Camera API

Version 2.0.0.0

User Manual (3rd Edition)

Document Revision : 2.0

1stVision Inc.

www.1stVision.com

Tel : +978-474-0044

Contents

1. License

1.1 Evaluation Version Limit & License

The Evaluation Version without 1stVision Inc. Camera API (ImCam Library) ‘s license can be used only for 1stVision Inc. Cameras. You can use it for the first one(1) hour and then this API is automatically closed and you cannot use all the functions of API.

You use Evaluation Version for a certain period and the expiry(expiration) date will be announced at the time of its distribution.(But when you begin the application again, it can be used for the additional one hour)

You shall refer to the standard EULA Document regarding other License regulations.

When you have any question to the formal Version purchase, please do not hesitate to call our company.

1.2 Note:

1stVision Inc. Camera API(ImCam Library) only supports 1stVision Inc. hardware and you are not allowed to use this Camera API to build Application for other camera hardware. The EVALUATION VERSION SOFTWARE is provided to you “AS IS” without warranty. The entire risk of the quality and performance of the software is with you. We appreciate any feedback and bug report, however, we can not guarantee satisfactory response.

1.3 Legal Notice

By installing, copying or otherwise using the SOFTWARE, you agree to be bound by the terms of the **End User License Agreements (EULA)**. The SOFTWARE includes 1stVision Inc. and 1stVision Inc. suppliers’ intellectual property. Please read 1stVision Inc. and 1stVision Inc. suppliers’ EULA before installing the SOFTWARE. If you do not accept the terms of the license agreements, please do not install, copy or use the SOFTWARE.

2. System & Software Requirement

2.1 SYSTEM

1stVision Inc. Camera API (ImCam Library) is the library functions to control the cameras produced by 1stVision Inc. , and the performance of CPU is the most important to process all the real-time data transmitted by the camera.

The API operates in the CPU with PI III-1.0 up without a hitch, however, the actual number of frame can be reduced under it.

2.2 SOFTWARE

The following software are required to use 1stVision Inc. Camera API;

- 1stVision Inc. IEEE-1394 Camera Device Driver
- Compiler : Microsoft Visual C ++ 6.0, C++Builder 6.0 or Visual Basic 6.0
- DirectX 8.1 or higher(in Windows2000, Windows98), DirectX 9.0 is recommended in Windows XP
- Windows Media Encoder 9 series(http://www.microsoft.com/windows/windowsmedia)
  (It is necessary for saving WMV type stream capture. Please refer to library reference)

3. Setup

Installing library and setting up the project to use 1stVision Inc. camera API (ImCam API).

3.1 Construction of API File

- 3 files are in "Lib" directory.

ImCamDef.h	: structure definition file
ImCamApi.h	: API Header file
ImCam.lib	: API Library file (for Visual C++ users)

- 2 files are in "Builder" directory

ApiImport.h	: API import header file (for Builder or Visual C++ users)
ApiImport.cpp	: API import header file (for Builder or Visual C++ users)

- 1 file is in "VBasic" directory

ImCamApi.bas	: API import module file (for Visual Basic users)

- 2 files is in "Bin" directory

ImCam.dll	: API runtime dll file
LibSample.exe	: Sample project execution file.

- Sample project is in "LibSample" directory.

Sample code	: Example application source using the "ImCam" API. We provide sample code for Visual C++ only.

3.2 Setup

Copy "Lib" directory including the provided library files to a new project directory, or adds up the library necessary for those library and Direct Show(Refer to 3.3, Setting of Project)..

3.3 Visual Studio Environment

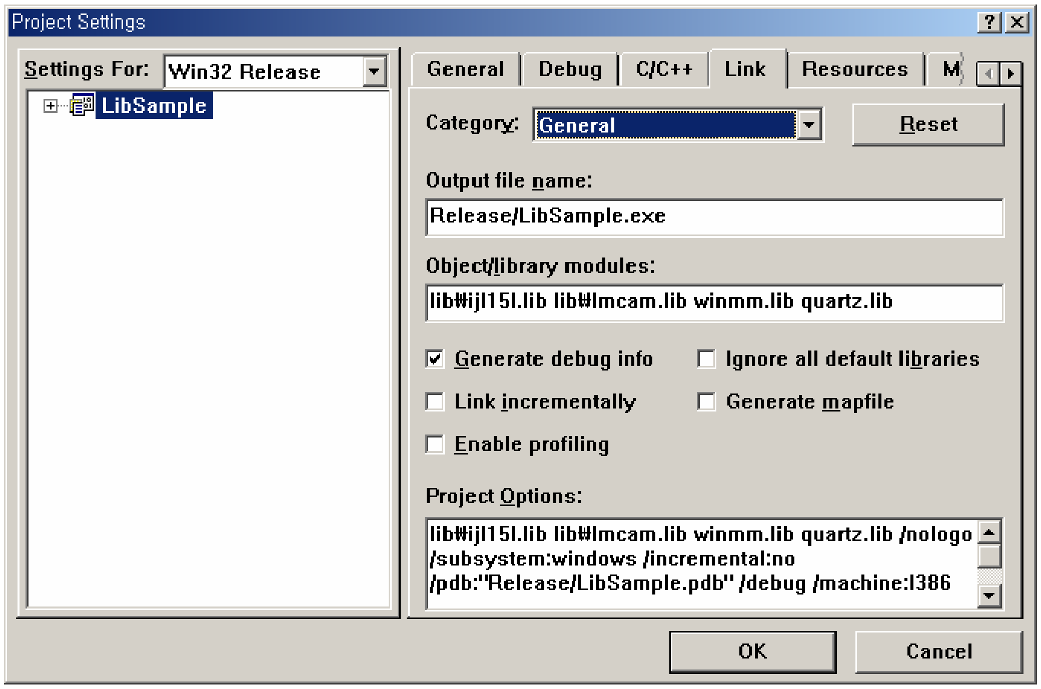
Please refer to "LibSample" project regarding actual set-up.

Adds up the next libraries in "Link" tab of "Project Setting" dialog. This is necessary if user want to use "ImCam.lib" file while compilling the program. If user want to use "ApiImport.h" and "ApiImport.cpp", just insert these source into user project without linking with the ImCam.lib file.

lib\ImCam.lib

winmm.lib

quartz.lib



Category: General

Output file name:
Release/LibSample.exe

Object/library modules:
lib₩ijl15l.lib lib₩Imcam.lib winmm.lib quartz.lib

[✓] Generate debug info
[ ] Ignore all default libraries
[ ] Link incrementally
[ ] Generate mapfile
[ ] Enable profiling

Project Options:

```
lib₩ijl15l.lib lib₩Imcam.lib winmm.lib quartz.lib /nologo
/subsystem:windows /incremental:no
/pdb:"Release/LibSample.pdb" /debug /machine:I386
```

3.4 C++ Builder Environment

For C++ Builder, user can simply add "ApiImport.cpp" and "ApiImport" source to the project and build program. But in the program, the "LoadImCamAPI" function should be called before using other API functions. The "LoadImcamAPI" load the runtime library and get function pointer for each library function. And the "UnloadImCamApi" function should be called before the application terminates. Please refer to the source code for details.

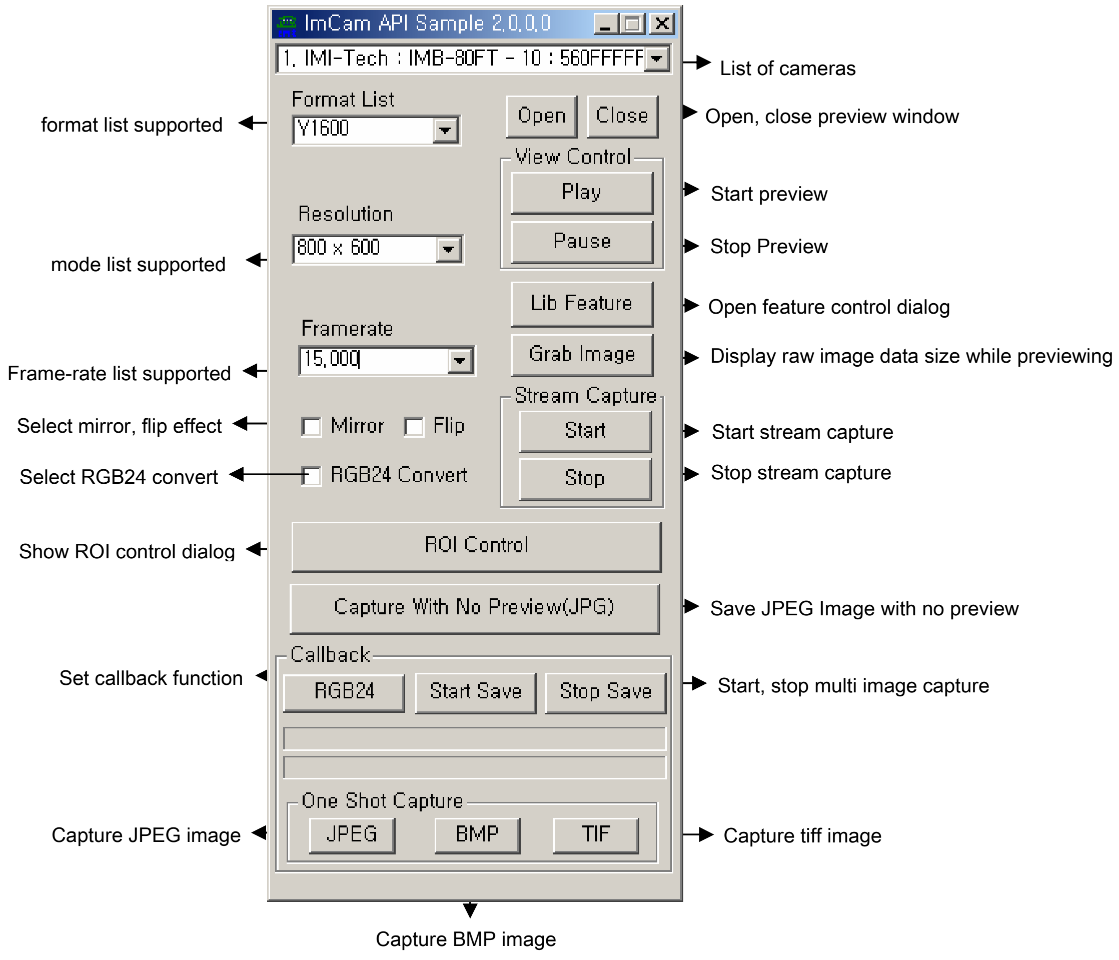
3.5 Visual Basic Environment

For Visual Basic, user can imply add the "ImCamApi.bas" module file to the project and build the application. This module also fines all the functions included in the ImCamAPI library. Please refer to the source code for details.

4. ImCam – 1stVision Inc. Camera API Sample Application

LibSample, as a sample project, shows the example of how to use the individual functions in 1stVision Inc. Camera API, and provides the function of saving the captured image into the Jpeg, Bmp and Tiff file as well as the basic functions (the capturing functions of JPEG, BMP, and TIF are included in the API)

(Please refer to the sample source regarding the executing of the function each.)



1. IMI-Tech : IMB-80FT - 10 : 560FFFFF

Format List: Y1600

Resolution: 800 x 600

Framerate: 15.000

5. Summary of Functions

The functions which have BOOL type as return value in the API, will return IMC_SUCESS if it succeeds and return IMC_FAIL or minus value if it fails. IMC_SUCCESS and IMC_FAIL is defined in “ImCamDef.h” as 1 and 0 respectively.

Category	Function	Description
Initialize Functions	ImCamInit	Initialize API and returns the number of camera connected to the system.
	ImCamUninit	Terminate the use of API.
Information Functions	ImCamSelectCam	Select one camera among the cameras connected to the system.
	ImCamGetName	Read the name of the camera including company name, model name and serial number.
	ImCamGetModelName	Read the model name of the camera.
	ImCamGetFirmwareVersion	Read the H/W version of the camera.
Format, Mode, Frame rate	ImCamGetFormatList	Get the format list the camera supports.
	ImCamSetVideoFormat	Set the camera to the designated format.
	ImCamGetVideoFormat	Get the format currently set from the camera.
	ImCamGetModeList	Get the mode list the camera supports for the format defined.
	ImCamSetVideoMode	Set the camera to the designated mode.
	ImCamGetVideoMode	Get the mode currently set from the camera.
	ImCamGetFrameRateList	Get the frame rate list the camera support for the format and mode defined.
	ImCamSetFrameRate	Set the camera to the designated frame rate.
	ImCamGetFrameRate	Get the frame rate currently set from the camera.
Display Control	ImCamOpen	Prepare the API for displaying the camera.
	ImCamClose	Resolve the API for stopping the camera display.
	ImCamIsOpen	Check the state of the camera.
	ImCamStart	Set the camera to start data out.
	ImCamStop	Set the camera to stop data out.
Feature Control	ImCamIsFeatureSupport	Check if the feature is supported by the camera.
	ImCamIsFeatureAuto	Check the auto state of the feature.
	ImCamSetFeatureAuto	Set the feature to auto/manual state.
	ImCamIsOnePushSupport	Check if the one-push is supported by the feature.

	ImCamSetOnePush	Set the feature to one-push mode.
	ImCamGetFeatureRange	Get the valid range of the feature.
	ImCamGetFeatureValue	Get the current value of the feature.
	ImCamSetFeatureValue	Set the feature to the designated value.
	ImCamSetFeatureDefault	Set the feature to the factory default state.
	ImCamSetAllFeatureDefault	Set all features to the factory default state.
	ImCamGetDataBits	Get the valid data bits.
	ImCamSetAWBRegion	Set the AWB region for white balancing.
	ImCamSetDefaultAWBRegion	Set the AWB region to the factory default state.
Capture Functions	ImCamGetImageSize	Get the image size of the current mode.
	ImCamGetBufSize	Get the data buffer size of the current mode.
	ImCamSetCallback	Register callback function.
	ImCamSaveBMP	Save frame data to BMP image.
	ImCamSaveTIF	Save frame data to TIF image.
	ImCamSaveJPG	Save frame data to JPG image.
	ImCamStartAVICapture	Start AVI stream capture.
	ImCamStopAVICapture	Stop AVI stream capture.
	ImCamStartWMVCapture	Start WMV stream capture.
	ImCamStopWMVCapture	Stop WMV stream capture.
ROI Functions	ImCamIsROISupport	Check if ROI mode is supported by the camera.
	ImCamEnableROIMode	Activate ROI mode.
	ImCamDisableROIMode	Inactivate ROI mode.
	ImCamGetROISize	Get the ROI region size currently set.
	ImCamGetMaxArea	Get the maximum ROI region size.
	ImCamGetCurrentArea	Get the ROI information currently set.
	ImCamSetCurrentArea	Set the ROI information.
Image Effect Functions	ImCamSetMirror	Set the Mirror effect.
	ImCamGetMirror	Get the state of the Mirror effect.
	ImCamSetFlip	Set the Flip effect.
	ImCamGetFlip	Get the state of the Flip effect.
	ImCamSetNegative	Set the Negative effect.
	ImCamGetNegative	Get the state of the Negative effect.
Trigger Functions	ImCamGetTrigMode	Get the Trigger mode state..
	ImCamSetTrigMode	Set the Trigger mode.
Read, Write functions	ImCamReadQuadlet	Read 32bit data form the 1394 register.

	ImCamReadBlock	Read multiple of 32bit data from the 1394 register.
	ImCamWriteQuadlet	Write data to the 1394 register.

6. API Function Details

6.1 Camera and API initialization Functions

API should be initialized before calling any other API function. If API function is called while is not initialized, the function will always return -1.

After completing the use of the API, user should call the ImCamUninit function before closing the user application.

6.1.1 ImCamInit

DEFINITION
BOOL ImCamInit();
PARAMETERS
None
RETURN VALUE
Number of the camera API detects.
REMARKS
This function initializes the API and returns the number of the camera detects. So, this function should be called before any other API function. This function will return -1 if it can not initialize the API.

6.1.2 ImCamUninit

DEFINITION
VOID ImCamUninit();
PARAMETERS
None
RETURN VALUE
REMARKS
After completing the use of the API, user should call this function before terminating the user application to inform the API to release information contained.

6.2 Camera Information Functions

After initializing the API, user should select the camera first to control. This is important for synchronization with the API and user should call this function once when the camera to control is changed.

For the 1394 camera, all cameras should have regions defined in IEEE1394-1995, IEEE1394-2000, IEEE1212 and IIDC specification to save information about the camera maker, camera model and other characteristics. User can read this area by calling the ImCamGet(Model)Name function or functions explained in section 6.10.

6.2.1 ImCamSelectCam

DEFINITION
BOOL ImCamSelectCam(IN INT index);
PARAMETERS
index : Index of the camera installed in the system. It begins with 0 and should be in the range of the number return by the ImCamInit function.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
Selects the camera corresponding to the index among the cameras installed.

6.2.2 ImCamGetName

DEFINITION
BOOL ImCamGetName(IN INT nCamIndex, OUT INT *pnSize, OUT CHAR *pstrName)
PARAMETERS
nCamIndex : camera index to get information. **pnSize :** length of the valid string “strName”. **strName :** name of the camera
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
The company, model and GUID is included in the name and is expressed as “company – model : GUID”.

6.2.3. ImCamGetModelName

DEFINITION
BOOL ImCamGetModelName(IN INT nCamIndex, OUT INT *pnSize, OUT CHAR *pstrName)
PARAMETERS
nCamIndex : index of the camera to get information **pnSize :** length of the valid string in the strName **strName :** model name of the camera
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function returns only the camera model name.

6.2.4 ImCamGetFirmwareVersion

DEFINITION
BOOL ImCamGetFirmwareVersion(IN INT nCamIndex, CHAR *pstrVersion);
PARAMETERS
nCamIndex : index of the camera to get information **pstrVersion** : version information
RETURN VALUE
IMC_SUCCESS : Success Else : Ffail
REMARKS
This function can be used to check the H/W version of the camera made by 1stVision Inc.. The H/W version is expressed as “X.XXX, Y.YYY” where “X.XXX” means micom version and “Y.YYY” means FPGA version.

6.3 Video Format, Mode, Frame rate Functions

The image streamed out from the camera is decided by the image format, image size and frame rate.

The API classifies the stream information with 3 categories – format, mode, frame rate. The format means data format camera outs and the mode means image size for the designated format. The frame rate will vary to the designated format and mode. So, if user changes format or mode for the camera, user should recompose the mode and frame rate.

For example, consider the situation where a camera has 2 formats, 4 modes and frame rates like as follows. To decide what format should be out from the camera, user should select the format first. And user selects one mode supported by the format selected and selects frame rate supported by the mode selected sequentially.

IMC_FORMAT_Y800	IMC_MODE_640x480	
	IMC_MODE_1024x768	IMC_FRATE_3_75
		IMC_FRATE_7_5
		IMC_FRATE_15
IMC_FORMAT_Y422	IMC_MODE_640x480	
	IMC_MODE_1024x768	

6.3.1 ImCamGetFormatList

DEFINITION

BOOL ImCamGetFormatList(IN INT nCamIndex, IN OUT INT *pnListSize, OUT INT pFormat[])

PARAMETERS

nCamIndex : index of the camera to get the format list supported.

nListSize : number of the valid format in the list. This value should be set to the count of the pFormat array when calling this function.

pFormat : format list supported by the selected camera.

RETURN VALUE

IMC_SUCCESS : Success

Else : Fail

REMARKS

This function is used to get the list of the format supported by the camera. The pnListSize value should be 6 at least when call this function. If this value is smaller than the format supported by the camera, this function will return error.

The relation between the format value provided by the API and the real camera data format is like as follows, and these values are defined in the “ImCamDef.h” header file.

	IMC_VIDEO_FORMAT
RGB 24bit	IMC_FORMAT_RGB24
Y 800	IMC_FORMAT_Y800
Y 1600	IMC_FORMAT_Y1600
YUV 4:1:1	IMC_FORMAT_Y411
YUV 4:2:2	IMC_FORMAT_Y422
YUV 4:4:4	IMC_FORMAT_Y444
	IMC_FORMAT_UNKNOWN

* In case mono where more than 8 bits, only 8 bits is used for display.

6.3.2 ImCamSetVideoFormat

DEFINITION
BOOL ImCamSetVideoFormat(IN INT nCamIndex, IN INT format)
PARAMETERS
nCamIndex : index of the camera to set video format. **Format :** the format value to set. This argument should have one of the value get by the ImCamGetFormatList function.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to set the camera to the designated format.

6.3.3 ImCamGetVideoFormat

DEFINITION
BOOL ImCamGetVideoFormat(IN INT nCamIndex, OUT INT *pFormat);
PARAMETERS
nCamIndex : index of the camera to get format information **pFormat :** the format value currently set.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.3.4 ImCamGetModeList

DEFINITION

```
BOOL ImCamGetModeList(IN INT nCamIndex, IN INT Format,
                      IN OUT INT *pnListNum, OUT INT pMode[]);
```

PARAMETERS

nCamIndex : index of the camera to get mode list.

Format : the format to get mode list

pnListNum : size of the list of supported mode for the format. This value should be set to the count of the pMode array when calling this function.

pMode : list of modes supported for the format.

RETURN VALUE

true : Success

false : Fail

REMARKS

The mode expresses image size and has the followings. These values are defined in the “ImCamDef.h” header file.

IMC_MODE_160x120
IMC_MODE_320x240
IMC_MODE_640x480
IMC_MODE_800x600
IMC_MODE_1024x768
IMC_MODE_1280x960
IMC_MODE_1280x1024
IMC_MODE_1360x1032
IMC_MODE_1600x1200
IMC_MODE_VARIABLE_MODE0~MODE7
IMC_MODE_UNKNOWN

6.3.5 ImCamSetVideoMode

DEFINITION
BOOL ImCamSetVideoMode(IN INT nCamIndex, IN INT Mode)
PARAMETERS
nCamIndex : index of the camera to set mode **Mode :** mode to set
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to set the camera as mode user select. The ImCamGetModeList function should be called before calling this function to get the modes supported for the format.

6.3.6 ImCamGetVideoMode

DEFINITION
BOOL ImCamGetVideoMode(IN INT nCamIndex, OUT INT *pMode)
PARAMETERS
nCamIndex : index of the camera to get mode. **pMode :** mode value currently set in the camera.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.3.7 ImCamGetFramerateList

DEFINITION

BOOL ImCamGetFramerateList(IN INT nCamIndex, IN INT Format, IN INT Mode,
IN OUT INT *pnListNum, OUT INT pRate[])

PARAMETERS

nCamIndex : index of the camera to get frame rate information.

Format : format value currently set in the camera.

Mode : mode value currently set in the camera.

pnListNum : the size of the valid frame rate list. This value should be set to the count of the pRate array when calling this function.

pRate : list of the frame rate for the format and mode.

RETURN VALUE

IMC_SUCCESS : Success

Else : Fail

REMARKS

This function is used to get the frame rate supported for the format and mode. The ImCamSetVideoMode function should be called before this function.

The relation between the values retained in the frame rate list and real frame is like as follows and these values are defined in the "ImCamDef.h" header file.

1.875 fps	IMC_FRATE_1_875
3.250 fps	IMC_FRATE_3_75
7.500 fps	IMC_FRATE_7_5
15.000 fps	IMC_FRATE_15
30.000 fps	IMC_FRATE_30
60.000 fps	IMC_FRATE_60
120.000 fps	IMC_FRATE_120
240.000 fps	IMC_FRATE_240
	IMC_FRATE_UNKNOWN

6.3.8 ImCamSetFrameRate

DEFINITION
BOOL SetFrameRate(IN INT nCamIndex, IN INT Frate)
PARAMETERS
nCamIndex : index of the camera to set frame rate. **Frate :** frame rate value to set.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to set the frame rate to use.

6.3.9 ImCamGetFrameRate

DEFINITION
BOOL ImCamGetFrameRate(IN INT nCamIndex, IMC_VIDEO_FRATE *pFrate)
PARAMETERS
nCamIndex : index of the camera to get frame rate. **pFrate :** frame rate value currently set in the camera.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

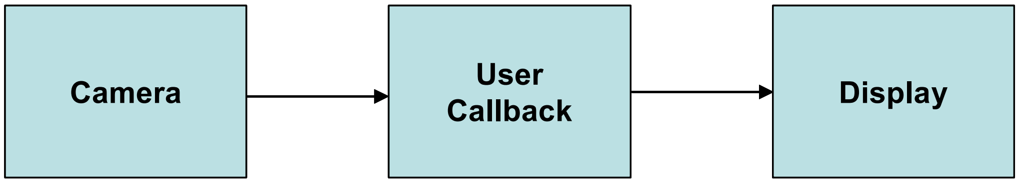
6.4 Display Functions

After setting the format, mode and frame rate, user can get the data from the camera. The API uses DirectX and so, to get the camera, we should create DirectX graph first and set the camera to output the data.

While getting the camera, user can use preview or not and user can get every frame data for his own use.

Following figure describes the data flow for display.

To set the camera to stream out, user should call ImCamOpen and ImCamStart function sequentially and to stop the stream out, user should call ImCamStop and ImCamClose function sequentially.



The API user can set data format passed to the callback function regardless of the image format streamed out by the camera by setting the argument of the ImCamOpen function.

The API calls user callback function if user has registered his own callback function with every frame data with the format user defined.

6.4.1 ImCamOpen

DEFINITION

```
BOOL ImCamOpen(IN INT nCamIndex, IN BOOL bUseCallBack, IN BOOL bUsePreview,
               IN HWND hPreviewWnd, IN ULONG ulResizeView, IN BOOL bFullView,
               IN INT grabMode);
```

PARAMETERS

nCamIndex : index of the camera to open.

bUseCallback : flag to specify whether to use callback function or not.

bUsePreview : flag to specify whether to use preview window or not.

hPreviewWnd : handle of the user window used for display.

ulResizeView : method of the preview window layout.

bFullView : flag to specify whether to user full window mode or not when the bUsePreview is TRUE.

grabMode: image data format passed to callback function.

RETURN VALUE

IMC_SUCCESS : Success

Else : Fail

REMARKS

If the bUsePreview is TRUE and hPreviewWnd is NULL, the API will use DirectX preview window for display and If hPreviewWnd is not NULL, the API use user window to display. The ulResizeView can be set as one of the IMC_VIEW_RESIZE, IMC_VIEW_STATIC, IMC_VIEW_FILL and has meaning only if the hPreviewWnd is not NULL. This argument defines the layout of the camera stream data in the display.

If the ulResizeView is set to IMC_VIEW_RESIZE, the API will resize the preview window to the size of the real stream data. If the value is set to IMC_VIEW_STATIC, the API will not resize the preview and display the real stream data in the preview window regardless of the size. If the value is set to IMC_VIEW_FILL, the API will expand or reduce the real stream data to the size of preview window and it results in display image which is not clean if the preview window is set to more larger than the real stream data size. The default value of this argument is IMC_VIEW_FILL.

If user wants to get every frame data, user should register by calling ImCamSetCallback before this function. Please refer to ImCamSetCallback section.

The grabMode argument defines the format passed to the user callback function and can be set as one the following values defined in "ImCamDef.h" header file.

```
IMC_GRAB_RAW,           // camera original data format
IMC_GRAB_RGB888,        // RGB24
IMC_GRAB_RGB555,        // RGB16
IMC_GRAB_RGB565,        // RGB16
```

This function doesn't set the camera to stream out data, so user should call ImCamStart function to set the camera to stream out.

6.4.2 ImCamClose

DEFINITION
BOOL ImCamClose(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to close.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function stops preview and closes all interfaces opened by the ImCamOpen function. This function should be called after ImCamStop function.

6.4.3 ImCamIsOpen

DEFINITION
BOOL ImCamIsOpen(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to check open state.
RETURN VALUE
IMC_SUCCESS : camera is opened. Else : camera is not opened.
REMARKS

6.4.4 ImCamStart

DEFINITION
BOOL ImCamStart(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to stream out data.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function controls the camera to stream out data. If the preview window is set, the API use the preview window for display so the user can live image in the preview window.

6.4.5 ImCamStop

DEFINITION
BOOL ImCamStop(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to stop stream data
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function controls camera to stop stream data. This function doesn't close the preview window, so user should call ImCamClose to close preview window. If user calls this function followed by ImCamStart, user can live stream again in the preview window.

6.5 Feature Control Functions

The feature supported by the camera is varies to the camera. So, the feature has to be verified whether the camera support or not before use the feature.

The API, now, handle amount to 17 features defined in the “ImCamDef.h” header file and the supported feature by the API can be expanded.

The user must know the support of the feature, support of the auto mode, support of the one-push and valid range of the feature value before using the feature.

Because the API gets feature information from the camera when the ImCamIsFeatureSupport is called, this function should be called before any other feature control functions.

6.5.1 ImCamIsFeatureSupport

DEFINITION
BOOL ImCamIsFeatureSupport(IN INT nCamIndex, IN INT feature)
PARAMETERS
nCamIndex : index of the camera to verify feature. **feature :** feature id to verify.
RETURN VALUE
IMC_SUCCESS : camera support this feature. Else : camera doesn't support this feature.
REMARKS
According to the camera, supported feature, the property of the feature(range, auto and one-push support) can vary, so user first call this function with all the feature define in the "ImCamDef.h". Before this function is called, the API doesn't get the feature information from the camera.

6.5.2 ImCamIsFeatureAuto

DEFINITION
BOOL ImCamIsFeatureAuto(IN INT nCamIndex, IN INT feature)
PARAMETERS
nCamIndex : index of the camera to verify feature. **feature** : feature id to check whether the auto mode is supported or not.
RETURN VALUE
-1 : feature doesn’t support auto mode 1 : feature supports auto mode and currently set to auto mode 0 : feature supports auto mode and currently set to manual mode.
REMARKS
This function is used to check the auto mode supported or not for the feature.

6.5.3 ImCamIsOnePushSupport

DEFINITION
BOOL ImCamIsOnePushSupport(IN INT nCamIndex, IN INT feature);
PARAMETERS
nCamIndex : index of the camera to verify the feature. **feature** : feature id to check whether the one-push is supported or not.
RETURN VALUE
IMC_SUCCESS : feature supports one-push mode. Else : feature doesn't support one-push mode.
REMARKS

6.5.4 ImCamSetFeatureAuto

DEFINITION
BOOL ImCamSetFeatureAuto(IN INT nCamIndex, IN INT feature, IN BOOL bAuto)
PARAMETERS
nCamIndex : index of the camera to set the feature. **feature :** feature id to set to auto or manual mode. **bAuto :** flag for auto(TRUE) or manual(FALSE)
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.5.5 ImCamSetOnePush

DEFINITION
BOOL SetOnePush(IN INT CamIndex, IN INT feature, IN BOOL bOnePush);
PARAMETERS
nCamIndex : index of the camera to set the feature. **feature** : feature id to set one-push mode. **bOnePush** : flag for one-push mode.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.5.6 ImCamGetFeatureRange

DEFINITION
BOOL ImCamGetFeatureRange(IN INT nCamIndex, IN INT feature, OUT INT *pMin, OUT INT *pMax);
PARAMETERS
nCamIndex : index of the camera to get feature information. **feature :** feature id to get the range. **pMin :** minimum value of the feature. **pMax :** maximum value of the feature.
RETURN VALUE
IMC_SUCCESS : Success Else : FAIL
REMARKS

6.5.6 ImCamGetFeatureValue

DEFINITION
BOOL ImCamGetFeatureValue(IN INT nCamIndex, IN INT feature, OUT INT *pValue);
PARAMETERS
nCamIndex : index of the camera to get feature value. **feature :** feature id to get information. **pValue :** feature value currently set.
RETURN VALUE
IMC_SUCCESS : Success Minus Value : Fail
REMARKS

6.5.7 ImCamSetFeatureValue

DEFINITION
BOOL ImCamSetFeatureValue(IN INT nCamIndex, IN INT feature, IN INT nValue);
PARAMETERS
nCamIndex : index of the camera to set the feature. **feature :** feature id to set. **nValue :** value of the feature.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.5.8 ImCamSetFeatureDefault

DEFINITION
BOOL SetFeatureDefault(IN INT nCamIndex, IN INT feature);
PARAMETERS
nCamIndex : index of the camera to set the feature. **feature :** feature id to set to the factory default value.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS

6.5.9 ImCamSetAllFeatureDefault

DEFINITION
BOOL ImCamSetAllFeatureDefault(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to set the feature.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to set all the features supported by the camera to the factory default values.

6.5.10 ImCamGetDataBits

DEFINITION
BOOL ImCamGetDataBits(IN INT nCamIndex, OUT INT *pnDataBits);
PARAMETERS
nCamIndex : index of the camera to get information. **pnDataBits :** real valid bit at Y1600 mode.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to get the real valid data bits among the 16bits data transferred from the camera when Y1600 mode is set. The valid data bits is dependent on the camera, so users have to check the camera manual for this function.

6.5.11 ImCamSetAWBRegion

DEFINITION
BOOL InCamSetAWBRegion(IN INT nCamIndex, IN INT nLeft, IN INT nTop, IN INT nRight, IN INT nBottom);
PARAMETERS
nCamIndex : index of the camera to set information. **nLeft, nTop** : x and y coordinates of the left-top point to set region of the AWB. **nRight, nBottom :** x and y coordinates of the right-bottom point to set region of the AWB.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
The region designated by this function defines the region for processing AWB(Auto White Balancing) algorithm and this region has meaning only in auto or one-push mode. This function is dependent on the camera, so users should check camera manual to check whether the camera support this function or not.

6.5.12 ImCamSetDefaultAWBRegion

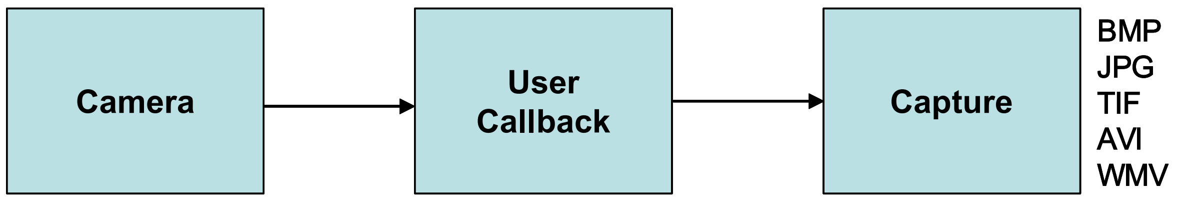
DEFINITION
BOOL ImCamSetDefaultAWBRegion(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to set information.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to set the AWB region to the factory default value. This function is dependent on the camera, so users should check camera manual to check whether the camera support this function or not.

6.6 Size and Capture Functions

The API provides information about the image size currently camera stream out and also provides function which can save this stream data into the image or stream.

If the user callback is registered, the user processed data is passed to the saving functions.

Data flow in capture is as follows.



But, in case WMV capture, user should install Windows Media Encoder because the API use this codec.

6.6.1 ImCamGetImageSize

DEFINITION
BOOL ImCamGetWidth(IN INT nCamIndex, OUT INT *pnWidth, OUT INT *pnHeight);
PARAMETERS
nCamIndex : index of the camera to get information. **pnWidth :** image width size of the current mode. **pnHeight :** image height size of the current mode.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used after calling the ImCamOpen function.

6.6.2 ImCamGetBufSize

DEFINITION
LONG ImCamGetBufSize(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to get information.
RETURN VALUE
Buffer size in bytes for one frame data.
REMARKS
This function is used to get the buffer size for current mode.

6.6.3 ImCamSetCallback

DEFINITION
BOOL ImCamSetCallback(IN INT nCamIndex, IN INT (*pCallbackFunc)(DOUBLE SampleTime, BYTE * pBuffer, LONG lBufferSize));
PARAMETERS
nCamIndex : index of the camera to set. **pCallbackFunc :** pointer of the user callback function.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
If user wants to get the frame data from the camera, user should register his own callback function by calling this function. And the API calls user callback when one frame data is available. In case use of callback function, user should set the first argument of ImCamOpen to TRUE after calling this function with user callback function. User can receive raw image format or RGB24 image format according to the “grabMode” argument of the ImCamOpen function. If user set “grabMode” to IMC_GRAB_RAW, user can receive image data the camera streams unchanged. If user set “grabMode” to IMC_GRAB_RGB888, user can receive RGB24 formatted data. If the callback function is registered, user can receive every frame of the camera, but the process in the callback function can be overhead to the CPU and this can reduce the frame rate transferred to the API. **For the detailed use of this function, please refer to the sample source code.**

6.6.4 ImCamSaveBMP

DEFINITION
BOOL ImCamSaveBMP(IN INT nCamIndex, IN CHAR *pstrFileName, IN BOOL bResize, IN INT nDestWidth, IN INT nDestHeight);
PARAMETERS
nCamIndex : index of the camera to save. **pstrFileName :** file name to save the image. **bResize :** flag for selecting resize. **nDestWidth :** destination width if resize flag is set to TRUE. **nDestHeight :** destination height if resize flag is set to TRUE.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to save frame data as BMP image. This function can save the image in original size or in modified size. If the “bResize” flag is set to “FALSE”, next arguments of size are ignored by the API.

6.6.5 ImCamSaveTIF

DEFINITION
BOOL ImCamSaveTIF(IN INT nCamIndex, IN CHAR *pstrFileName, IN BOOL bResize, IN INT nDestWidth, IN INT nDestHeight);
PARAMETERS
nCamIndex : index of the camera to save. **pstrFileName :** file name to save the image. **bResize :** flag for selecting resize. **nDestWidth :** destination width if resize flag is set to TRUE. **nDestHeight :** destination height if resize flag is set to TRUE.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to save frame data as TIFF image. This function can save the image in original size or in modified size. If the "bResize" flag is set to "FALSE", next arguments of size are ignored by the API.

6.6.6 ImCamSaveJPG

DEFINITION
BOOL ImCamSaveJPG(IN INT nCamIndex, IN CHAR *pstrFileName, IN BOOL bResize, IN INT nDestWidth, IN INT nDestHeight);
PARAMETERS
nCamIndex : index of the camera to save. **pstrFileName :** file name to save the image. **bResize :** flag for selecting resize. **nDestWidth :** destination width if resize flag is set to TRUE. **nDestHeight :** destination height if resize flag is set to TRUE.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to save frame data as JPG image. This function can save the image in original size or in modified size. If the “bResize” flag is set to “FALSE”, next arguments of size are ignored by the API.

6.6.7 ImCamStartAVICapture

DEFINITION
BOOL StartAVICapture(IN INT nCamIndex, IN CHAR *pstrFileName, IN BOOL bUseCallback, IN BOOL bView, IN HWND hPreviewWnd)
PARAMETERS
nCamIndex : index of the camera to capture. **pstrFileName :** file name to save the AVI file(including extension). **bUseCallback :** flag for selecting whether to use the callback or not. **bView :** flag for preview while capturing. **hPreviewWnd** : handle of the preview window. If the bView is true, the API display data in the window designated by this handle.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
If user wants to capture stream after his own processing, user should register callback function and call this function. The processed data will be captured to the captured stream file.

6.6.8 ImCamStopAVICapture

DEFINITION
BOOL ImCamStopAVICapture(IN INT nCamIndex)
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function stops AVI capture.

6.6.9 ImCamStartWMVCapture

DEFINITION
BOOL StartWMVCapture(IN INT nCamIndex, IN CHAR *pstrFileName, IN BOOL bUseCallback, IN BOOL bView, IN HWND hPreviewWnd)
PARAMETERS
nCamIndex : index of the camera to capture. **pstrFileName :** file name to save the AVI file(including extension). **bUseCallback :** flag for selecting whether to use the callback or not. **bView :** flag for preview while capturing. **hPreviewWnd** : handle of the preview window. If the bView is true, the API display data in the window designated by this handle.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to capture the image data as WMV stream data. The file extension can be “asf” or “wmv”. To use this function, user should install Windows Media Encoder 9.

6.6.10 ImCamStopWMVCapture

DEFINITION
BOOL ImCamStopWMVCapture(IN INT nCamIndex)
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function stop WMV capture.

6.7 Format 7(ROI) Functions

The ROI is supported by the camera, functions in this section is supported only the camera which support ROI mode. User should check the camera manual before using these functions.

ROI means that the camera output data only for the area defined by the user, but in this mode all the image size may not be controllable. So, the user refer to the camera manual for valid step size for width and height direction.

6.7.1 ImCamIsROISupport

DEFINITION
BOOL ImCamIsROISupport(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to check.
RETURN VALUE
IMC_SUCCESS : camera supports ROI(Format 7) mode Else : camera doesn't support ROI(Format 7) mode.
REMARKS
This function is used to check whether the camera supports the ROI mode or not. Please refer to the camera manual.

6.7.2 ImCamEnableROIMode

DEFINITION
BOOL ImCamEnableROIMode(IN INT nCamindex);
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to activate ROI mode.

6.7.3 ImCamDisableROIMode

DEFINITION
BOOL ImCamDisableROIMode(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to inactivate the ROI mode.

6.7.4 ImCamGetROISize

DEFINITION
BOOL ImCamGetROISize(IN INT nCamIndex, OUT INT *pnWidth, OUT INT *pnHeight);
PARAMETERS
nCamIndex : index of the camera to control. **pnWidth :** width size of the current ROI mode. **pnHeight :** height size of the current ROI mode.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to get the image size of the current ROI mode.

6.7.5 ImCamGetMaxArea

DEFINITION
BOOL ImCamGetMaxArea(IN INT nCamIndex, OUT PIMC_ROI_PROPERTY pProp);
PARAMETERS
nCamIndex : index of the camera to get information. **pProp :** pointer to get the information.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to get maximum ROI area size supported by the camera. The structure for ROI information is like as follows and defined in the “ImCamDef.h” header file.

```
typedef struct {
        int             hpos;
        int             vpos;
        int             hsize;
        int             vsize;
        int             bpp_min;
        int             bpp_max;
        int             bpp;
} IMC_ROI_PROPERTY, *PIMC_ROI_PROPERTY;
```

6.7.6 ImCamGetCurrentArea

DEFINITION
BOOL ImCamGetCurrentArea(IN INT nCamIndex, OUT PIMC_ROI_PROPERTY pProp);
PARAMETERS
nCamIndex : index of the camera to control. **pProp :** pointer to the ROI information.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to get the area size currently set by the user.

6.7.7 ImCamSetCurrentArea

DEFINITION
BOOL ImCamGetCurrentArea(IN INT nCamIndex, IN IMC_ROI_PROPERTY Prop);
PARAMETERS
nCamIndex : inex of the camera to control. **Prop :** ROI information to set.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function is used to set ROI area size.

6.8 Image Effect Functions

The API provides some image effect functions including mirror, flip and negative and these functions are valid in RGB 24 mode.

The user can enable or disable this effect when open the camera with RGB24 mode.

6.8.1 ImCamSetMirror

DEFINITION
BOOL ImCamSetMirror(IN INT nCamIndex, IN BOOL bMirror);
PARAMETERS
nCamIndex : index of the camera to set. **bMirror** : flag for setting mirror effect.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
When using this function, frame rate can be reduced because the calculated volume of the CPU becomes much larger. Mirror effect can be used only if user calls ImCamOpen with IMC_GRAB_RGB888.

6.8.2 ImCamGetMirror

DEFINITION
BOOL ImCamGetMirror(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : mirror effect is enabled. Else : mirror effect is disabled.
REMARKS

6.8.3 ImCamSetFlip

DEFINITION
BOOL ImCamSetFlip(IN INT nCamIndex, IN BOOL bFlip);
PARAMETERS
nCamIndex : index of the camera to control. **bFlip** : flag for setting flip effect.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
When using flip effect, frame rate can be reduced because the calculated volume of the CPU becomes much larger. Flip effect can be used only if user calls ImCamOpen with IMC_GRAB_RGB888.

6.8.4 ImCamGetFlip

DEFINITION
BOOL ImCamGetFlip(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : flip effect is enabled. Else : flip effect is disabled.
REMARKS

6.8.5 ImCamSetNegative

DEFINITION
BOOL ImCamSetNegative(IN INT nCamIndex, IN BOOL bNegative);
PARAMETERS
nCamIndex : index of the camera to control. **bNegative** : flag for setting negative effect.
RETURN VALUE
IMC_SUCCESS : success Else : fail
REMARKS
When using this effect, frame rate can be reduced because the calculated volume of the CPU becomes much larger. Negative effect can be used only if user calls ImCamOpen with IMC_GRAB_RGB888.

6.8.6 ImCamGetNegative

DEFINITION
BOOL ImCamGetNegative(IN INT nCamIndex);
PARAMETERS
nCamIndex : index of the camera to control.
RETURN VALUE
IMC_SUCCESS : flip effect is enabled. Else : flip effect is disabled.
REMARKS

6.9 Trigger Functions

Trigger is supported by the camera, so user should check the camera manual to see whether the camera support trigger mode or not.

The trigger functions handles only mode setting for hardware trigger.

In trigger mode, all the features is applied as in live mode, and the difference is only the camera outputs one frame data after ths trigger signal is pulsed.

6.9.1 ImCamGetTrigMode

DEFINITION
BOOL ImCamGetTrigMode(IN INT nCamIndex, OUT BOOL *bTrig, OUT INT *pMode, OUT INT *pParam);
PARAMETERS
nCamIndex : index of the camera to get information. **bTrig** : pointer to get the trigger state. **pMode** : pointer to get the mode value. **pParam** : pointer to get parameter for the mode.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to set trigger mode. The trigger mode and parameter is working as defined in the IIDC Specification 1.30.

6.9.2 ImCamSetTrigMode

DEFINITION
BOOL ImCamSetTrigMode(IN INT nCamIndex, IN BOOL bTrig, IN INT nMode, IN INT nParam);
PARAMETERS
nCamIndex : index of the camera to set information. **bTrig** : flag for setting trigger mode(on=TRUE, off=FALSE) **nMode** : trigger mode to set(please refer to the camera manual). **nParam** : trigger mode argument.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to set trigger mode and following parameter for the mode. If user sets the trigger mode to 0, the API ignores parameter argument.

6.10 Register Functions

For the 1394 camera, all cameras should provide memory and register defined by the IEEE1394-1995, IEEE1394-2000 and IIDC Specification.

The API provides function for user to read and write these areas directly.

All the data for reading and writing is handled in 32bits.

6.10.1 ImCamReadQuadlet

DEFINITION
BOOL ImCamReadQuadlet(IN INT nCamIndex, IN ULONG address, OUT ULONG *pulValue);
PARAMETERS
nCamIndex : index of the camera to control. **address** : address of the 1394 Configuration ROM or camera control register to read. **pulValue** : pointer to receive register value.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to read 32-bit data from the configuration ROM address specified in IEEE 1394-1995 and IEEE 1394 –2000 and the Camera Control Register described in IIDC V1.30. With this function, you can read only 32-bit data at once.

6.10.2 ImCamReadBlock

DEFINITION
BOOL ImCamReadBlock(IN INT nCamIndex, IN ULONG ulAddress, IN ULONG nQuadCount, OUT ULONG *pRet);
PARAMETERS
nCamIndex : index of the camera to control. **ulAddress :** address of the 1394 Configuration ROM or camera control register to read. **nQuadCount :** the number of quadlet(32bit unit) to read. **pRet** : pointer to receive return values.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to read 32-bit data from the configuration ROM address specified in IEEE 1394-1995 and IEEE 1394 –2000 and the Camera Control Register described in IIDC V1.30. With this function, you can read any size of data you want.

6.10.3 ImCamWriteQuadlet

DEFINITION
BOOL ImCamWriteQuadlet(IN INT nCamIndex, IN ULONG ulAddress, IN ULONG ulValue);
PARAMETERS
nCamIndex : index of the camera to control. **ulAddress** : address of the 1394 Configuration ROM or camera control register to write. **ulValue** : value to write.
RETURN VALUE
IMC_SUCCESS : Success Else : Fail
REMARKS
This function can be used to read 32-bit data from the configuration ROM address specified in IEEE 1394-1995 and IEEE 1394 –2000 and the Camera Control Register described in IIDC V1.30. With this function, you can write only 32-bit data at once.

7. For previous version user (API Version 1.4.x.x)

The API has been changed to support compilers other than Microsoft Visual C++. For convenience of the users using previous version, we named the new function as follows.

7.1 Naming conventions.

- In the previous version, we export ICamera class itself, but from the version 2.0 we export only the member function of the previous ICamera class as follows.
  : ICamera:(Member Function) is changed to ImCam(MemberFunction)
  ex) ICamera::Open() → ImCamOpen()
  ex) ICamera[0].Open() → ImCamOpen(0, …)

7.2 Return values of the functions.

- The most return value of the function is changed to BOOL which is indicating success or fail.
- Please check the return value of the functions.

7.3 Enum type

- We don't use enum type any more.
- IMC_VIDEO_FORMAT enum type is changed to define values in ImCamDef.h.
  : The elements have same value to the previous version.
  : Change the IMC_VIDE_FORMAT to INT.
- IMC_VIDEO_MODE enum type is changed to define values in ImCamDef.h.
  : The elements have same value to the previous version.
  : Change the IMC_VIDEO_MODE to INT.
- IMC_VIDEO_FRATE enum type is changed to define values in ImCamDef.h.
  : The elements have same value to the previous version.
  : Change the IMC_VIDEO_MODE to INT.
- IMC_FEATURE enum type is changed to define values in ImCamDef.h.
  : The elements have same value to the previous version.
  : Change the IMC_FEATURE to INT
- IMC_GRAB_MODE enum type is changed to define values in ImCamDef.h.
  : The elements have same value to the previous version.
  : Change the IMC_GRAB_MODE to INT

7.4 Function changes

- The "bool" type argument is changed to BOOL type.
- ICamera::GetCurrentFormat is removed. Instead, use ImCamGetVideoFormat.
- ICamera::SetCurrentFormat is renamed to ImCamSetVideoFormat.
- ICamera::SetCurrentMode is removed. Instead, use ImCamSetVideoMode.
- ICamera::GetCurrentMode is renamed to ImCamGetVideoMode.

- ImCamSetVideoMode’s usage is changed.
- ICamera::GetCurrentFrate is changed to ImCamGetFrameRate.
- ImCamOpen’s usage is changed.
- Functions regarding ROI is fixed.
- ImCamSaveResizeBMP, ImCamSaveResizeJPG, ImCamSaveResizeTIF functions merged to ImCamSaveBMP, ImCamSaveJPG, ImCamSaveTIF respectively.
- Please refer to section 5.