Snappy

Snappy Programmer's Reference

Release Version 3.0

Copyright © 1996-1999 by Play Incorporated.

Snappy is a registered trademark of Play Incorporated.

Visual Basic, Visual C++, ActiveX, OLE, and Windows are either registered trademarks or trademarks of Microsoft Corporation.

All other trademarks are the property of their respective owners.

Table of Contents

Introduction

The **Snappy Programmer's Reference** provides a programmer with information required for creating custom applications for Snappy. The Snappy API provides two independent and mutually exclusive methods for controlling the Snappy hardware. The API provided by the embeddable OLE/OCX control is easy to use and provides all the functionality of the Snappy application itself. There are also interfaces within the Snappy DLL which can be accessed directly through COM. In general, using the COM objects will be more complicated than using the OCX and some niceties such as the ability to save image files are not available through the COM interface.

Roadmap

This section contains brief descriptions of OLE and COM, and references to additional information.

The **Snappy Programmer's Reference** contains the following chapters:

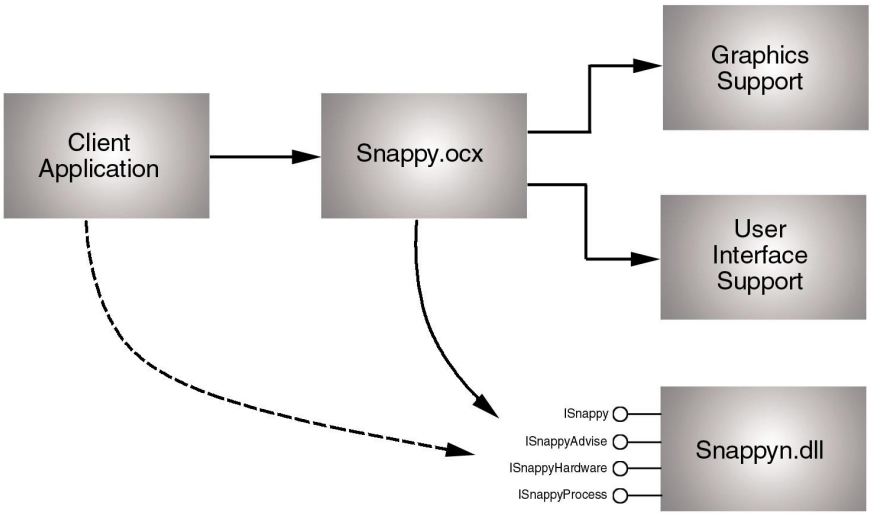
- **Introduction** provides some sense of direction for newcomers to the COM object model by providing links to lots of help, references, and tutorials. The section *Graphic Orientation to the Snappy API* shows the relationship between various components in the Snappy API. The section *Snappy Data Processing Steps* shows the different processing phases that occur when using Snappy.
- **Snappy OLE Control** is the reference for the OLE control object provided by *Snappy.ocx*
- **Snappy COM Interface** is the reference for the COM object interface to *Snappyn.dll* or *Snappyp.dll*, defined by *Snappy.idl.* (*Snappyn.dll* is the NTSC version while *Snappyp.dll* is the PAL version.)

Graphic Orientation to the Snappy API

Figure 1 shows the two Snappy APIs: the Snappy OLE Control, provided through *Snappy.ocx*, and the COM interfaces to Snappy's Dynamic Link Library, *Snappyn.dll* or *Snappyp.dll* (NTSC and PAL versions).

Most applications will use the API provided through the embedded control, *Snappy.ocx*, since it provides complete access to file processing, GUI objects, and low-level Snappy processing DLLs.

Some applications may provide their own file and GUI facilities in place of those provided by Play Incorporated. These applications may access the COM interfaces to *Snappyn.dll* or *Snappyp.dll* directly without going through *Snappy.ocx*, as indicated by the dashed line in Figure 1.



High-Level API	Snappy.ocx	Snappy OLE Control
Low-Level API	Snappyn.dll or Snappyp.dll	Snappy dynamic link library
Low-Level Support Libraries	Graphics	Set of objects for loading, saving, and translating different file and memory based graphics image formats
	User Interface	Snappy user interface objects

Figure 1. Hierarchy of Snappy components

Snappy Data Processing Steps

Figure 2 illustrates the different phases that a signal progresses through from the initial video input to a final picture.

During the first phase, the video signal is acquired and analyzed. The first phase results in unprocessed Snappy data which is stored in memory. This memory is accessed with the **ISnappyProcess** memory members: *UseSnappyData,* and *GetSnappyData*. The members: *LoadSnappyData* and *SaveSnappyData* provide file storage capability.

In the second phase, Snappy data is actually processed to generate an RGB Device-Independent Bitmap (DIB).

An application program can take advantage of the separate phases by instantiating different objects to handle the second phase. After Snappy produces the unprocessed data, indicated by a notification event, some instantiation of a second-phase object can call *GetSnappyData* to make a copy of the data. Snappy may then immediately

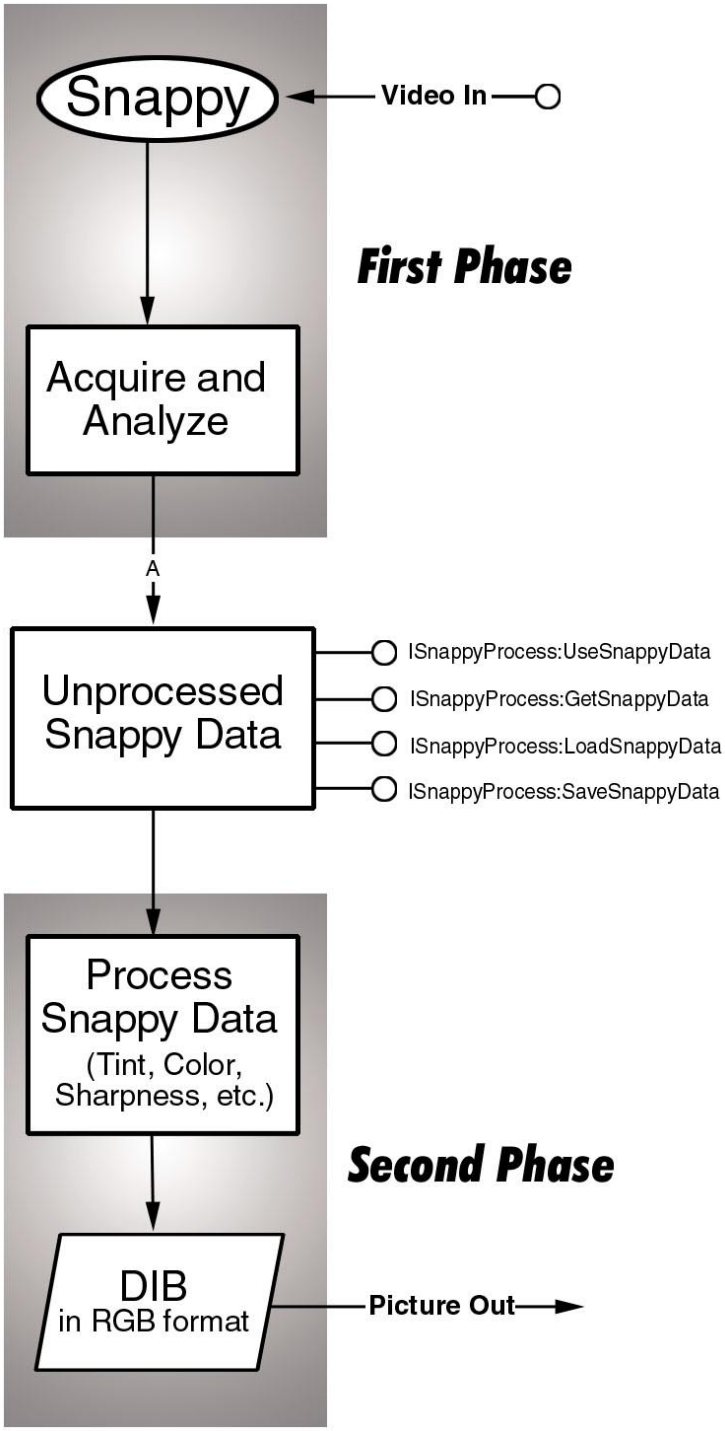


Figure 2. Snappy Data Processing Blocks

resume capturing unprocessed data. Meanwhile, the second-phase object uses the copy of the previous data through a call to *UseSnappyData*, and begins processing with whatever it wants to do. In this manner, multiple second-phase objects can perform concurrent independent processing, while Snappy continues to create unprocessed data.

Snappy OLE Control

The Snappy OLE Control, *Snappy.ocx*, provides a method to completely control Snappy and the Snappy application interface through an independent developer's application. The control may be inserted into container applications. It should be noted that container applications exhibit many different behaviors with respect to OLE control objects. Controls that work perfectly with one container may fail to work in another and Snappy is no different.

Note The Snappy hardware must be present on the machine for this to work!

The embeddable control behaves similarly to the Snappy application: please refer to the Snappy user's manual for information on using Snappy.

You can also write container applications using Visual Basic or Visual C++ to control the methods and properties for the embedded control.

Where to Find More Information

The chapter *Snappy OLE Control* assumes some familiarity with OLE controls. If you are new to OLE control programming, please refer to the appropriate Microsoft documentation as well as the documentation for your programming environment.

Snappy COM Interface

The Snappy COM interface, *Snappy.idl*, provides a method to control Snappy through certain low-level methods. Note that this interface does not make any provisions for a user interface or file access--these must be provided entirely by the independent developer's application.

Where to Find More Information

The chapter *Snappy COM Interface* assumes a solid understanding of COM programming. While not necessarily so, this chapter assumes that the programmer is using the Visual C++ language and WIN32 SDK environments. If you are new

to programming with the Component Object Model (COM), please refer to the appropriate Microsoft documentation, such as: the **OLE Programmer's Reference** and the **OLE COM Tutorial** included in the **WIN32 SDK Online Help**. A complete book on the subject is: Brockschmidt, Kraig, **Inside Ole**, Second Edition, Microsoft Press, 1995.

Snappy
OLE Control

This chapter covers *Snappy.ocx*, the OLE control for controlling Snappy from custom applications. (This chapter assumes some familiarity with OLE. Please refer to the appropriate Microsoft documentation for more information.)

Note If you have a release of *Snappy.ocx* dated prior to 11-28-98, please discard it and use the new version instead!

Getting Started

An easy way to see how the Snappy OLE Control works, is to drop the control into a container provided by Microsoft Visual C++ or Visual Basic then simply peruse to see what it does.

License

Before you can use *Snappy.ocx*, the license file, *Snappy.lic* must be located in the same directory. The license is distributed with the Snappy release.

Purpose of the OLE Reference Sections

The following sections specify the *Snappy.ocx* features:

- **OLE Control Properties**
- **OLE Control Methods**
- **OLE Control Events**

OLE Control Properties

Most Snappy properties fall into one of three categories: *Acquire*, *Adjust*, and *Save*. The *Acquire* properties set Snappy parameters for acquiring the video snapshot. The *Adjust* properties set Snappy parameters for processing Snappy data and creating a DIB. The *Save* properties are settings for saving the processed image.

The syntax for accessing the properties will vary depending upon the development environment. Following are examples for Visual Basic and Visual C++. If you

are using a different environment you will need to use the appropriate conventions for that environment.

In Visual Basic 5, you simply use the name of the property and the language will determine if you are getting or setting the property from the context in which you are using the property name. For example, if you have a Snappy control named *SnappyCtrl* embedded in a form, you can use the syntax
SnappyCtrl.AutomaticSnapMode = 0
to set a property value and you can use the syntax
snapmode = SnappyCtrl.AutomaticSnapMode
to retrieve a property value.

Visual C++ 5, on the other hand, requires that you use the prefixes *Get* and *Set* with the property name when accessing a property. If you have a member variable of type CSnappy named *m_SnappyCtrl* you would use the syntax
m_SnappyCtrl.SetAutomaticSnapMode(0);
to set a property value and the syntax
nSnapMode = m_SnappyCtrl.GetAutomaticSnapMode();
to retrieve a property value.

Example programs files are available for reference purposes. You should also refer to the appropriate documentation for working with OLE controls within your development environment.

AutomaticSnapMode ID 8

short AutomaticSnapMode;

Indicates the snapping mode used when the *Snap()* method is invoked. This correlates directly to the "Snap Type" setting within the Snappy Application.

Valid values are:
0 -- Normal Snapping
1 -- Continuous Snapping
2 -- Countdown Snapping, with delay determined by *DelaySnapTime*

It should be noted that some programming environments will only allow the "Normal" snap mode to be used. This is determined by how the programming environment embeds the control within the application. Furthermore, a programming environment may not be able to set this property to a particular value at design time but the application being developed may be able to set it to the same value at run time. If the value is unavailable at run time then an Error event will be fired when the client tries to set the property to that value.

See Also *DelaySnapTime*

AutoShow ID 6

boolean AutoShow;

Set to TRUE to show a picture full screen at the end of processing and wait for a keyboard input or mouse click--requires user input when TRUE.

AutoSnapTime ID 9

short AutoSnapTime;

Not currently implemented.

Blue ID 24

short Blue;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

Brightness ID 18

short Brightness;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

CameraSource ID 13

boolean CameraSource;

If TRUE, sets *time-based* input (*camera*). If FALSE, sets *tape* input.

ColorSource ID 14

boolean ColorSource;

If TRUE, sets to *color* input. If FALSE, sets to *black-and-white* input.

Contrast ID 19

short Contrast;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

CropBottom ID 29

short CropBottom;

Set the offset, in field coordinates, of the bottom of the source cropping rectangle from that of the maximum field size. It does not affect the size of the DIB.

CropLeft ID 27

short CropLeft;

Set the offset, in field coordinates, of the left side of the source cropping rectangle from that of the maximum field size.

CropRight ID 28

short CropRight;

Set the offset, in field coordinates, of the right side of the source cropping rectangle from that of the maximum field size.

CropTop ID 26

short CropTop;

Set the offset, in field coordinates, of the top of the source cropping rectangle from that of the maximum field size.

DelaySnapTime ID 10

short DelaySnapTime;

Sets the Delay for *AutomaticSnapMode,* in seconds.

See Also *AutomaticSnapMode*

Gamma ID 20

short Gamma;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

Green ID 23

short Green;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

ImageAspectCorrect ID 4

boolean ImageAspectCorrect;

If TRUE, maintains the 4:3 aspect ratio of the image when changing one of the *ImageWidth* or *ImageHeight* settings by adjusting the opposite control to match the aspect ratio. This only affects subsequent settings of the *ImageWidth* and *ImageHeight* properties. The current property values will remain unaffected.

ImageBitDepth ID 33

short ImageBitDepth;

The highest bit depth that images will be processed to. The bit depth of the DIBs created by the *Snap()* and *Process()* methods will be the lesser of this value and the bit depth of the display.

Valid values are:
4 -- 16 colors
8 -- 256 colors
16 -- 65K colors (High Color)
24 -- 16 million colors (True Color)

See Also *ImageOptimizedPalette*

ImageHeight ID 32

short ImageHeight;

The image height in pixels. If *ImageAspectCorrect* is TRUE, then setting this value will cause the *ImageWidth* property to change also.

See Also *ImageAspectCorrect*

ImageOptimizedPalette ID 5

boolean ImageOptimizedPalette;

Optimizes the palette based upon the colors in the captured video.

ImageWidth ID 31

short ImageWidth;

The image width in pixels. If *ImageAspectCorrect* is TRUE, then setting this value will cause the *ImageHeight* property to change also.

See Also *ImageAspectCorrect*

LPTPort ID 12

short LPTPort;

The current LPT port that Snappy is assigned to (typically 1-3). Setting this property to zero will cause the default port value stored in the system registry to be used. The OCX, upon instatiation, will use the value in the registry by default so typically this value does not need to be set.

Negative ID 15

boolean Negative;

If TRUE, the input is considered to be a *negative* and thus the colors are complemented to create a normal color image. This conversion is done during Snappy data acquisition and does not effect the process stage.

NewWndOpt ID 7

boolean NewWndOpt;

If TRUE, new windows are opened for each snapped image. This only applies if the Snappy user interface is visible.

Picture ID 30

IPictureDisp* Picture;

This is the active picture. Getting this value gives you the picture that was last snapped or processed. Setting this value allows you to select a previously active image for saving. (If the image to be saved was the last one snapped or processed there is no need to set this property.)

PreviewTimeOut ID 35

short PreviewTimeOut;

The preview time out value. Previewing will stop after the specified number of seconds. A value of zero disables the time out feature. Some programming environments do not support the preview time out feature so it will always be disabled in applications created with those environments.

PreviewType ID 53

short PreviewType;

The type of preview image.

Valid values are:
0 -- Black and White Preview
1 -- Color Preview

If the *ColorSource* property is FALSE, the preview will be black and white regardless of this setting.

ProcessFilter ID 54

short ProcessFilter;

The type of filtering to be used during processing.

Valid values are:
0 -- Fast (lowest quality)
1 -- Normal
2 -- Slow (High Definition mode)

Red ID 22

short Red;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

Saturation ID 17

short Saturation;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

SaveFileCompression ID 2

short SaveFileCompression;

For file types that support compression, this specifies the amount of compression desired. The range will vary with the file format. Not all formats support compression.

For the version of Snappy addressed by this document, only two file formats support compression. For JPEG files, this value actually relates to the JPEG quality factor and thus ranges from 0 to 100 with 100 being highest quality and thus lowest compression. For TIFF files, a value of 0 indicates no compression, 1 indicates RLE compression, and 2 indicates JPEG compression. LZW compression is not supported due to licensing issues.

SaveFileName ID 1

BSTR SaveFileName;

The filename to store the image data under. This is the full path, filename, and extension. Note that the extension specified as part of this filename has no bearing on the format used to store the file.

See Also *SaveFileType*, *PictureNameChanged*

SaveFileType ID 3

BSTR SaveFileType;

Indicates the file *type* by file *extension*.

Currently, the supported types are:
BMP -- Windows Bitmap
IFF -- Amiga Interchange File Format
JPG -- JPEG
PCX -- PC Paintbrush
PNG -- Portable Network Graphic
TGA -- Targa
TIF -- Tagged Image File

Note that this determines only the format used to save the file – it does not determine the file extension of that file. The file extension is specified as part of the filename.

Sharpness ID 21

short Sharpness;

Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

SnapMode ID 11

short SnapMode;

Sets the current snap mode. This is equivalent of the "Picture Quality" setting within the Snappy application.

Valid values are:
0 -- Moving
1 -- Still
2 -- High Quality
3 -- Highest Quality

Termination ID 25

short Termination;

Indicates whether Snappy is terminating the video source or if another device such as a monitor is terminating the source through the Snappy *Video Thru* connector.

Tint ID 16

short Tint;
Value to be used on the next Snappy data process phase. Legal values range from -50 to 50.

UnprocessedSnappyData ID 36

OLE_HANDLE UnprocessedSnappyData;

The handle to the snapped data that Snappy processes into an image. Snapping creates this data, processing uses it to create the final image. See Figure 2.

Any call to retrieve this value should be paired with a call to *FreeUnprocessedSnappyData()* when the client no longer needs the data. Failure to do so will result in a memory leak. Multiple instances of Snappy Data may be kept by the client and selected as the current data by setting this property.

If data is successfully acquired (the return value is not NULL) then the client may invoke the *Process()* method after setting this property to the data previously obtained.

See Also *FreeUnprocessedSnappyData(), Process(), Snap(), SnapOnly()*

VideoFormat ID 34

BSTR VideoFormat;

0 NTSC, requires *Snappyn.dll*
1 PAL, requires *Snappyp.dll*

OLE Control Methods

This section covers the dispatch interface for the Snappy OLE Control.

Abort ID 40

boolean Abort();

Aborts a *Preview(), Snap(),* or *SnapOnly()* that is in progress.

AboutBox ID DISPID_ABOUTBOX

boolean AboutBox();

Invoking this function displays Snappy's About box.

CanProcess ID 46

boolean CanProcess();

Indicates if the *Process()* method can process the current unprocessed data using the current process settings.

CopyToClipboard ID 47

boolean CopyToClipboard();

Copies the current processed image bitmap to the Windows Clipboard.

FreeUnprocessedSnappyData ID 56

boolean FreeUnprocessedSnappyData(OLE_HANDLE hSnappyData);

Frees the Snappy Data created by retrieving the *UnprocessedSnappyData* property. All calls that retrieve the property value should be paired with a call to this method or a memory leak will result.

See Also *UnprocessedSnappyData*

GetMaxCropHeight ID 49

short GetMaxCropHeight();

Returns the maximum useable video field height.

GetMaxCropWidth ID 48

short GetMaxCropWidth();

Returns the maximum useable video field width.

PrepareToSnap ID 39

boolean PrepareToSnap();

Turn Snappy on so that next *Snap()* or *SnapOnly()* occurs as quickly as possible.

Preview ID 38

boolean Preview();

Start the preview process. The preview will be of the type specified by the *PreviewType* property. Each time a preview image is ready, the *PreviewAvailable* event will be fired.

Process ID 52

boolean Process();

Process the current snap data using the current process settings to create a DIB. This is invoked after the *SnapOnly()* method has resulted in the *DataAvailable* event being fired. After processing is complete, the *PictureAvailable* event is fired.

The *Process()* method can be called multiple times with the same snap data but different process settings and each will result in its own DIB. Thus by using the *SnapOnly()* and *Process()* methods you could allow the user to adjust process properties such as *Contrast* and reprocess a snap without resnapping.

See Also *Snap(), SnapOnly()*

ProcessLine ID 45

boolean ProcessLine(short nLine, long* pBits);

Process one line of snap data. Clients using this OLE control will usually not invoke this method and will instead invoke either the *Snap()* method which automatically does processing or the *SnapOnly()* and *Process()* methods.

SavePicture ID 42

boolean SavePicture();

Saves the current picture according to the settings previously made with the *Save* properties.

SaveSettingsAsDefault ID 43

boolean SaveSettingsAsDefault();

Sets the current settings such as snap mode, input source, and source type as the default in the system registry.

ShowPictureAndWait ID 50

boolean ShowPictureAndWait();

Takes the current Snappy picture, displays it full screen, then waits for a mouse click or keypress to continue. If no picture is available then FALSE is returned.

ShowUI ID 41

boolean ShowUI(boolean bShow);

Opens or Closes the Snappy user interface as determined by the parameter bShow.

Snap ID 37

boolean Snap();

Snap a picture using the current snap mode then automatically process it with the current process settings and create a DIB. When the processed image is ready, the *PictureAvailable* event is fired. This does NOT allow direct reprocessing of snaps using the *Process()* method. If the image needs to be adjusted, the *Snap()* method can be invoked again after the appropriate settings have been changed -or- the client may retreive the *UnprocessedSnapData* property, set it back, alter the desired process settings, then invoke the *Process()* method.

See Also *UnprocessedSnapData, SnapOnly(), Process()*

SnapOnly ID 55

boolean SnapOnly();

Snap a picture using the current snap mode but do not process the data. When the snap is complete and the snap data is ready the *DataAvailable* event is fired. To get a DIB that can be used, the *Process()* method must be invoked. Using the *SnapOnly()* method results in snap data that can be processed over and over without resnapping. The client need not worry about the *UnprocessedSnapData* property as the data is managed within the OCX when this method is used. This method may only be invoked if the UI is not currently being shown.

See Also *Snap(), Process()*

UseDefaultSettings ID 44

boolean UseDefaultSettings();

Get the default property values such as snap mode, input source, and source type from the system registry.

VerifyPicSaved ID 51

boolean VerifyPicSaved();

This function checks to see if all snapped pictures up to this point have been saved. A good place to call this is when handling the *RequestCloseUI* event.

OLE Control Events

This section covers the event dispatch interface for Snappy OLE Control. These are events that are fired upon the conditions specified. Your application can have routines that handle each of these events as required and/or desired.

BatteryLowChanged ID 7

void BatterLowChange(boolean bLow);

The state of Snappy's voltage has changes. If *bLow* is TRUE, the battery is low.

DataAvailable ID 14

void DataAvailable();

The snap data is available. This event is fired only after the *SnapOnly()* method is invoked. It indicates that the snap has taken place and the data is ready to be processed into an image using the *Process()* method.

Error ID DISPID_ERROREVENT

void Error(short Number, BSTR* Description, long Scode, BSTR Source,
BSTR HelpFile, long HelpContext, boolean* CancelDisplay);

Standard OLE Error notification called when the Snappy control has an error.

Following is a list of the most likely error codes you will see:
57 – Snappy was busy.
68 – Snappy not found (or LPT port not available).
604 – Settings changed while snapping.
607 – Timer Failure. The OCX experienced problems with a timer event.
(Does your programming environment allow Snappy to use timers?)
608 – File save failed.

FieldSnapped ID 1

void FieldSnapped();
One video field was just snapped.

PictureAvailable ID 2

void PictureAvailable();

A processed picture (DIB) is available. Your application needs to examine the *Picture* property to retrieve the image.

PictureNameChanged ID 12

void PictureNameChanged(BSTR PicName);

The control's *SaveFileName* property changed. If *SaveFileName* was the default filename, *SaveFilename* is modified every time the picture is snapped.

PowerChanged ID 5

void PowerChanged(boolean bOn);

The Snappy power state changed. If *bOn* is TRUE, the power is now on. If FALSE, the power is off.

PreviewAvailable ID 3

void PreviewAvailable();

A preview image is available for display. Your application needs to examine the *Picture* property to retrieve the preview image. Note that a color preview DIB does not have square pixels like most people assume. If you render it assuming square pixels, the image will look very squished. You need to render it into a rectangle with a 4:3 aspect ratio. B&W preview does have square pixels.

PreviewTimeout ID 15

void PreviewTimeout();

The Snappy was previewing and the time period defined by the *PreviewTimeOut* property has passed. This event indicates that the Snappy is no longer previewing and thus no more *PreviewAvailable* events will be fired until the *Preview()* method is invoked again.

Some programming environments do not allow the preview time-out feature to function. Applications developed in such environments will never receive this event as the feature is effectively disabled.

ReadyToSnap ID 4

void ReadyToSnap();

The Snappy hardware is powered up and ready to capture video.

RequestCloseUI ID 13

void RequestCloseUI(boolean* Cancel);

When the Snappy user interface is shown and the user requests the UI be closed (by clicking on the close window button for example), this event is sent so that a confirmation box may be presented. Set Cancel to FALSE to allow the close.

See Also *VerifyPicSaved*

UIClose ID 11

void UIClose();

This event indicates that the Snappy user interface successfully changed from the open state to the closed state.

UIOpened ID 10

void UIOpened();

This event indicates that the Snappy user interface successfully changed from the closed state to the open state.

VideoAvailChanged ID 9

void VideoAvailChanged(boolean bAvail);

The state of the input video has changed. If *bAvail* is FALSE, video is not available.

VideoHotChanged ID 8

void VideoHotChanged(boolean bHot);

The state of Snappy's video signal legality has changed. If *bHot* is TRUE, the video is hot.

VideoThruChanged ID 6

void VideoThruChanged(boolean bUsingThru);

The termination changed. If *bUsingThru* is TRUE the Snappy is terminating (not Video Thru – no monitor is attached). FALSE indicates Snappy is not terminating (Video Thru – a monitor is attached).

Snappy
COM Interface

This chapter covers the set of COM interfaces for accessing *snappyn.dll.* These interface are:

- **ISnappy** for loading and saving defaults, and for snapping data.
- **ISnappyAdviseSync** for obtaining notification about processing status.
- **ISnappyHardware** for obtaining hardware status.
- **ISnappyPreview2** for previewing a snap before actually snapping.
- **ISnappyProcess** for processing the snapped data.

(The information in this chapter assumes a solid understanding of the Component Object Model, or COM. Please see the references noted in the chapter *Introduction*, section *Roadmap* for more information.)

Getting Started

Using the MIDL Compiler

The *idl* file (for interface definition language), *snappy.idl*, is to be passed through the *midl* command-line compiler. *Midl* will analyze the snappy.idl file and produce files named dlldata.c, snappy.h, snappy_i.c, and snappy_p.c.

What Are All These Files For?

The file *snappy.idl* of course, is the definition of the interface that you input into *midl*. The *midl* outputs are:

- dlldata.c
- snappy.h
- snappy_i.c
- snappy_p.c

dlldata.c

You may delete this file. It isn't necessary.

Snappy.h

The interface header file, *snappy.h,* contains type definitions and function declarations based on the interface definition in the IDL file. Include *snappy.h* in the source file for your client application.

Snappy_i.c

The interface UUID file contains the actual definitions of the IIDs and CLSIDs, which your client uses for interface identification. When you build your client, make and link *snappy_i.c* as part of the project.

Snappy_p.c

This would be the source for the RPC proxy/stub dll for passing interface function arguments and return values across process/machine boundaries (marshaling), if we were remoting Snappy. But we're not--*snappyn.dll* is strictly to be used as an in-process server on a local machine, so you can throw away *snappy_p.c*.

Interface ISnappy

The **ISnappy** interface provides functionality for:

- Loading and saving default settings for the snap mode, the input source, and the source type.
- Snapping picture data with or without processing, and aborting snaps.

Data Members

ESnapMode Enumeration of modes: *Moving*, *Still*, *HighQuality*, and *HighestQuality*.

ISnappy::ESnapMode

typedef enum tagSnapMode
{

Moving,	1 video field
Still,	1 field, looked at twice to get better color
HighQuality,	2 fields (frame), looked at twice (full color frame)
HighestQuality	Special 35ns mode (requires time-based input)

} ESnapMode;

See Also *GetSnapMode*, *SetSnapMode*

Function Members

Snappy Settings

Set

SetAdvise	Sets an optional **AdviseSync** interface to be used to notify snap progress, when snap is complete, and/or errors.
SaveCurrentSettingsAsDefault	Sets the current *SnapMode*, *InputSource*, and *SourceType* as the default in the system registry.
SetSnapMode	Set the current *SnapMode.*
SetInputSource	Set the current input *source. (camera or tape)*
SetSourceColor	Set the current input *type. (color or black-and-white)*
SetSourceNegative	Set the current negative input option.

Get

GetDefaultSettings	Get the default *SnapMode*, *InputSource*, and *SourceType* from the system registry.
GetSnapMode	Get the current *SnapMode*.
IsSourceCamera	Get current *camera* vs. *tape* setting.
IsSourceColor	Get current *color* vs. *black-and-white* setting.
IsSourceNegative	Get current *normal* vs. *negative* setting.

Snapping Functions

Snap	Snap a picture now (using current settings). Process it with the current process settings (set with the **ISnappyProcess** interface), and create a DIB.
SnapToUnprocessedData	Snap data now (using current settings).
Abort	Abort snapping.

ISnappy::Abort

HRESULT Abort();

Abort snapping.

See Also *ISnappyAdviseSync::OnWarning, ISnappyPreview::Abort, ISnappyProcess::Abort*

ISnappy::GetDefaultSettings

HRESULT GetDefaultSettings();

Get the default *SnapMode*, *InputSource*, and *SourceType* from the system registry.

See Also *GetSnapMode, SaveCurrentSettingsAsDefault, IsSourceCamera, IsSourceColor, IsSourceNegative*

ISnappy::GetSnapMode

HRESULT GetSnapMode([out] ESnapMode* pSnapMode);

pSnapMode Pointer to the *SnapMode* variable that will be set.

Get the current *SnapMode*.

See Also *ESnapMode, SetSnapMode, GetDefaultSettings, SaveCurrentSettingsAsDefault*

ISnappy::IsSourceCamera

HRESULT IsSourceCamera([out] BOOL* pbCamera);

pbCamera Pointer to variable to be set to TRUE if camera input, FALSE if not.

Get current *camera* vs. *tape* setting.

See Also *GetDefaultSettings, SetInputSource, IsSourceColor, IsSourceNegative*

ISnappy::IsSourceColor

HRESULT IsSourceColor([out] BOOL* pbColor);

pbColor Pointer to variable that will be set TRUE if color input, FALSE if B&W.

Get current *color* vs. *B&W* setting.

See Also *GetDefaultSettings, SetSourceColor, IsSourceCamera, IsSourceNegative*

ISnappy::IsSourceNegative

HRESULT IsSourceNegative([out] BOOL* pbNegative);

pbNegative Pointer to variable that will be set TRUE if *negative* input, FALSE if *normal.*

Get current *normal* vs. *negative* setting.

See Also *GetDefaultSettings, SetSourceNegative, IsSourceCamera, IsSourceColor*

ISnappy::SaveCurrentSettingsAsDefault

HRESULT SaveCurrentSettingsAsDefault();

Sets the current *SnapMode*, *InputSource*, and *SourceType* as the default in the system registry.

See Also *GetDefaultSettings, SetAdvise, SetSnapMode, SetInputSource, SetSourceColor, SetSourceNegative*

ISnappy::SetAdvise

HRESULT SetAdvise([in] ISnappyAdviseSync* pNotify);

pNotify Pointer to the interface to call for notification.

Sets an optional **AdviseSync** interface to be used to notify snap progress, when snap is complete, and/or errors.

Note *pNotify* may be NULL; if NULL, calls that use *Advise* will not return until they are complete.

See Also **ISnappyAdviseSync,** *ISnappyHardware::SetAdvise, ISnappyPreview::SetAdvise, ISnappyProcess::SetAdvise*

ISnappy::SetInputSource

HRESULT SetInputSource([in] BOOL bCamera);

bCamera If TRUE, sets *camera* (time-based) input otherwise, sets *tape* input.

Set the current input source.

See Also *GetDefaultSettings, IsSourceCamera, SaveCurrentSettingsAsDefault*

ISnappy::SetSnapMode

HRESULT SetSnapMode([in] ESnapMode eSnapMode);

eSnapMode The *SnapMode* variable to be used.

Set the current *SnapMode*

Note If *eSnapMode* is HighestQuality, the *InputSourceType* is assumed to be a *camera* (time-based) input.

See Also *ESnapMode, GetSnapMode, SaveCurrentSettingsAsDefault*

ISnappy::SetSourceColor

HRESULT SetSourceColor([in] BOOL bColor);

bColor If TRUE, sets to *color* input, otherwise *black-and-white* input.

Set the current input type.

See Also *IsSourceColor, GetDefaultSettings, SaveCurrentSettingsAsDefault*

ISnappy::SetSourceNegative

HRESULT SetSourceNegative([in] BOOL bNegative);

bNegative If TRUE, sets to *negative* input, otherwise *normal* input.

Set the current *negative* input option.

See Also *IsSourceNegative, GetDefaultSettings, SaveCurrentSettingsAsDefault*

ISnappy::Snap

HRESULT Snap([in] HGLOBAL hPackedDIB);

hPackedDIB Handle of DIB to snap into. May be NULL.

Snap a picture now (using current settings). Process it with the current process settings (set with the **ISnappyProcess** interface), and create a DIB. An **ISnappyAdviseSync** notification interface must have been previously defined through *SetAdvise* in order to be notified when the DIB is ready.

Note

- If the *Snappy* hardware is busy snapping, *SNAPERR_BUSY* is returned.
- If an *hPackedDIB* is provided and it is not adequate (width, height, etc.), then *E_BADARG* is returned.
- If *hPackedDIB* is NULL, a packed DIB is allocated and returned as a parameter of *ISnappyAdviseSync::OnDIBReady*. It is up to the caller to free the memory.

See Also *ISnappyAdviseSync::OnDIBReady, SetAdvise, SnapToUnprocessedData, Abort*

ISnappy::SnapToUnprocessedData

HRESULT SnapToUnprocessedData();

Snap data now (using current settings).

Note An **ISnappyAdviseSync** notification interface must have been previously defined (with *SetAdvise*) in order to be notified when the *UnprocessedData* is ready.

See Also *ISnappyAdviseSync::OnDataReady, SetAdvise, Snap, Abort, ISnappyProcess::GetSnappyData*

Interface ISnappyAdviseSync

ISnappyAdviseSync provides a notification interface for various events:

- OnDataReady
- OnDIBReady
- OnError
- OnFieldSnapped
- OnHotVideoChanged
- OnLineProcessed
- OnLowBatteryChanged
- OnPowerChanged
- OnPreviewImageReady
- OnReadyToSnap
- OnStatusUpdate
- OnVideoAvailableChanged
- OnVideoThruChanged
- OnWarning

Note Use the appropriate *SetAdvise* function listed below in order to receive notification of desired events.

See Also *ISnappy::SetAdvise, ISnappyPreview::SetAdvise, ISnappyProcess::SetAdvise*

Data Members

ESnappyWarning	Enumeration for video warnings.
ESnappyPhase	Enumeration for the Snappy phases during a snap.

ISnappyAdviseSync::ESnappyPhase

typedef enum tagSnappyPhase
{

SP_Acquiring,	Indicates that Snappy is performing video-acquisition.
SP_Analyzing,	Indicates that Snappy is performing pre-processing analysis.
SP_Processing	Indicates that Snappy is processing.

} ESnappyPhase;

Enumeration for the Snappy phases during a snap.

See Also *OnStatusUpdate*

ISnappyAdviseSync::ESnappyWarning

typedef enum tagSnappyWarning
{

SW_VideoLevelLow,	Video signal is not usable (double terminated?)
SW_NotCamera	Camera mode was asked for, but the video is not time-based.

} ESnappyWarning;

Enumeration for video warnings.

See Also *OnWarning*

Function Members

OnDIBReady	A processed DIB is ready.
OnDataReady	Snapped data is ready to be processed.
OnFieldSnapped	One field of video was just snapped.
OnLineProcessed	A line of *SnappyData* has been processed.
OnReadyToSnap	Snappy is powered up and ready to capture video.
OnPowerChanged	Snappy turned on or off.
OnVideoThruChanged	Snappy termination changed.
OnPreviewImageReady	A preview image is ready to be displayed.
OnLowBatteryChanged	The state of Snappy's voltage has changed.

OnHotVideoChanged	The state of Snappy's video signal legality has changed.
OnVideoAvailableChanged	The state of the input video has changed.
OnError	An error occurred.
OnStatusUpdate	Progress indication.

ISnappyAdviseSync::OnDataReady

HRESULT OnDataReady();

Snapped data is ready to be processed.

Note Use *ISnappyProcess::GetSnappyData* to retrieve a copy of the data.

See Also *ISnappyProcess::GetSnappyData, ISnappyProcess::ProcessSnappyData, ISnappy::SnapToUnprocessedData, ISnappyProcess::SetAdvise, ISnappy::SetAdvise*

ISnappyAdviseSync::OnDIBReady

HRESULT OnDIBReady([in] HGLOBAL hPackedDIB);

hPackedDIB Handle to a packed DIB (*BITMAPINFOHEADER* and bits).

A processed DIB is ready.

See Also *ISnappy::Snap, ISnappyProcess::ProcessSnappyData*

ISnappyAdviseSync::OnError

HRESULT OnError([in] int nErrorNum, int nCheckPoint);

eErrorNum A number specifying the type of error.
eCheckpoint A number specifying the location at which the error was trapped.

An error occurred.

ISnappyAdviseSync::OnFieldSnapped

HRESULT OnFieldSnapped();

One field of video was just snapped.

ISnappyAdviseSync::OnHotVideoChanged

HRESULT OnHotVideoChanged([in] BOOL bHot);

bHot TRUE if video is hot (signal too high), FALSE if not.

The state of Snappy's video signal legality has changed.

Note Hot video areas are displayed *red* in the preview.

See Also **ISnappyHardware,** *OnPreviewReady*

ISnappyAdviseSync::OnLineProcessed

HRESULT OnLineProcessed([in] WORD nLine, [in] BYTE* pBits);

nLine Current line.
pBits Pointer to line of 24-bit RGB triples.

A line of *SnappyData* has been processed.

See Also *ISnappyProcess::ProcessSnappyDataLine*

ISnappyAdviseSync::OnLowBatteryChanged

HRESULT OnLowBatteryChanged([in] BOOL bLow);

bLow TRUE if Battery is Low, FALSE if not.

The state of Snappy's voltage has changed.

Note This is only called when Snappy is *previewing*.

See Also **ISnappyHardware, ISnappyPreview**

ISnappyAdviseSync::OnPowerChanged

HRESULT OnPowerChanged([in] BOOL bPower);

bPower TRUE if power is now on, FALSE if off.

Snappy's power turned on or off.

See Also **ISnappyHardware**

ISnappyAdviseSync::OnPreviewImageReady

HRESULT OnPreviewImageReady([in] HGLOBAL hPackedDIB,
[in] ISnappyPreview* pPreview);

hPackedDIB Either an 8-bit DIB or a 24-bit packed DIB depending upon the type of preview. Hot video areas are displayed red.

pPreview Pointer to an **ISnappyPreview** interface. This allows a call to *pPreview->UnlockPreviewImage()* when the *hPackedDIB* is used, so that the memory may be used for future previews.

A preview image is ready to be displayed. Note that a color preview DIB does not have square pixels like most people assume. If you render it assuming square pixels, the image will look very squished. You need to render it into a rectangle with a 4:3 aspect ratio. B&W preview does have square pixels.

Note *pPreview->UnlockPreviewImage()* must be called when the *hPackedDIB* is used and may be made available for continued previewing, or the Previewing will stop.

See Also *ISnappyPreview::SetAdvise, OnHotVideoChanged*

ISnappyAdviseSync::OnReadyToSnap

HRESULT OnReadyToSnap();

Snappy is powered up and ready to capture video.

See Also *ISnappy::SetAdvise*

ISnappyAdviseSync::OnStatusUpdate

HRESULT OnStatusUpdate([in] ESnappyPhase ePhaseNum,
[in] WORD nValue, [in] WORD nTotal);

ePhaseNum Indicates which processing step Snappy is working on.

nValue The current value (use *nValue* / (*nTotal*-1) to calculate percent complete).

nTotal The total number of steps to complete.

Progress indication.

See Also *ESnappyPhase*

ISnappyAdviseSync::OnVideoAvailableChanged

HRESULT OnVideoAvailableChanged([in] BOOL bVideoAvailable);

bVideoAvailable TRUE if video is available, FALSE if not.

The state of the input video has changed.

See Also **ISnappyHardware**

ISnappyAdviseSync::OnVideoThruChanged

HRESULT OnVideoThruChanged([in] BOOL terminated);

bTerminated TRUE if Terminated (not Video Thru), FALSE if Unterminated (Video Thru).

Snappy termination.

See Also **ISnappyHardware**

ISnappyAdviseSync::OnWarning

HRESULT OnWarning([in] ESnappyWarning eWarningNum);

*eWarningNum A*n *ESnappyWarning* number specifying the type of warning (see note below).

Something unexpected happened.

Note Clients should respond to the call with one of the following HRESULT values:
E_ABORT = abort
S_FALSE = retry
S_OK or E_NOTIMPL = continue

See Also *ESnappyWarning*

Interface ISnappyHardware

This is the Snappy hardware status interface.

Function Members

Settings

AdviseSync Interface

SetAdvise	Sets an **AdviseSync** interface to notify when the hardware state changes or, and/or errors.

Termination

IsTerminated	Determines if Snappy is terminating the video source (i.e. Video Thru not being used).
SetTermination	Set whether Video Thru is terminated or not.

Video Mode

IsSVideo	Determines if Snappy is set to *SVideo* mode.
SetSVideo	Set whether *SVideo* mode is used.

Power Mode

IsPowerSaver	Determines if Snappy is set to *PowerSaver* mode.
PowerSaver	Set whether *PowerSaver* mode is used.

Ports

GetLPTPort	Get the current LPT Port.
SetLPTPort	Sets the LPT Port for Snappy to use.
GetNumPorts	Get the number of available parallel ports.

Snap Preparation

PrepareToSnap	Turn Snappy on so that next snap occurs as fast as possible.

ISnappyHardware::GetLPTPort

HRESULT GetLPTPort([out] WORD* pnPortNum);

pnPortNum The LPT port Snappy is currently assigned to.

Get the current LPT port.

Note If no LPT port has been assigned this call returns *E_???*.

See Also *SetLPTPort, GetNumPorts*

ISnappyHardware::GetNumPorts

HRESULT GetNumPorts([out] unsigned long* pnNumPorts);

pnNumPorts The number of available parallel ports.

Get the number of available parallel ports.

Note Always returns *S_OK*.

See Also *GetLPTPort, SetLPTPort*

ISnappyHardware::IsPowerSaver

HRESULT IsPowerSaver([out] BOOL* pbPowerSaver);

pbPowerSaver Pointer to **BOOL**, set TRUE if *PowerSaver* is set, FALSE if not.

Determines if Snappy is set to *PowerSaver* mode.

Note *PowerSaver* mode causes Snappy to turn off after snapping and *preview* mode to stop if left alone.

See Also *PowerSaver*

ISnappyHardware::IsSVideo

HRESULT IsSVideo([out] BOOL* pbSVideo);

pbSVideo Pointer to **BOOL**, set TRUE if *SVideo* is set, FALSE if not.

Determines if Snappy is set to *SVideo* mode.

See Also *SetSVideo*

ISnappyHardware::IsTerminated

HRESULT IsTerminated([out] BOOL* pbTerminated);

pbTerminated Pointer to **BOOL**, set TRUE if terminated, FALSE if not.

Determines if Snappy is terminating the video source (i.e. Video Thru not being used).

See Also *SetTermination*

ISnappyHardware::PowerSaver

HRESULT PowerSaver([in] BOOL bPowerSaver);

bPowerSaver TRUE if *PowerSaver* is to be set, otherwise FALSE.

Set whether *PowerSaver* mode is used.

See Also *IsPowerSaver*

ISnappyHardware::PrepareToSnap

HRESULT PrepareToSnap();

Turn Snappy on so that next snap occurs as quickly as possible.

See Also *ISnappyAdviseSync::OnReadyToSnap*

ISnappyHardware::SetAdvise

HRESULT SetAdvise([in] ISnappyAdviseSync* pNotify);

pNotify Pointer to the interface to call for notification.

Sets an **AdviseSync** interface to notify when the hardware state changes or, and/or errors.

See Also **ISnappyAdviseSync,** *ISnappy::SetAdvise, ISnappyPreview::SetAdvise, ISnappyProcess::SetAdvise*

ISnappyHardware::SetLPTPort

HRESULT SetLPTPort([in] WORD nPortNum);

nPortNum Set the LPT port for Snappy to use.

Sets the LPT port for Snappy to use.

See Also *GetLPTPort, GetNumPorts*

ISnappyHardware::SetSVideo

HRESULT SetSVideo([in] BOOL bSVideo);

bSVideo TRUE if *SVideo* is to be set, otherwise FALSE.

Set whether *SVideo* mode is used.

See Also *IsSVideo*

ISnappyHardware::SetTermination

HRESULT SetTermination([in] BOOL bTerminate);

bTerminate TRUE if termination is to be set FALSE otherwise.

Set whether *Video Thru* is terminated or not.

See Also *IsTerminated*

Interface ISnappyPreview2

This is the interface for creating Snappy previews.

Function Members

SetAdvise	Sets an **AdviseSync** interface to notify when an image is available, and/or errors.
Alloc24BitPreviewDIB	Allocates a packed DIB that can be used for color previewing.
AllocPreviewDIB	Allocates a packed DIB that can be used for black and white previewing.
FreePreviewDIB	Frees a packed DIB allocated with *Alloc24BitPreviewDIB* or *AllocPreviewDIB.*
SetPreviewDIB	Assigns the DIB used by preview.
StartPreviewing	Begin previewing.
Abort	Stops previewing.
UnlockPreviewImage	Allows Preview to continue previewing.

ISnappyPreview2::Abort

HRESULT Abort();

Stops previewing.

See Also *ISnappy::Abort, ISnappyAdviseSync::Abort, ISnappyProcess::Abort*

ISnappyPreview2::Alloc24BitPreviewDIB

HRESULT Alloc24BitPreviewDIB([out] HGLOBAL* phPackedDIB);

phPackedDIB Handle pointer to packed device-independent bitmap.

Allocates a packed DIB that can be used for color previewing. Note that a color preview DIB does not have square pixels like most people assume. If you render it assuming square pixels, the image will look very squished. You need to render it into a rectangle with a 4:3 aspect ratio.

See Also *AllocPreviewDIB, FreePreviewDIB, SetPreviewDIB*

ISnappyPreview2::AllocPreviewDIB

HRESULT AllocPreviewDIB([out] HGLOBAL* phPackedDIB);

phPackedDIB Handle pointer to packed device-independent bitmap.

Allocates a packed DIB that can be used for black and white previewing.

See Also *Alloc24BitPreviewDIB, FreePreviewDIB, SetPreviewDIB*

ISnappyPreview2::FreePreviewDIB

HRESULT FreePreviewDIB([in] HGLOBAL hPackedDIB);

hPackedDIB Handle to packed device-independent bitmap.

Frees a packed DIB allocated with *Alloc24BitPreviewDIB* or *AllocPreviewDIB*.

See Also *Alloc24BitPreviewDIB, AllocPreviewDIB, SetPreviewDIB*

ISnappyPreview2::SetAdvise

HRESULT SetAdvise([in] ISnappyAdviseSync* pNotify);

pNotify Pointer to the interface to call for notification.
Sets an **AdviseSync** interface to be used to notify when an image is available, and/or errors.

See Also **ISnappySetAdvise**, *ISnappyAdviseSync::OnPreviewImageReady, ISnappy::SetAdvise, ISnappyHardware::SetAdvise, ISnappyProcess::SetAdvise*

ISnappyPreview2::SetPreviewDIB

HRESULT SetPreviewDIB([in] HGLOBAL hPackedDIB);

hPackedDIB Handle to packed device-independent bitmap.

Assigns the DIB used by preview. If the DIB was allocated with *Alloc24BitPreviewDIB* the preview will be in color and if it was allocated with *AllocPreviewDIB* the preview will be in black and white.

See Also *Alloc24BitPreviewDIB, AllocPreviewDIB, FreePreviewDIB*

ISnappyPreview2::StartPreviewing

HRESULT StartPreviewing();

Begin previewing.

Note *SetAdvise* must have been called, so you can be notified when images are available.

See Also *Abort*

ISnappyPreview2::UnlockPreviewImage

HRESULT UnlockPreviewImage();

Allows Preview to continue previewing.

Note Call this from an implementation of *OnPreviewImageReady*.

Interface ISnappyProcess

This is the interface for processing Snappy data. An ordinary application would use **ISnappyProcess** to load and save captured Snappy data for processing such as: color, brightness, and contrast.

ISnappyProcess is quite versatile--application programs might use **ISnappyProcess** functions to examine and/or modify sequences of captured images on the fly.

Function Members

Notification

SetAdvise Sets an **AdviseSync** interface to notify when an image is available, and/or errors.

Loading and Saving

Snappy Data

GetSnappyData Returns a copy of the current unprocessed data to memory.

UseSnappyData Loads Snappy data from memory for processing to image.

SaveSnappyData Saves the *unprocessed* Snappy data to a file.

LoadSnappyData Load *unprocessed* Snappy data from a file.

Process Settings

SaveCurrentProcessSettingsAsDefault
Sets the current process settings (*tint*, *sharpness*, *image size*, etc.) as the default.

GetDefaultProcessSettings
Get the default process settings (*tint*, *sharpness*, *image size*, etc.) from the system registry.

Process Snappy Data

ProcessSnappyData Process the Snappy data and produce a DIB.

ProcessSnappyDataLine
Process one line of Snappy data.

Abort Abort processing Snappy data.

CanProcess Checks to see if Snappy data may be processed with the current settings.

DIB

AllocProcessDIB Allocates a packed DIB based on the current process width, height & bit depth (currently only 24-bit depth). This DIB can be used with *SetProcessDIB*.

FreeProcessDIB Frees a packed DIB allocated with *AllocProcessDIB.*

SetProcessDIB Assigns the DIB to process into.

Speed

SetProcessSpeed	Sets speed and quality of process step.

Image Size and Cropping

GetImageSize	Get the final size of the next image to be processed.
SetImageSize	Set the final size of the next image to be processed.
GetSourceCropping	Get the rectangle (in field coordinates) of the source video to be used when processing.
SetSourceCropping	Set the rectangle (in field coordinates) of the source video to use when processing.
GetMaxCropSize	Get the usable video field size.

Image Settings (Setting Range -50 to 50)

GetTint	Get the tint value to be used on next process.
SetTint	Set the tint value to use for next process.
GetRed	Get the Red value to be used on next process.
SetRed	Set the Red value to use for next process.
GetGreen	Get the Green value to be used on next process.
SetGreen	Set the Green value to use for next process.
GetBlue	Get the Blue value to be used on next process.
SetBlue	Set the Blue value to use for next process.
GetSaturation	Gets the amount of saturation (color) to be used on next process.
SetSaturation	Set the Saturation value to use for next process.
GetBrightness	Get the amount of *brightness* to be used on next process.
SetBrightness	Set the *brightness* value to use for next process.
GetContrast	Get the amount of *contrast* to be used on next process.
SetContrast	Set the *contrast* value to use for next process.
GetGamma	Get the amount of *gamma* (picture) to be used on next process.
SetGamma	Set the *gamma* value to use for next process.
GetSharpness	Get the amount of *sharpness* to be used on next process.
SetSharpness	Set the *sharpness* value to use for next process.

ISnappyProcess::Abort

HRESULT Abort();

Abort processing Snappy data.

See Also *ISnappy::Abort, ISnappyAdviseSync::Abort, ISnappyPreview::Abort*

ISnappyProcess::AllocProcessDIB

HRESULT AllocProcessDIB([out] HGLOBAL* phPackedDIB);

phPackedDIB Handle pointer to packed device-independent bitmap.

Allocates a packed DIB based on the current process width, height & bit depth. This DIB can be used with *SetProcessDIB.*

See Also *SetProcessDIB, FreeProcessDIB, ISnappyPreview::AllocPreviewDIB, ISnappyPreview::FreePreviewDIB, ISnappyPreview::SetPreviewDIB*

ISnappyProcess::CanProcess

HRESULT CanProcess([out] BOOL* pbProcessAllowed);

pbProcessAllowed Set to TRUE if process can occur. Determines if Snappy data can be processed with the current settings.

Here is an example of a *See Also* reference that provides descending order of importance with various scope references, which might occur in the discussion of the object **ISnappyPreview,** for the member *ISnappyPreview::SetAdvise*:.

See Also **ISnappySetAdvise**, *ISnappyAdviseSync::OnPreviewImageReady, ISnappy::SetAdvise, ISnappyHardware::SetAdvise, ISnappyProcess::SetAdvise*

ISnappyProcess::FreeProcessDIB

HRESULT FreeProcessDIB([in] HGLOBAL hPackedDIB);

hPackedDIB Handle to packed device-independent bitmap.

Frees a packed DIB allocated with *AllocProcessDIB.*

See Also *AllocProcessDIB, SetProcessDIB, ISnappyPreview::AllocPreviewDIB, ISnappyPreview::FreePreviewDIB, ISnappyPreview::SetPreviewDIB*

ISnappyProcess::GetBlue

HRESULT GetBlue([out] long* pnBlue);

pnBlue Pointer to long to use for *blue.*

Get the *blue* value to be used on next process.

See Also *SetBlue*

ISnappyProcess::GetBrightness

HRESULT GetBrightness([out] long* pnBrightness);

pnBrightness Pointer to long to use for *brightness.*

Get the amount of *brightness* to be used on next process.

See Also *SetBrightness*

ISnappyProcess::GetContrast

HRESULT GetContrast([out] long* pnContrast);

pnContrast Pointer to long to use for *contrast*.

Get the amount of *contrast* to be used on next process.

See Also *SetContrast*

ISnappyProcess::GetDefaultProcessSettings

HRESULT GetDefaultProcessSettings();

Get the default process settings (*tint, sharpness, image size,* etc.) from the system registry.

See Also *SaveCurrentProcessSettingsAsDefault*

ISnappyProcess::GetGamma

HRESULT GetGamma([out] long* pnGamma);

pnGamma Pointer to long to use for *gamma.*

Get the amount of *gamma* (picture) to be used on next process.

See Also *SetGamma,* **Snappy User's Manual, Picture Adjust**

ISnappyProcess::GetGreen

HRESULT GetGreen([out] long* pnGreen);

pnGreen Pointer to long to use for *green*.

Get the *green* value to be used on next process.

See Also *SetGreen*

ISnappyProcess::GetImageSize

HRESULT GetImageSize([out] WORD* pnWidth,
[out] WORD* pnHeight,
[out] WORD* pnBitDepth);

pnWidth Pointer to word to use for *pixel width*.
pnHeight Pointer to word to use for *pixel height*.
pnBitDepth Pointer to word to use for *pixel depth* (i.e. 24 for true color). Currently only supports 24-bit depth.

Get the final size of the next image to be processed.

See Also *SetImageSize, GetSourceCropping, GetMaxCropSize*

ISnappyProcess::GetMaxCropSize

HRESULT GetMaxCropSize([out] WORD* pnWidth,
[out] WORD* pnHeight);

pnWidth Pointer to word to use for *width.*
pnHeight Pointer to word to use for *height.*

Get the usable video field size.

See Also *GetSourceCropping, SetSourceCropping, GetImageSize*

ISnappyProcess::GetRed

HRESULT GetRed([out] long* pnRed);

pnRed Pointer to long to use for *red*.

Get the *red* value to be used on next process.

See Also *SetRed*

ISnappyProcess::GetSaturation

HRESULT GetSaturation([out] long* pnSaturation);

pnSaturation Pointer to long to use for *saturation.*

Gets the amount of *saturation* (color) to be used on next process.

See Also *SetSaturation*

ISnappyProcess::GetSharpness

HRESULT GetSharpness([out] long* pnSharpness);

pnSharpness Pointer to long to use for *sharpness.*

Get the amount of *sharpness* to be used on next process.

See Also *SetSharpness*

ISnappyProcess::GetSnappyData

HRESULT GetSnappyData([in] [out] BYTE** ppData);

ppData Pointer to a BYTE* pointer to be set to point to the data.

Returns a copy of the current unprocessed data.

See Also *ISnappyAdviseSync::OnDataReady, ISnappy::SnapToUnprocessedData, UseSnappyData, SaveSnappyData, LoadSnappyData*

ISnappyProcess::GetSourceCropping

HRESULT GetSourceCropping([out] WORD* pnLeft,
[out] WORD* pnTop,
[out] WORD* pnRight,
[out] WORD* pnBottom);

pnLeft Pointer to word to use for *left*.
pnTop Pointer to word to use for *top*.
pnRight Pointer to word to use for *right*.
pnBottom Pointer to word to use for *bottom*.

Get the offsets from the maximum field size (in field coordinates) of the source video to be used when processing.

See Also *SetSourceCropping, GetMaxCropSize*

ISnappyProcess::GetTint

HRESULT GetTint([out] long* pnTint);

pnTint Pointer to long to use for *tint.*

Get the *tint* value to be used on next process.

See Also *SetTint*

ISnappyProcess::LoadSnappyData

HRESULT LoadSnappyData([in] LPCTSTR pFileName);

pFileName Pointer to name of file to load.

Load unprocessed Snappy data from a file.

See Also *UseSnappyData, GetSnappyData, SaveSnappyData*

ISnappyProcess::ProcessSnappyData

HRESULT ProcessSnappyData();

ProcessSnappyData immediately returns control to the caller and begins processing video data. When the process finishes, the caller is notified.

Note *SetAdvise* must have been called, so you can be notified when images are available through *ISnappyAdviseSync::OnDataReady.*

ISnappyAdviseSync::OnLineProcessed will notify after each line is processed.

See Also *SetAdvise, ISnappyAdviseSync::OnDataReady, ISnappyAdviseSync::OnLineProcessed, ProcessSnappyDataLine, SetProcessSpeed, AllocProcessDIB, FreeProcessDIB, SetProcessDIB*

ISnappyProcess::ProcessSnappyDataLine

HRESULT ProcessSnappyDataLine([in] WORD nLine, [in] BYTE* pBits);

pBits Pointer to line data to be filled out.
nLine The image line to be processed.

Immediately process one line of Snappy data, and then return to the caller.

Note This routine processes only one line. To process an entire image, use *ProcessSnappyData* instead.

See Also *ProcessSnappyData, SetProcessSpeed, AllocProcessDIB, FreeProcessDIB, SetProcessDIB*

ISnappyProcess::SaveCurrentProcessSettingsAsDefault

HRESULT SaveCurrentProcessSettingsAsDefault();

Sets the current process settings (*tint*, *sharpness*, *image size*, etc.) as the default.

See Also *GetDefaultProcessSettings*

ISnappyProcess::SaveSnappyData

HRESULT SaveSnappyData([in] LPCTSTR pFileName);

pFileName Pointer to name of file to create.

Saves the unprocessed Snappy data.

See Also *LoadSnappyData, GetSnappyData, UseSnappyData*

ISnappyProcess::SetAdvise

HRESULT SetAdvise([in] ISnappyAdviseSync* pNotify);

pNotify Pointer to the interface to call for notification.

Sets an **AdviseSync** interface to be used to notify when an image is available, and/or errors.

See Also **ISnappyAdviseSync**, *ISnappy::SetAdvise, ISnappyPreview::SetAdvise, ISnappyHardware::SetAdvise*

ISnappyProcess::SetBlue

HRESULT SetBlue([in] long nBlue);

nBlue The *blue* value.

Set the *blue* value to use for next process.

See Also *GetBlue*

ISnappyProcess::SetBrightness

HRESULT SetBrightness([in] long nBrightness);

nBrightness The *brightness* value.

Set the *brightness* value to use for next process.

See Also *GetBrightness*

ISnappyProcess::SetContrast

HRESULT SetContrast([in] long nContrast);

nContrast The *contrast* value.

Set the *contrast* value to use for next process.

See Also *GetContrast*

ISnappyProcess::SetGamma

HRESULT SetGamma([in] long nGamma);

nGamma The *gamma* value

Set the *gamma* value to use for next process.

See Also *GetGamma*

ISnappyProcess::SetGreen

HRESULT SetGreen([in] long nGreen);

nGreen The *green* value.

Set the *green* value to use for next process.

See Also *GetGreen*

ISnappyProcess::SetImageSize

HRESULT SetImageSize([in] WORD nWidth,
[in] WORD nHeight,
[in] WORD nBitDepth);

nWidth Pixel *width*.
nHeight Pixel *height*.
nBitDepth Pixel *depth* (i.e. 24 for true color). Currently only 24-bit depth is supported.

Set the final size of the next image to be processed.

See Also *GetImageSize*

ISnappyProcess::SetProcessDIB

HRESULT SetProcessDIB([in] HGLOBAL hPackedDIB);

hPackedDIB Handle to packed device-independent bitmap.

Assigns the DIB to process into.

See Also *AllocProcessDIB, FreeProcessDIB*

ISnappyProcess::SetProcessSpeed

HRESULT SetProcessSpeed([in] BOOL bSpeed);

bSpeed This parameter was originally defined as a boolean but it actually recognizes three values. A value of 1 indicates fast but lower-quality processing while 0 indicates normal processing and 2 indicates HiDefinition mode (very slow).

Sets speed and quality of process step. Fast uses a simpler algorithm for higher speed, however, the image quality is reduced.

ISnappyProcess::SetRed

HRESULT SetRed([in] long nRed);

nRed The *red* value.

Set the *red* value to use for next process.

See Also *GetRed*

ISnappyProcess::SetSaturation

HRESULT SetSaturation([in] long nSaturation);

nSaturation The *saturation* value.

Set the *saturation* value to use for next process.

See Also *GetSaturation*

ISnappyProcess::SetSharpness

HRESULT SetSharpness([in] long nSharpness);

nSharpness The *sharpness* value.

Set the *sharpness* value to use for next process.

See Also *GetSharpness*

ISnappyProcess::SetSourceCropping

HRESULT SetSourceCropping([in] WORD nLeft,
[in] WORD nTop,
[in] WORD nRight,
[in] WORD nBottom);

nLeft Left.
nTop Top.
nRight Right.
nBottom Bottom.

Get the offsets from the maximum field size (in field coordinates) of the source video to be used when processing.

See Also *GetSourceCropping*, *GetMaxCropSize*

ISnappyProcess::SetTint

HRESULT SetTint([in] long nTint);

nTint The *tint* value.

Set the *tint* value to use for next process.

See Also *GetTint*

ISnappyProcess::UseSnappyData

HRESULT UseSnappyData([in] BYTE *pData);

pData Pointer to data loaded by this call.

Loads Snappy data for processing to image.

See Also *GetSnappyData*, *SaveSnappyData*, *LoadSnappyData*