT Grab Stop. If you need to abort the acquisition of an image you can set the global variable IDT Grab Abort to TRUE. This variable will reset itself after the image is aborted.

6.6.5. Triggering

Trigger modes can be set to five different values (internal, external edge-low, external edge-high, external pulse-low, external pulse-high). In internal mode the camera acquires images using internal frame rate, while in external modes it requires an external TTL-level trigger signal.

6.6.6. Error handling

The LabVIEW interface uses the standard error cluster found in many LabVIEW Vis. The error cluster includes status, code and source parameters. When an error occurs, status is set to TRUE, source is set to the VI that caused the error, and code is set to one of the values in the table below.

Error Code	Description
1	Driver Fault
2	Camera not found
3	Bad image format
4	Invalid parameter value
5	Parameter or info not supported
100	Generic error

6.7. Examples VIs

6.7.1. 1_enum_cameras

It's a very simple example which shows how to display the result of a cameras enumeration. The output of the "IDT List Cameras" VI is displayed in a group of four LED and four edit boxes. If a camera is enumerated the corresponding LED is turned on and the camera ID is displayed in the edit box.

6.7.2. 2_getinfo

This VI shows how to retrieve information from the camera such as camera model, firmware version, etc. The first available is open and the following information is retrieved and displayed: camera model, CCD model, firmware version.

6.7.3. 3_snap

This example shows how to capture and display a single image. The example opens the first available camera, configures it with the default parameters, and it acquires a single image. The acquisition output is displayed in a preview window, and then the camera is closed.

6.7.4. 4_acquire

This example shows how to capture and display a stream of monochrome images. The example opens the first available camera, configures it with the default parameters, and it continuously acquires and displays a single image. When the stop button is pressed the camera is closed.

6.7.5. 5_acquire_with_error_check

This example shows how to capture and display a stream of monochrome images and how to handle an error condition. The example opens the first available camera, configures it with the default parameters, it continuously acquires and displays a single image. When the stop button is pressed the camera is closed.

6.7.6. 6_acquire_with_parameters

This example shows how to capture monochrome images and interactively configure the camera. The example opens the first available camera and allows the user to configure the following parameters: exposure time, contrast, brightness, binning and ROI. Note that set 0 to exposure, binning, ROI Width or ROI height, means set it to the default value. For ex. Setting 0 to exposure you obtain an exposure value of 16384 microseconds. Then the camera is configured and a single image is acquired. The acquisition output is displayed in a preview window.

6.7.7. 7_acquire_with_trigger

This example shows how to capture monochrome images and interactively configure the camera trigger mode. The example opens the first available camera and allows the user to configure the trigger mode. Then the camera is configured and a single image is acquired. The acquisition output is displayed in a preview window.

7. sharpVISION™ MATLAB™ Interface Reference

7.1. Overview

The sharpVISION™ MATLAB™ Interface allows to acquire images and to control the sharpVISION digital cameras from inside the Mathworks™ MATLAB application. The interface works with MATLAB 6.5 and greater, on Windows 2000/XP Professional. Windows NT is not supported.

The sharpVISION™ MATLAB™ Interface includes the 'MEX' file for controlling the camera (packaged in a library called sharpML.dll) and a few example .m files to show how to use the interface.

Every routine may be called from a MATLAB™ script file in the form:

[output1, output2 ...] = SharpML [input1, input2 …]

The number of inputs and outputs depends on the function selected. In any function call input1 is the name of the requested command (for example, 'IdtSvEnumCameras') and output1 is the result of the operation (0 = SUCCESS, otherwise ERROR).

More details on the commands syntax may be retrieved by typing "help SharpML" at MATLAB command prompt or opening the file **SharpML.m** with a text editor.

The MATLAB interface reflects the SDK Application Program Interface (see sharpVISION SDK reference section) with a few exceptions. The MATLAB interface and examples are listed below.

The interface is supported in Windows version of sharpVISION SDK only.

7.2. Initialization Functions

7.2.1. Overview: Initialization functions

Initialization functions allow the user to initialize the sharpVISION camera, enumerate the available cameras, open and close them.

IdtSvVersion: this function retrieves the driver version.

IdtSvEnumCameras: this function enumerates the IDs of the sharpVISION cameras connected o the computer.

IdtSvOpenCamera: this function opens an sharpVISION camera. The user may open a specific camera supplying its ID or open the first available camera supplying 0 as ID.

IdtSvCloseCamera: this function closes a sharpVISION camera previously opened.

7.2.2. IdtSvVersion

[*strVersion***]** = SharpML (‘**IdtSvVersion**’)

Inputs

None

Outputs

strVersion

Specifies the driver version string (for example, ‘2.03’)

Remarks

This function must be called to retrieve the sharpVISION™ MATLAB interface version string.

See also:

7.2.3. IdtSvEnumCameras

[*nResult*, *nItems*, *svArray***] =** SharpML **('IdtSvEnumCameras')**

Inputs

None

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nItems

Specifies the number of detected cameras

svArray

Specifies the array containing the IDs of the detected cameras

Remarks

The routine enumerates the active cameras and return an array filled with the detected cameras IDs. This routine must be called before **IdtSvOpenCamera** to find out which cameras are available. The nItems variable contains the number of detected cameras. If any error occurs during the cameras enumeration, the nResult variable contains an error code.

See also: IdtSvOpenCamera

7.2.4. IdtSvOpenCamera

[*nResult*, *nCameraId***] =** SharpML **(‘IdtSvOpenCamera’**, *nInputId***)**

Inputs

nInputId

Specifies the ID of the camera to be opened, or 0 for the first available camera

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nCameraId

Specifies the ID of the opened camera

Remarks

The routine opens the camera whose ID is in the variable *nInputId*. The value can be retrieved calling the **IdtSvEnumCameras** enumeration function. The user may supply a specific camera ID or 0: in this case the first available camera is opened. If any error occurs during the camera opening, the routine returns an error code in the nResult variable, otherwise it returns 0. The function also returns the camera Id.

See also: IdtSvCloseCamera

7.2.5. IdtSvCloseCamera

[*nResult***]** = SharpML ('**IdtSvCloseCamera**', *nCameraId*)

Inputs

nCameraId

Specifies the ID of the camera to be closed

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function closes a camera previously open. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: IdtSvOpenCamera

7.3. Configuration functions

7.3.1. Overview: Configuration functions

Configuration functions allow the user to read information from the camera, read configuration parameters from the camera and write configuration parameters to the camera.

IdtSvGetCameraInfo: this function reads information from the camera, such as camera model, firmware version, sensor type, sensor model, etc. The info key is one of the input parameters.

IdtSvGetParameter: this function reads a specific parameter from the camera. The parameter key is one of the input parameters.

IdtSvSetParameter: this function writes a specific parameter to the camera. The parameter key is one of the input parameters.

IdtSvConfigure: this function writes the configuration parameters to the camera: contrast, brightness, exposure time, image format, ROI, and binning.

IdtSvSetTriggerMode this function sets the trigger mode.

7.3.2. IdtSvGetCameraInfo

[*nResult*, *nInfoValue***] =** SharpML **('IdtSvGetCameraInfo',** *nCameraId***,** *nInfoKey***)**

Inputs

nCameraId

Specifies a valid camera ID

nInfoKey

Specifies which parameter the function has to return

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nInfoValue

Specifies the value of the info parameter

Remarks

This function returns camera specific information, such as sensor type or version numbers, generally state-independent information. See the Appendix B for a list of all the available nInfoKey values.

See also:

7.3.3. IdtSvGetParameter

[*nResult, nValue, nMinValue, nMaxValue***] =** SharpML **('IdtSvGetParameter'**, *nCameraId*, *nParamKey***)**

Inputs

nCameraId

Specifies a valid camera ID

nParamKey

Specifies the index of the parameter

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nValue

Specifies the current value of the parameter

nMinValue

Specifies the minimum value of the parameter

nMaxValue

Specifies the maximum value of the parameter

Remarks

This function reads a specific configuration parameter from the camera and returns the parameter value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: IdtSvSetParameter

7.3.4. IdtSvSetParameter

[*nResult***] =** SharpML **('IdtSvSetParameter'**, *nCameraId*, *nParamKey, nValue***)**

Inputs

nCameraId

Specifies a valid camera ID

nParamKey

Specifies the index of the parameter

nValue

Specifies the value of the parameter

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function writes a specific configuration parameter to the camera. The parameter key is one of the input parameters. A list of the parameters indexes is available in Appendix C. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: IdtSvGetParameter

7.3.5. IdtSvConfigure

[*nResult***] =** SharpML **('IdtSvConfigure',** *nCameraId*, *nFormat*, *nBinning, nRoiX*, *nRoiY*, *nRoiWidth*, *nRoiHeight*, *nExposure*, *nContrast*, *nBrightness***)**

Inputs

nCameraId

Specifies a valid camera ID

nFormat

Specifies the value of the image format (Gray8, Gray16).

nBinning

Specifies the value of the binning (1x1, 2x2, 3x3, 4x4).

nRoiX, nRoiY, nRoiWidth, nRoiHeight

Specifies the values of the ROI.

nExposure

Specifies the value of the exposure time in µs.

nContrast

Specifies the contrast [0,4095].

nBrightness

Specifies the brightness [0,4095].

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function writes all the configuration parameters to the camera. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0. Note that for several parameters (nBinning, nRoiWidth, nRoiHeight and nExposure) , setting their value to 0 means to set them to the default value. For ex. If you set 0 to exposure, you obtain a value of 16384 for exposure.

See also: IdtSvSetParameter

7.3.6. IdtSvSetTriggerMode

[*nResult***] =** SharpML **('IdtSvSetTriggerMode',** *nCameraId*, *nTriggerMode***)**

Inputs

nCameraId

Specifies a valid camera ID

nTriggerMode

Specifies the value to set for the trigger mode (0 = Internal, 1=Ext. Edge High, 2=Ext. Edge Low, 3=Ext. Pulse High, 4=Ext. Pulse High).

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0).

Remarks

This function sets the trigger mode.

See also: IdtSvSetParameter

7.4. Grab Functions

7.4.1. Overview: Grab Functions

Preview Mode grab functions allow the user to snap and acquire images from the camera.

IdtSvStartAcquire: this function starts an acquisition in the camera. It sends the configuration to the camera and starts the acquisition process.

IdtSvAcquire: this function grabs an image from the camera. The image grab is synchronous and the function exits when the frame has been grabbed or a time out occurs.

IdtSvStopAcquire: this function stops the current acquisition.

IdtSvImageIsReady: this function is used to know whether the image acquisition has been completed and ready to be read.

7.4.2. IdtSvStartAcquire

[nResult**] =** sharpML **(‘IdtSvStartAcquire’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function starts an acquisition in the camera. It sends the configuration to the camera and starts the acquisition process.

See also: IdtSvStopAcquire

7.4.3. IdtSvAcquire

[*nResult, image***] =** sharpML **(‘IdtSvAcquire’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

image

Specifies the array where the acquired image is stored

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

It grabs an image (or two) from the camera. The image grab is synchronous and the function exits when the frame has been grabbed or a time out occurs. If the camera mode is set to double exposure, two frames are acquired and the function outputs two image buffers (arrays), otherwise only the first is valid. The array dimension depends on the image size and pixel depth: if the pixel depth is 8, the array is an ‘**unsigned char**’ array; if the pixel depth is 10, the array is an ‘**unsigned short**’ array.

See also:

7.4.4. IdtSvImageIsReady

[*nResult, nIsReady***]** = sharpML ('**IdtSvImageIsReady**', *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nIsReady

It is 0 if image acquisition has not been completed yet, or 1 if the image acquisition has been completed.

Remarks

this function is used to know whether the image acquisition has been completed and ready to be read. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: IdtSvAcquire

7.4.5. IdtSvStopAcquire

[*nResult***]** = sharpML ('**IdtSvStopAcquire**', *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function stops any camera acquisition previously started. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: IdtSvStartAcquire

7.5. How to use the Interface functions

7.5.1. Opening and closing a camera

A camera must be opened before using its functions and then it must be closed. To open a specific camera you have to supply to the IdtSvOpenCamera function the ID of that camera. You can also supply 0 to open the first available camera. To obtain the list of all available cameras you may use the IdtSvEnumCameras function.

7.5.2. Configuring a camera

The camera configuration parameters may be set using a single function which configures all the camera parameters (IdtSvConfigure). The current camera parameters may be read using the IdtSvGetParameter routine. The parameters that may be read or written by these routines are: exposure time; pixel size (8,10); binning; ROI (X, Y, width, height) as an array of four unsigned long; trigger mode.

If you want to read or write a single parameter you may respectively use the IdtSvGetParameter or the IdtSvSetParameter routines.

7.5.3. Acquiring images

The correct sequence to grab an image is to setup the driver to capture the image (**IdtSvStartAcquire**), test if an image is ready to be acquired (**IdtSvImageIsReady**), read the image (**IdtSvAcquire**) and stop the acquisition (**IdtSvStopAcquire**). You may obtain better performance in acquiring multiple images if you call the IdtSvStartAcquire once, then a loop of IdtSvImageIsReady/IdtSvAcquire, followed by one IdtSvStopAcquire.

7.5.4. Error handling

The sharpVISION MATLAB interface returns the same error codes displayed in the Error Handling section of the LabVIEW interface reference topic.

7.6. Examples

7.6.1. IdtSvEnumEx

This example shows how to obtain the list of all available cameras.

7.6.2. IdtSvInfoEx

This example shows how to obtain some information from the camera.

7.6.3. IdtSvReadParmEx

This example shows how to read a specific parameter from the camera.

7.6.4. IdtSvSnapEx

This example shows how to capture and display a single image.

7.6.5. IdtSvSaveEx

This example shows how to capture and save a stream of 5 monochrome images as bitmap files.

7.6.6. IdtSvDisplayEx

This example shows how to capture and display a stream of monochrome images and set all camera parameters like binning, image format, exposure, ROI, contrast, brightness, exposure and trigger mode.

8. Appendix

8.1. Appendix A - Return Codes

The following table shows the values of the codes returned by the sharpVISION APIs. The values can be found in the **SharpAPI.h** header file in the **Include** subdirectory.

Code	Value	Notes
SVC_SUCCESS	0	OK – No errors
SVC_NOTSUPPORTED	1	Function is not supported for this device
SVC_INVALIDVALUE	2	Invalid parameter value
SVC_BADSETTINGS	3	Bad SV_SETTINGS structure
SVC_NOUSERDRIVER	4	
SVC_NOFIREWIREDRIVER	5	No fire-wire device driver is installed
SVC_DRIVERCONNECTION	6	Problems on driver connection
SVC_DRIVERALREADYLOADED	7	Too many calls to SvLoadDriver – the driver is already loaded
SVC_DRIVERNOTLOADED	8	The driver is not loaded yet
SVC_INVALIDHANDLE	9	Invalid camera handle
SVC_UNKNOWNCAMERA	10	Unknown camera model for this driver version
SVC_INVALIDCAMERAID	11	Invalid camera id used in SvOpenCamera
SVC_NOMORECONNECTIONS	12	Obsolete
SVC_HARDWAREFAULT	13	Hardware fault
SVC_FIREWIREFAULT	14	Fire-wire fault
SVC_CAMERAFAULT	15	Camera fault
SVC_DRIVERFAULT	16	Driver fault
SVC_INVALIDFRAMEINDEX	17	Invalid frame index
SVC_BUFFERTOOSMALL	18	Frame buffer is too small for the acquired image
SVC_OUTOFMEMORY	19	Out of memory error
SVC_OUTOFSHAREDMEMORY	20	Out of shared memory error
SVC_BUSY	21	Fire-wire is busy
SVC_QUEUEFULL	22	Driver queue is full
SVC_CANCELLED	23	Operation cancelled
SVC_NOTSTREAMING	24	Streaming must be set to on before calling this function
SVC_LOSTSYNC	25	Lost synchronization, invalid frame
SVC_BLACKFILL	26	This frame is damaged, some data is missing
SVC_FIREWIREOVERFLOW	27	Fire-wire overflow - restart streaming
SVC_UNPLUGGED	28	Camera has been unplugged or turned off
SVC_ACCESSDENIED	29	The camera is already open
SVC_STRMFAULT	30	Stream allocation failed
SVC_NEEDUPDATE	31	The driver needs update
SVC_ROITOOSMALL	32	The region of interest is too small

8.2. Appendix B – Information Parameters

The following table shows the values and a brief description of the parameters that can be read calling the **SvGetCameraInfo** routine. The numeric values of the parameters can be found in the **SharpAPI.h** header file in the **Include** subdirectory.

Parameter	Description
SVI_CAMERATYPE	Camera Model (see SV_CAM_TYPE in SharpAPI.h)
SVI_SERIALNR	Camera Serial Number
SVI_HW_VERSION	Hardware version
SVI_FW_VERSION	Firmware version
SVI_CCD	CCD Model (see SV_CCD_TYPE in SharpAPI.h)
SVI_BIT_DEPTH	Maximum number of bits
SVI_COOLED	1 if the camera has a cooler, otherwise 0
SVI_RESERVED1	Factory tests values
SVI_IMG_WIDTH	Width of ROI, in pixels
SVI_IMG_HEIGHT	Height of ROI, in pixels
SVI_IMG_SIZE	Image size, in bytes
SVI_CCD_TYPE	CCD type (0:monochrome, 1: color)
SVI_CCD_WIDTH	Maximum CCD width
SVI_CCD_HEIGHT	Maximum CCD height
SVI_FW_BUILD	Camera firmware build (version)
SVI_UNIQUE_ID	Camera unique ID (serial number)

8.3. Appendix C – Camera Settings

The following table shows the values and a brief description of the parameters that can be read and written in the camera. The numeric values of the parameters can be found in the **SharpAPI.h** header file in the **Include** subdirectory.

Parameter	Description
SVP_GAIN	Camera gain (gain on CCD output)
SVP_OFFSET	Camera offset (offset in CCD ADC
SVP_EXPOSURE	Camera exposure in microseconds
SVP_BINNING	Binning (1x1, 2x2, 3x3, 4x4)
SVP_READOUT	Read out frequency (20 MHz, 10 MHz, 5 MHz, 2.5 MHz)
SVP_TRIGGERTYPE	Trigger type (free-run, hardware and software trigger)
SVP_COOLERACTIVE	1 turns the cooler on, 0 turns it off
SVP_IMGFORMAT	Image format (see SV_IMG_FMT in SharpAPI.h)
SVP_ROIX	Upper left x coordinate of ROI
SVP_ROIY	Upper left y coordinate of ROI
SVP_ROIWIDTH	Width of ROI, in pixels
SVP_ROIHEIGHT	Height of ROI, in pixels
SVP_DBL_EXP	Set to 1 for double exposure mode

8.4. Appendix D – Data types

This appendix describes the data types defined in the **SharpAPI.h** header file.

8.4.1. SV_CAM_TYPE

The SV_CAM_TYPE type enumerates the camera models.

SV_CT_UNKNOWN: Unknown camera model

SV_CT_1300DE: sharpVISION 1300-DE.

SV_CT_1400DE: sharpVISION 1400-DE.

SV_CT_1500EX: sharpVISION 1500-EX.

8.4.2. SV_CCD_TYPE

The SV_CAM_TYPE type enumerates the CCD models.

SV_CCD_UNKNOWN: unknown CCD type.

SV_CCD_ICX085AL: Sony ICX085AL CCD.

SV_CCD_ICX285AL: Sony ICX285AL CCD.

SV_CCD_ICX205AL: Sony ICX205AL CCD.

8.4.3. SV_EXP_MODE

The SV_CAM_MODE enumerates the camera operation modes:

SV_EM_SINGLE: single exposure.

SV_EM_DOUBLE: double exposure.

8.4.4. SV_TRIG_MODE

The SV_TRIG_MODE enumerates the trigger mode:

SV_TM_FREERUN: continuous acquisition (fast)

SV_TM_EDGE_HI: exposure starts on edge, active High.

SV_TM_EDGE_LO: exposure starts on trigger, active Low.

SV_TM_PULSE_HI: exposure integrated over pulse, active High.

SV_TM_PULSE_LO: exposure integrated over pulse, active Low.

SV_TM_SOFTWARE: exposure starts on software trigger.

8.4.5. SV_BINNING

The SV_BINNING enumerates the binning values:

SV_BIN_1X1: binning 1x1.

SV_BIN_2X2: binning 2x2.

SV_BIN_3X3: binning 3x3.

SV_BIN_4X4: binning 4x4.

8.4.6. SV_READOUT_SPEED

The SV_READOUT_SPEED enumerates the camera readout frequencies:

SV_ROS_20MHZ: 20 MHz.

SV_ROS_10MHZ: 10 MHz.

SV_ROS_05MHZ: 5 MHz.

SV_ROS_02MHZ: 2.5 MHz.

8.4.7. SV_IMG_FMT

The SV_IMG_FMT enumerates the image formats:

SV_IF_RAW8: raw CCD output (8 bit).

SV_IF_RAW16: raw CCD output (10 bit)

SV_IF_GRAY8: gray 8 bit data.

SV_IF_GRAY: gray 16 bit data.

8.4.8. SV_CALLBACK_FLAGS

The SV_CALLBACK_FLAGS enumerates the Queue callback flags:

SV_CF_DONE: callback is called when the operation is completed.

8.4.9. SV_ERROR

The SV_ERROR enumerates the return codes. See Appendix A.

8.4.10. SV_INFO

The SV_INFO enumerates the camera information index. See Appendix B.

8.4.11. SV_PARAM

The SV_PARAM enumerates the parameters. See Appendix C.

8.5. Appendix E – Structures

This appendix describes the structures defined in the **SharpAPI.h** header file.

8.5.1. SV_SETTINGS

The SV_SETTINGS structure is an opaque structure that contains the all the camera parameters in compact format. The user may access the structure using the SvSetParameter and SvGetParameter routines.

```
typedef struct
{
     unsigned long size;
     unsigned long _private_data[ 40 ];
} SV_SETTINGS, *PSV_SETTINGS;
```

Members

size

It specifies the size of the structure. Must be set to sizeof (SV_SETTINGS), otherwise the related functions doesn't work.

_private_data

It specifies the opaque structure data, an array of 40 unsigned long values.

8.5.2. SV_ENUMITEM

The SV_ENUMITEM structure contains information about a camera. It must be used in the camera enumeration procedure with the SvEnumCameras routine.

```
typedef struct
{
     unsigned long cameraId;
     unsigned long cameraType;
     unsigned long uniqueId;
     unsigned long isOpen;
     unsigned long _reserved[ 10 ];
} SV_ENUMITEM, *PSV_ENUMITEM;
```

Members

cameraId

It specifies the ID which identifies a camera among others. The user must use this camera id to open the camera with SvOpenCamera.

cameraType

It specifies the camera model (sharpVISION 1300-DE, 1400-DE or 1500-EX).

uniqueId

It specifies a32-bit unique number, stored in the camera.

bIsOpen

It specifies whether the camera is currently open or not.

_reserved

Reserved for future use

8.5.3. SV_FRAME

The SV_FRAME structure contains information about the image frame to be grabbed. It is used to acquire images in the SvGrabOneFrame or SvQueueOneFrame routines.

```
typedef struct
{
     void*         pBuffer;
     unsigned long bufferSize;
     unsigned long format;
     unsigned long width;
     unsigned long height;
     unsigned long size;
     unsigned long bits;
     unsigned short frameNumber;
     unsigned long reserved;
     unsigned long errorCode;
     unsigned long timestamp;
     unsigned long _reserved[ 8 ];
} SV_FRAME, *PSV_FRAME;
```

Members

pBuffer

Specifies the pointer to the data

bufferSize

Specifies the data buffer size, in bytes

format

Specifies the image format; this field is filled when the grab routine returns

width

Specifies the image width; this field is filled when the grab routine returns

height

Specifies the image height; this field is filled when the grab routine returns

size

Specifies the image size, in bytes; this field is filled when the grab routine returns

bits

Specifies the image pixel depth; this field is filled when the grab routine returns

frameNumber

Specifies the rolling frame number.

reserved

Reserved for future use

errorCode

Specifies the result code of the Grab operation

timeStamp

Specifies the exposure time stamp.

_reserved

Fill bytes

8.5.4. SV_AsyncCallback

The SV_AsyncCallback is the prototype of the callback function passed to the SvQueueOneFrame or **SvQueueCameraSettings** routines. The callback is called by the driver when the change settings operation is completed. From the callback body only a subset of the sharpVISION API can be called.

```
typedef void (SHARPAPI *SV_AsyncCallback)
(
     void*         userPtr;
     unsigned long userData,
     SV_ERROR      errCode,
     unsigned long flags
);
```

Members

userPtr

Specifies a user defined data pointer, passed to the SvQueue routine.

userData

Specifies a user defined data value, passed to the SvQueue routine.

errCode

It specifies the operation return code.

flags

It specifies a combination of the SV_CALLBACK_FLAGS values.