Infrared Cameras, Inc.
2105 West Cardinal Drive,
Beaumont, TX 77703
Ph. No. 409-861-0788

Interface Control Document
Revision 3.14

This document is a user's manual for the Software Development Kit supplied as part of the Infrared Cameras, Inc. package.

Camera Core.

The camera system to be supplied will include a documented library (DLL) of functions with which the camera can be accessed. This group of DLL's and this ICD will be referred to as the Software Development Kit or SDK. The camera output data is converted to values of temperature within the supplied DLL software (not hardware). Examples of source code using the DLL package are included in the SDK. However; the DLL itself will be considered Infrared Cameras, Inc. proprietary material and actual source code will not be made available.

The camera system includes the following:

Camera Unit………… Provided by Infrared Cameras, Inc.
USB Cable………….. Provided by Infrared Cameras, Inc.

Infrared Cameras, Inc. accepts no liability in the use and application of the camera. It is the responsibility of the OEM partner to ensure proper training and use of the camera for their particular application.
By developing value added components to this camera unit for that application, the OEM partner certifies that they hold proper and adequate liability insurance for that particular application.

An infrared camera system cannot determine or diagnose faults, problems, or ailments in any form. That determination must be made by the technician applying the camera technology.

Performance

It is recommended that the camera be turned on and allowed to warm up for a short period of time before imaging begins to maximize stability and temperature measurement accuracy. This time will be dependent upon the difference in the thermal environment between the storage condition and the operation condition. The temperature sensors built into the camera system provides indicators relative to this equalization through our libraries. The Infrared Cameras, Inc. camera is subject to the same conditions all micro bolometer radiometers are which creates drift. While our design includes temperature sensors which tracks these conditions, allowing us to model our camera response to these conditions, a warm up period is still recommended.

Our calibration equations include, as variables, the conditions seen by these sensors. During operation, these sensors will be continuously interrogated and the calibration parameters adjusted accordingly by our DLL's. This will limit the drift seen in most radiometers.

The image type provided by the DLL is a 14-bit temperature image in degrees Celsius. This temperature is considered a "black body equivalent temperature". This means that the calculation process assumes that the scene is a black body scene without correction for any variation in emissivity or reflection. To make the correction for these factors theoretically traceable would require that you convert the temperature values to energy, make the correction, and then convert back to temperature by manipulating the Planck Equation. We must assume that the software developer is familiar with this process and can accomplish this task.

It must be highlighted that emissivity and the reflected energy can make a very large impact on the temperature values seen in an image. Laboratory based calibration accuracy statistics do not represent the true temperature accuracy of the application scene. Regardless of the unfortunate claims made by many camera manufacturers regarding measurement accuracy, assuming a credible job in the calibration process, accuracy is determined by the physics of the scene, which is usually defined by surface emissivity and energy reflections from that surface.

Ranges may change based on product and application

It is expected that a re-calibration for each camera be carried out **every six months** in the best case and **once per year** in the worst case.

Following proper power-up and power-down sequencing is vital to proper operation of the camera. The camera should always be connected to the computer before the application is started**. For 7000 series cameras, Never unplug the camera while the program is running.** Plugging and unplugging the USB connection from the camera or computer while the camera is running can and will destroy the camera and at a minimum cause the camera will need to be recalibrated. To avoid this scenario, please always make sure that the DestroyCamera() function is called before your application program exits. As mentioned earlier, failure to follow these steps may result in a long-term reduction of image quality and/or damage to the camera.

The SDK

SDK can be run on Windows Vista/7/8/8.1. We don't support Windows XP. If you need the SDK for Windows XP, please contact us. A sample Visual C++ 2015 project with source code and compiled binaries is provided that demonstrates the use of the various functions described in this document and found in the DLL.

All software development and/or support (non DLL) supplied by Infrared Cameras, Inc. will be accomplished through a separate purchase order and will be billed at an hourly rate.

The following folders are contained on the SDK CD:

CalFiles	- Contains camera calibration files
ICISharedFiles	- Firmware files necessary for operation of the camera
APITest	- Sample Visual C++ 2015 project. Shows proper usage of libraries. Includes precompiled binaries and full source code.
Drivers	- Camera driver files
API	- API DLL files
DemoProgram	- A sample program for running the camera. No source code provided.

Data Types

Name	C/C++	Bits
int32	Long	32
Uint8 / byte	Unsigned char	8
Unit16	Unsigned short	16
Uint32	Unsigned int	32
Int	Signed int	32
float32	Float	32

Using the ICI 7000 and 9000 Series Cameras

This section details the functions available in the SDK to operate the ICI 7000 or 9000 series cameras. All of the functions necessary to initialize and receive images from the camera are contained within this section.

```
//Call InitializedCamera with this argument to initialize either 7000 or 9000 series camera
enum CAMERA_TYPE
{
	ICI_CAMERA_7000 = 0,
	ICI_CAMERA_8000,
	ICI_CAMERA_9000,
	ICI_CAMERA_SWIR
};
```

Function Descriptions:

int stdcall InitializeCamera(int)

This function initializes the camera into a state where images can be captured. This function must be called before any other DLL functions. This function returns 1 on success and 0 on failure.

int __stdcall DestroyCamra(void)

This function "cleans-up" anything needed by the camera before the application exits. This function returns 1 on success and 0 on failure. NOTE: This function must be called before the program terminates to ensure proper camera operation. Improper shut down sequencing may cause adverse effects on the camera function and may eventually lead to decreased image quality and the need for a re-calibration.

int __stdcall GetImageReady(void)

This function returns 1 if a new image is ready for GetNextImage or GetNextImageFloat and returns 0 if not.

*unit16 * __stdcall GetNextImage(void)*

Returns a pointer to the most recently acquired image. The image is 320x240 pixels for 7320 or 9320 and 640x480 pixels for 9320 or 9640. Each pixel is a 16-bit unsigned integer (uint16) representing the temperature in degrees C at that pixel multiplied by 100 for calibrated images, raw sensor data is provided for un-calibrated images. The pixel array is arranged in x, y order with pixel 0,0 first and pixel 319,239/639,279 last. This function returns null on an error and a valid pointer to image data on success.
This pointer returned is a pointer to the image capture thread's internal buffer. This buffer will be re-used next time this function is called.

*float * __stdcall GetNextImageFloat(void)*

Returns a pointer to the most recently acquired image. The image is 320x240 pixels for 7320 or 9320 and 640x480 pixels for 9320 or 9640. Each pixel is a 32-bit floating point number representing the temperature in degrees C at that pixel. The pixel array is arranged in x, y order with pixel 0,0 first and pixel 319,239/639,279 last. This function returns null on an error and a valid pointer to image data on success. This pointer returned is a pointer to the image capture thread's internal buffer. This buffer will be re-used next time this function is called.

Note the following fuctions control motorized focus on the imager. Not all imagers provided by Infrared Cameras, Inc. have a motor focus mechanism. This mechanism is typically supplied as an option and requires a secondary housing for the camera core.

int __stdcall AutoFocusCamera()

- Not currently Implemented

int __stdcall SetCameraFocus(float focus);

- Not currently Implemented

float __stdcall GetCameraFocus(void);

- Not currently Implemented.

long __stdcall SetCameraFocusSpeed(long direction, long speed)

A direction of 1 will focus the camera towards objects that are nearer, 0 will focus to objects farther away. Speed values range from 0-255, with 0 being stopped and 255 being full speed. Because of the nature of motors, there can be a minimum startup value before the motor begins to move and the response may not be linear. The sample program uses speed values of 128 for focus movement.

int __stdcall NucCorrection(HWND wnd)

Applies non-uniformity correction to image. For cameras without a built-in shutter, a surface with a fairly uniform temperature should be placed directly in front of the camera lens. The images coming from the camera are then normalized to the surface presented during the nuc process or nuc flag (shutter) if present.

int __stdcall SetCalibrationFormula(int index)

Specifies which calibration formula should be applied to images coming from the camera. Formula 0 specifies raw sensor data. The default value programmed into the camera is used if no formula is specified by the application. Contact ICI if more information is needed about the operation of this function.

int __stdcall GetCalibrationFormula(void)

Returns the calibration formula currently in use by the API. Contact ICI if more information is needed about the operation of this function.

int __stdcall SetFrameAveraging(int N)

This function will enable averaging of incoming video frames from the camera. The images returned by the API will be the average of the last N images received from the camera.

int __stdcall GetFrameAveraging(void)

Returns the current frame averaging settings. A return value of 1 indicates that no frame averaging is being applied.

int __stdcall GetLastSensorReadings(float *lens, float *fpa)

Retrieves the latest readings form camera internal sensors. Returns 0 on success, other on failure.

float __stdcall GetAverageFPS(void)

Returns the average number of video frames processed per second.

int __stdcall GetCameraStatus(void)

Returns the status of the camera interface. The return values are listed in the table below. The enumerated type ICI7320_CAMERA_STATUS in FGInterface.h defines the return values. They are also listed in the table below.

CS_SHUT_DOWN	0
CS_SEARCHING_FOR_CAMERA	1
CS_INITIALIZZING_CAMERA	2
CS_CAMERA_ERROR	3
CS_CAMERA_RUNNING	4

int __stdcall GetImageSize(long *Width, long *Height)

Retrieves the size of images the camera will return. This should only be called after it has been determined that the camera is operating properly, otherwise improper values will be returned. A good place to put a call to this function is after ***GetImageReady*** () indicates that an image is ready from the camera. Returns 1 on success.

void __stdcall EmisSetWavelength(float w1, float w2)

Defines the wavelengths visible to the camera. Typical values are 3-5 and 8-12. Please verify wavelength values in the camera data sheet. These parameters will be used by the EmisCorrect() function.

void __stdcall EmisSetParameters(float ambient, float emissivity, float transmission)

Defines the ambient temperature, emissivity and transmission values to be used by the EmisCorrect() function. The ambient temperature is in degrees C. The emissivity and transmission values range from 0.0 to 1.0.

float __stdcall EmisCorrect(float observed_temperature)

This function adjusts a single temperature to account for target emissivity and atmospheric conditions. The EmisSetWavelength() and EmisSetParameters() functions need to be called at least once before this function can be used. Input temperature and return value are in degrees C.

long __stdcall UpdateFocusReadings()

This function will interrogate the focus motor of your camera if one is present. This function does not directly return any useable information, the GetFocusPosition and GetFocusSpeed functions use the information this function retrieves from the camera. This function returns 0 on success.

long __stdcall GetFocusPosition(long *position)

This function retrieves the current position of the focus motor . The position this function returns is determined from the data received by the last UpdateFocusReadings call. This function returns 0 on success.

long __stdcall GetFocusSpeed(long *speed, long *direction)

This function retrieves the speed and direction of the focus motor . The information this function returns is determined from the data received by the last UpdateFocusReadings call. This function returns 0 on success.

long __stdcall SetFocusPosition(long position)

This function will command the focus motor of the camera to move to a specified position. The function will return after the camera is commanded to move and the return code does not indicate a successful move, only that the command to move was sent successfully to the camera. This function returns 0 on success. NOTE: The positioning algorithm inside the camera's focus motor is to be considered BETA code and is not guaranteed to be accurate. The algorithm has not been fine tuned as of the writing of this manual. If accurate positioning is required it is recommended that you write a focus algorithm on the PC side to control and monitor the position of the focus motor.

long __stdcall GetCameraSerialNumber()

This function will get the camera serial number. This function returns camera serial number on success. This function returns -1 when DLL doesn't find a camera serial number. This function return 0 on unknown errors.

long __stdcall EnableAutocalibration(bool bEnable)

This functions is only applicable to ICI 9000 series. This function either enables or disables the auto calibration of the camera.

int __stdcall SetAutomaticCalibrationPeriod(const unsigned short nAutomaticCalibrationPeriodInMinutes)
This function is only applicable to ICI 9000 series. This function shall set the period between automatic calibrations (autocal period). This function does not store the autocal period in non-volatile storage to be preserved across power cycles.

Working with images from IR Flash

Beginning with SDK version 3.6, we have added functions to load and analyze images saved from our IR Flash software package. All of these new functions have the 3 letter Img prefix.

Function Descriptions:

long __stdcall ImgDeleteHandle(long ihandle)

This function is used to delete handles previously returned by the ImgLoad function to deallocate the memory associated with them.

Parameter	Type	Description
Ihandle	Long	Image handle

0	Success
Other	Failure

long __stdcall ImgGetBitmap(long ihandle)

This function will return the HBITMAP associated with an image.

Parameter	Type	Description
Name	Const char *	Name of the image to load
Buf	Float *	Buffer to hold output values
X	Int	X position of the pixel to retrieve
Y	Int	Y position of the pixel to retrieve

Return Code	Meaning
0	Error
Other	HBITMAP of image

*const char * __stdcall ImgGetErrorDescription(void)*

This function will retrieve the text description of the results of the last function call to the Img portion of the library.

Parameter	Type	Description
None		

Return Code	Meaning
Any	String containing the description of the last function call's return code

*long __stdcall ImgGetPixelTemperature(long ihandle, float *buf, int x, int y)*

This function will return the temperature of a given pixel in the image. All temperature values are in degrees Celsius.

Parameter	Type	Description
Ihandle	Long	Image handle
Buf	Float *	Buffer to hold output values
X	Int	X position of the pixel to retrieve
Y	Int	Y position of the pixel to retrieve

Return Code	Meaning
0	Success
Other	Failure

*long __stdcall ImgGetTemperatures(long ihandle, float *buf, int buffwidth, int stx, int sty, int endx, int endy)*

This function will return the temperatures of a rectangle and copy them to a buffer. The buffer must be allocated by the user application. All temperature values are in degrees Celsius.

Parameter	Type	Description
Ihandle	Long	Image handle
Buf	Float *	Buffer to hold output values
Buffwidth	Int	Width of the data buffer (buf) in bytes
Stx	Int	Starting x position of the temps to retrieve
Sty	Int	Starting y position of the temps to retrieve
Endx	Int	Ending x position of the temps to retrieve
Endy	Int	Ending y position of the temps to retrieve

Return Code	Meaning
0	Success
Other	Failure

*long __stdcall ImgLoad(const char *name)*

This function will load an image from disk to memory. The name parameter is the name of the image on disk to be loaded. The return value is the image handle that should be used with functions in this library. A return value of 0 indicates error.

Parameter	Type	Description
Name	Const char *	Image handle

0	Failure
Other	Image handle

long __stdcall ImgGetSensorReadings(long iHandle, float* lens, float* fpa);

This function retrieves camera sensor readings (fpa and lens temperature) which was recorded at the time of capture from the image.

Parameter	Type	Description
iHandle	Long	Image handle
lens	Float *	Buffer to store lens
fpa	Float *	Buffer to store fpa

0	Success
Other	Failure