IDentity-1 Iris Scanner

Product Guide

This document provides an overview of the IDentity-1 iris scanner's architecture. It also describes the scanner's API for integration with a biometrics SDK for template creation (iris enrollment & matching).

VIDEOLOGY®
IMAGING SOLUTIONS INC.
Original Equipment Manufacturer

Videology® Imaging Solutions, Inc.
37M Lark Industrial Parkway
Greenville, Rhode Island 02828 USA
Tel: (401) 949 – 5332 Fax: (401) 949 – 5276
Americas, Middle East, Far East & Australia sales:
sales@videologyinc.com

Videology® Imaging Solutions, Europe B.V.
Neutronenlaan 4
5405 NH Uden, The Netherlands
Tel: +31 (0) 413 256261 Fax: +31 (0) 413 251712
Europe & N. Eurasia sales:
info@videology.nl

Version History

Version	Remarks	Date
A	Initial Release	2/20/20
0.2	API update	9/20/19

Table of Contents

1 Overview

Videology's iris scanner IDentity-1 is a 5 Megapixel USB 3.0 iris acquisition solution, which delivers ISO/IEC 19794-6 iris compliant images to the host PC for iris enrollment and matching. IDentity-1 offers the flexibility of a UVC based Windows API for the host PC to communicate with the iris scanner. IDentity-1 can be integrated with any biometric software for template creation to perform iris enrollment and matching. IDentity-1 is designed and manufactured in the USA.

IDentity-1 performs the complete iris recognition process, from video capture to iris segmentation, on-board on a FPGA. This is Videology's unique approach to iris recognition. Our system consists of an iris camera module (32x32x20mm) and 2 IR LED boards (32x15mm each), which are symmetrically mounted inside a tiltable enclosure with a cold mirror in the front to aid with the alignment of the user's eyes. The iris camera module itself can also be easily integrated as an OEM component into a larger identity authentication system.

1.1 Operation Summary

We have integrated the iData SDK for template creation into our Windows viewer to demo iris enrollment and matching. Since this is a fixed license, we don't require a network connection to operate our iris application.

The top-level workflow follows the following scheme:

- The iris scanner receives a trigger from the application viewer.
- The iris scanner begins the auto-capture process to collect frames, assess its focus, adjust exposure and find the irises.
- Once the irises are detected, the scanner sends the frames containing the formatted irises to the viewer.
- We have a handshake mechanism with the viewer that allows us to filter out images below a quality grade of 75.
  (if an iris image pair does not score above 75, the viewer sends another

trigger to the camera, this process repeats up to n times where n is a configurable number set with the environment variable MAXRETRIES)

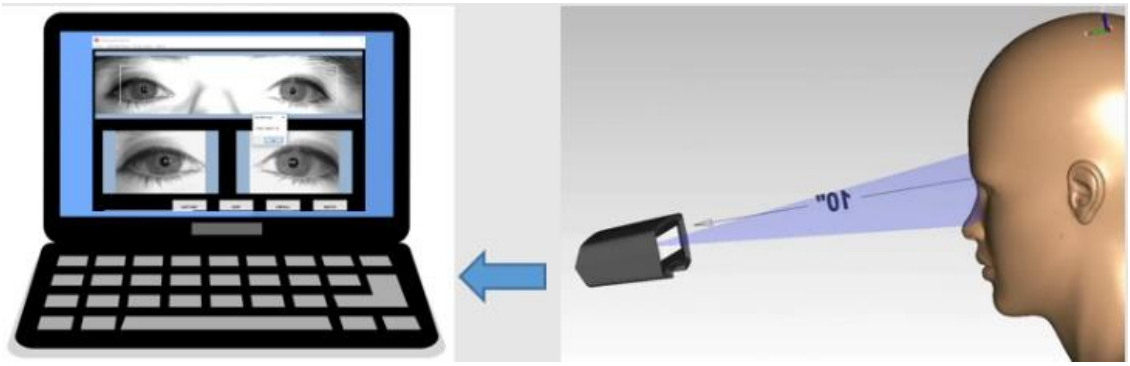


Figure1: IDentity-1 Operation Workflow

1.2 Product Specifications

- ISO/IEC 19794-6:2011 standard compliant images.
- Dual iris capture takes ~2 seconds to deliver 8-bit bitmaps ready for enrollment.
- Operating range = 300 +/- 5 mm
- 810nm IR light source with dual band IR wavelength capability.
- Resolution = 60% @ 4Lp/mm, ≥ 16pixels/mm
- Power Input = 5VDC, power consumption = 4.25Watts.
- PC Interface = USB3.0
- Product Package: Fully enclosed Iris Camera or OEM Iris camera module.
- Iris images can be used with any biometric SDK for enrollment and matching.
- Modular enclosure design allowing for a multimodal docking enclosure configuration with Videology's Photo ID camera (face camera) and/or a fingerprint module.

2 Hardware Architecture

The overall hardware architecture of IDentity-1 consists of the following components:

- Sensor board (5MP monochrome).
- FPGA-USB3 interface board.
- LED driver board (72V0408).
- IR LED boards (standard single wavelength mode operation @ 810nm, dual wavelength mode is also supported).
- Custom M12 lens (object distance = 300mm, resolution @ 60% of 4LP/mm).

Figure 2: IDentity-1 Hardware Architecture

The iris scanner is powered via the USB3.0 bus with a max power consumption of 4.25Watts.

3 Software Architecture

Our FPGA process follows the following steps to deliver an iris image pair to the host application:

a) Frames are captured at 30fps

b) Frames are assessed for focus, frames not in focus within a dual ROI are discarded.

c) In-Focus frames are split in “halves”

d) Split frames are sent to the segmentation engine to locate the iris coordinates.

e) Based on the coordinates of the irises, the eyes are cropped from the full frame into a VGA format.

f) A video multiplexer module inserts the cropped images into the video stream and sends them to the viewer.

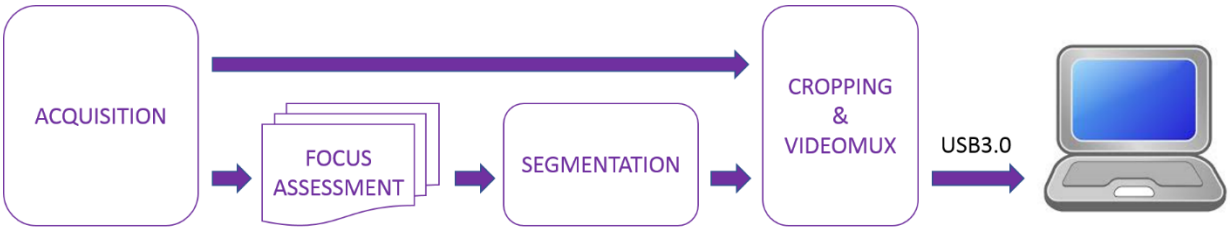


Figure 3: IDentity-1 FPGA Architecture

4 Acquisition Process Description

IDentity-1 produces an ISO 19794-6 iris image pair in 8-bit bitmap format.

Through the capture process, we process multiple frames sequentially. We have a modular mechanism in place to select the appropriate frame for iris segmentation. This includes auto adjustments for exposure, contrast levels and focus assessment. Our focus assessment engine operates basically at frame rate. Contrast settings are designed to facilitate edge detection which is more difficult to attain for certain eye colors.

Our segmentation module locates the iris coordinates, which are used by the subsequent modules to crop each iris from the full frame and to send the iris image pair to the host PC.

Our system architecture processes 30 frames per second and it evaluates the incoming frames in very small groups. If a frame fails the evaluation criteria within any module, it gets immediately discarded and the process runs a new set of frames. When a frame reaches the segmentation module, it takes about 0.5 second to locate the iris in each eye.

Approximately, we use between 24 to 240 frames to produce a good quality iris image. However, this depends on other factors such as the eye color and most importantly the location of the subject within our capture volume.

After completion of all the evaluation steps within our processing flow, IDentity-1 generates an ISO 19764 compliant iris image pair. To guarantee that our processed image has attain a high-quality score, we put a quality score handshake mechanism in place: When the acquired image is delivered to the viewer for display, it gets scored right away. If the score is below a configurable threshold (we use a threshold of 75 for enrollment quality), the viewer discards that image and sends another trigger to the iris scanner to request a new iris image pair. Our viewer offers the option to run with or without the quality score handshake.

Our bitmaps support metadata, we currently use the top row of the image to insert information about the image. We make significant use of this metadata for debugging and data mining.

We constantly collect metrics on our image quality parameters to tune up our camera settings. When our subject is steady within the capture volume, our scores are consistently scoring between mid-80s to mid-90s. However, when the subject lags to get through the capture volume, a marginal image gets processed and the acquisition time takes longer.

Figure 4: IDentity-1 Iris Bitmaps

5 Demo Viewer

IDentity-1 is a user-friendly device. The iris scanner is controlled from the application viewer. The communication between the viewer and the camera happens via UVC commands. Inside the camera, some of the UVC commands are translated into I2C commands.

IDentity-1 comes with a Windows API that handles all the DirectShow callbacks necessary for video management and UVC communication with the scanner. Additionally, for the creation of templates to demo iris enrollment and matching, the IDentity-1 viewer integrates the iData SDK from IrisID which requires a license. Therefore, the IDentity-1 viewer uses 2 DLLs: IDentity1Api.dll and iDataIris.dll

The viewer has an image quality handshake with the camera to send subsequent triggers if the image quality doesn't meet a threshold, which is configurable with the environment variables MAXRETRIES & MINQUALITY.

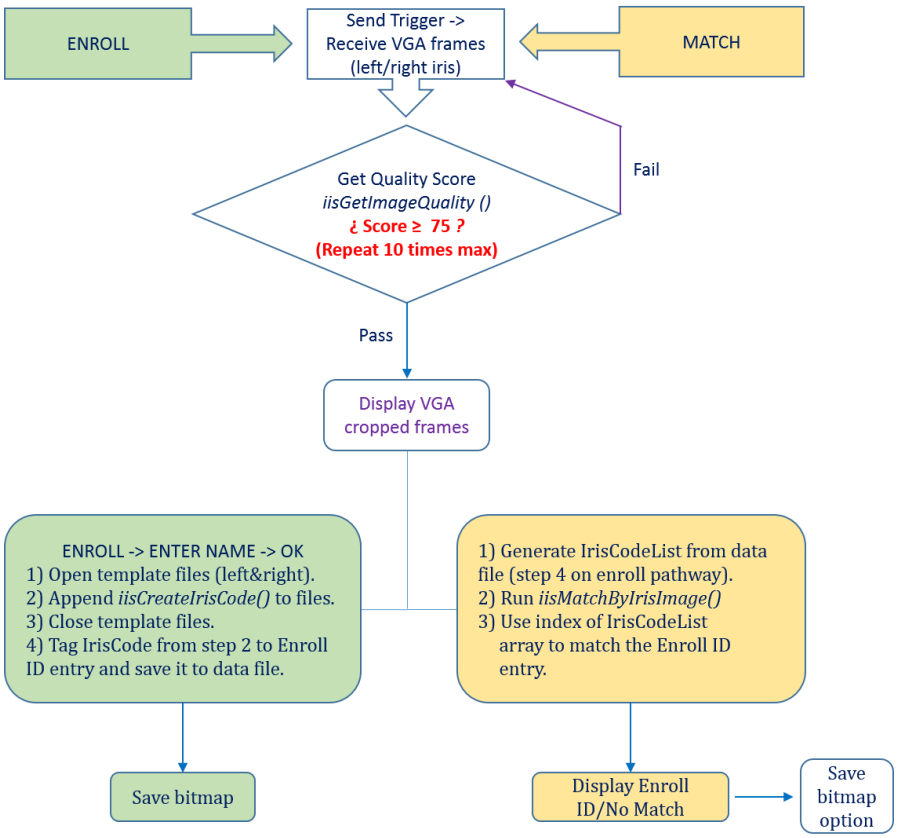


Figure 6: IDentity-1 Viewer Operation with the iData SDK

5.1 Viewer Operation

The viewer performs 4 main functions: enroll, match, capture and save. The enroll and match functions require a 3rd party license for the creation of templates, we use the iData SDK. The capture function retrieves iris bitmaps from the iris scanner without image quality grading. The save function saves the images displayed in the bottom boxes of the viewer.

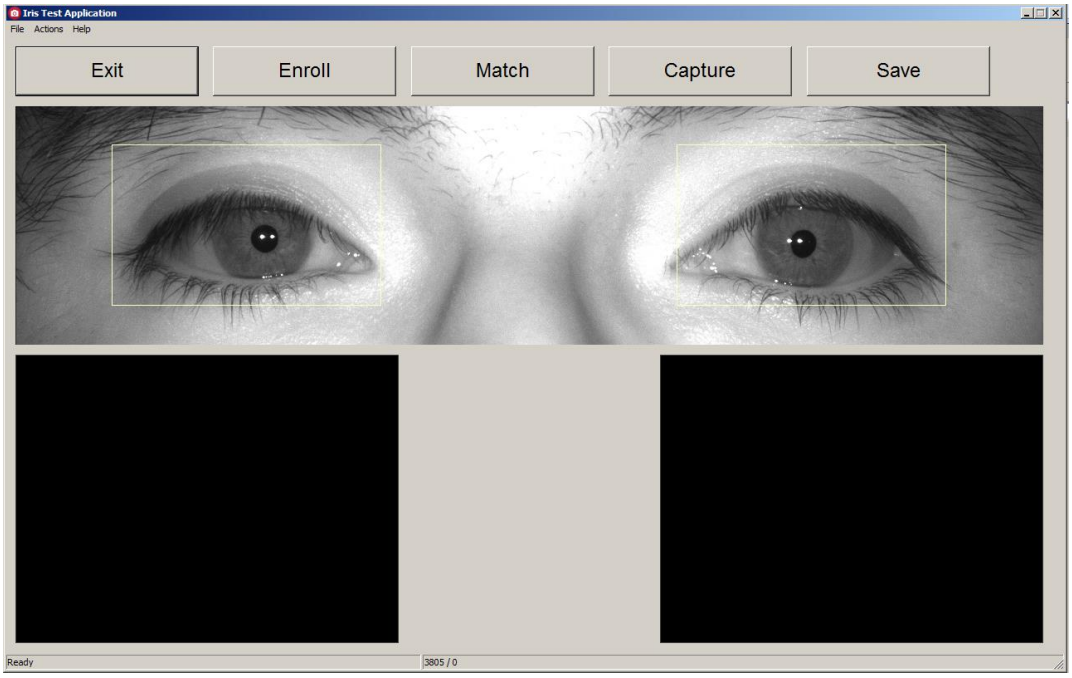


Figure 7: IDentity-1 Viewer

Enroll: the viewer sends a trigger to the scanner to initiate the acquisition process. The scanner starts in auto capture mode and when it locates a human iris, it sends the iris pair to the viewer for quality grading. If the grade is below a certain threshold, another trigger is sent to the scanner. If the quality grade is equal or greater than the threshold, a user record with its associated templates are created. The user record and templates files are automatically stored at C:\ProgramData\Videology\IDentity1\

The image quality threshold and the maximum # of triggers are configurable with the environment variables MINQUALITY and MAXRETRIES.

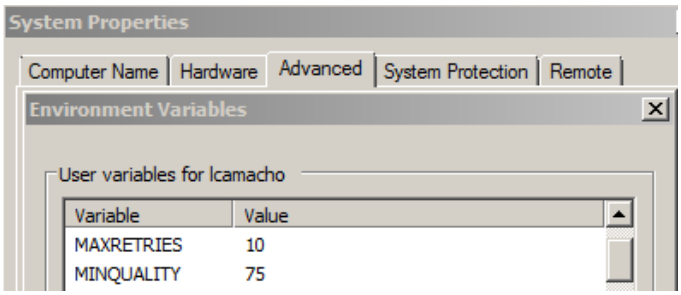


Figure 8: Configurable Environment Variables

Match: the viewer sends a trigger to the scanner to start the acquisition process. If the returned irises meet the quality threshold, the application search for a matching record within the template files. If a match is found, the quality grade and hamming distance is reported.

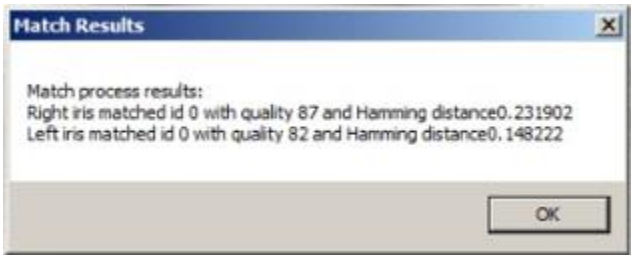


Figure 9: Match Results

Save: the viewer saves the acquired bitmaps to the application folder: C:\ProgramData\Videology\IDentity1\

The image format is 8-bit bitmap with the naming convention "Snapshot[L/R][date][time].bmp"

Capture: the viewer sends a single trigger to the scanner. However, the retrieved iris images are not graded (no quality score handshake). The received iris images are directly displayed in the bottom boxes of the viewer.

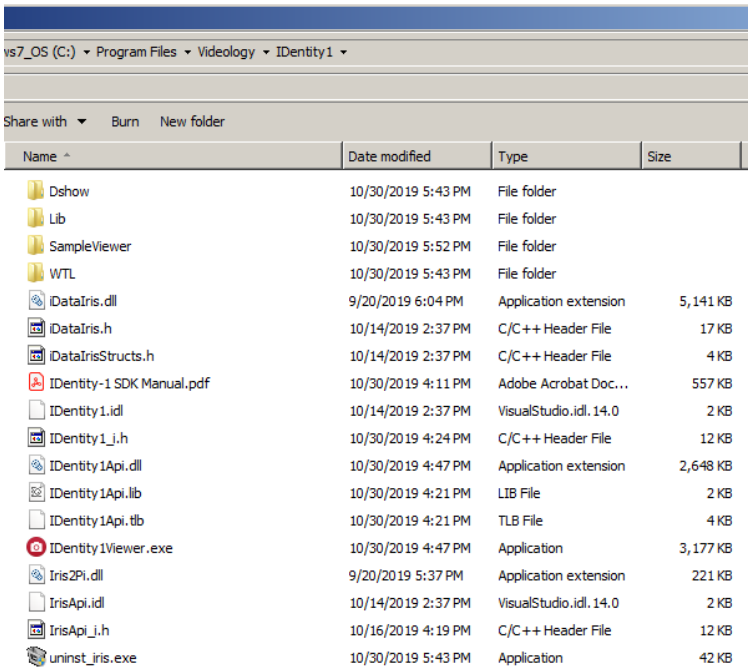
As opposed to the enroll & match functions, capture doesn't require a 3rd party license. This function is useful for SDK integration/evaluation, IDentity1 can be ran with any biometric SDK.

6 IDentity-1 API

The IDentity1 API is designed to help you acquire iris images from the IDentity1 iris scanner in a Windows platform. The API handles the camera interface and the DirectShow graph handling, enabling you to focus on handling frames as delivered. The developer can integrate our iris acquisition API with their biometric SDK of choice to develop a full iris application solution.

Given our UVC architecture, our API is used to retrieve the iris images from the IDentity-1 iris scanner, which are ready to be interfaced with your biometric SDK of choice.

The installation file creates an application folder and registers our Videology DLL in your Windows platform.



Minimum PC requirements:

- Windows based PC (Windows 7)
- Windows 7 64-bit / Windows 10 64-bit
- iCore i5 or equivalent
- 8GB RAM
- Microsoft .NET Framework ver. 3.5
- Dedicated USB3.0 port

USB driver support (Microsoft UVC):

6.1 API Implementation

The Iris API is implemented as an in-process COM server. The API was designed for C++, although it can also be used from C#.

The most convenient way to instantiate the component is to use the Microsoft Visual Studio COM extensions. This allows you to import the component's type library (IrisApi.tlb) directly, which imports all of the structures and function definitions. A C++ header file is also available (IrisAPI_i.h), if you wish to use the ATL COM abstractions.

In one of your header files, up with the #includes, add this as one line. Note that you may need to include a relative path to the type library file.

```
#import "IrisApi.tlb" no_namespace, raw_interfaces_only,
exclude("tagBITMAPINFOHEADER"), exclude("tagSIZE")
…
IVISIrisCameraPtr m_Graph;
…
CoInitialize( NULL );
…
m_Graph.CreateInstance(__uuidof(VISIrisCamera));
```

The interface is:

```
{
      virtual HRESULT Initialize(
      /* [in] */ int ord) = 0;

      virtual HRESULT GetPreviewSize(
            /* [out] */ SIZE *bih) = 0;
```

```
        virtual HRESULT GetIrisSize(
              /* [out] */ SIZE *bih) = 0;

        virtual HRESULT SetCallback(

              /* [in] */ void *pfnCallback,
              /* [in] */ void *pvContext) = 0;

        virtual HRESULT ConvertBitmap(
              /* [in] */ const BITMAPINFOHEADER *biSrc,
              /* [in] */ const uint8_t *pjSrc,
              /* [in] */ BITMAPINFOHEADER *biDst,
              /* [out] */ uint8_t *pjDst) = 0;

        virtual HRESULT SetGrayscalePalette(
              /* [in] */ BITMAPINFOHEADER *bih,
              /* [out] */ ULONG *pulPalette) = 0;

        virtual HRESULT Start( void) = 0;

        virtual HRESULT Stop( void) = 0;

        virtual HRESULT Close( void) = 0;

        virtual HRESULT Trigger( void) = 0;

        virtual HRESULT TriggerReady( void) = 0;

        virtual HRESULT WriteI2C(
              /* [in] */ uint8_t device,
              /* [in] */ uint8_t reg,

        uint8_t data) = 0;

        virtual uint8_t ReadI2C(
              /* [in] */ uint8_t device,
              /* [in] */ uint8_t reg) = 0;
}
```

6.2 API Description

HRESULT Initialize(int ord);

The Initialize method locates the camera and sets up communication with the camera driver. The “ord” parameter specifies which Iris camera should be used, if there are multiple cameras connected to the system, with numbering starting at 0. If the ordinal exceeds the number of cameras, the return value is E_INVALID. If the camera cannot be initialized, the return value is E_FAIL.

HRESULT GetPreviewSize(SIZE *bih);

The GetPreviewSize method returns the size of the preview images. This size will always be 2592 x 600.

HRESULT GetIrisSize(SIZE *bih);

The GetIrisSize method returns the size of the iris section within the streaming images. This size will always be 640 x 480.

HRESULT SetCallback(void *pfnCallback, void *pvContext);

The SetCallback method specifies a callback function to be called when new frames are available. The signature of the pfnCallback method is

```
void ImageCallback(
      void *context,
      int which,
      BITMAPINFOHEADER *bih,
      uint8_t *bits
);
```

The "pvContext" parameter is passed to the callback unchanged. This is a handy way to pass a "this" parameter to a static method in order to call a class method. The "which" parameter is 0 for preview images, and 1 for iris images. The "bih" structure describes the structure of the image bits. This will always indicate a 2592x600 image in Y8 format (8 bits per pixel, grayscale, stored with the top line first). For an iris image, only the upper left 640x480 pixels contain data; the rest of the image is all zero. The "bits" array contains the pixels of the image, a total of 1,555,200 bytes.

The callback is called within the context of the DirectShow graph. The callback should return within a few milliseconds in order to keep the graph flowing correctly.

HRESULT ConvertBitmap(
const BITMAPINFOHEADER *biSrc,
const uint8_t *pjSrc,
BITMAPINFOHEADER *biDst,
uint8_t *pjDst) ;

The ConvertBitmap function provides a service for converting from one bitmap format to another. The function is able to convert between Y8, RGB8,

and RGB24 bitmaps. biSrc points to the source bitmap description. pjSrc points to the source bitmap's pixels. biDst points to the destination bitmap description. pjDst points to the destination bitmap's pixels. The method will fix up fields in biDst to match the incoming bitmap. The method will not change the bitmap size.

HRESULT SetGrayscalePalette(BITMAPINFOHEADER *bih, ULONG *pulPalette);

The SetGrayscalePalette provides a convenient method of creating a palette that allows the Y8 images returned to the callback to be used as RGB8 bitmaps. No copying of the pixels is necessary. Just copy the Y8 BITMAPINFOHEADER to a new BITMAPINFO structure, change the biCompression field to BI_RGB (0), set the biHeight field negative (to indicate top-down), and fill in the palette. The resulting BITMAPINFO can be passed to APIs like SetDIBitsToDevice in order to draw the images on the screen.

In most cases, when using a palettized bitmap format like RGB8, the GDI APIs want the palette to follow the BITMAPINFOHEADER. This can be done using a construct like:

```
struct BITMAPINFOPALETTE : BITMAPINFOHEADER { RGBQUAD pal[256]; }
```

Using that definition, you can initialize the structure by setting up the BITMAPINFOHEADER fields normally and callnig SetGrayscalePalette like this:

```
ZeroMemory( &m_bihIris, sizeof(BITMAPINFOHEADER) );
m_bihIris.biSize = sizeof(BITMAPINFOHEADER);
m_bihIris.biPlanes = 1;
m_bihIris.biBitCount = 8;
m_bihIris.biWidth = 2592;
m_bihIris.biHeight = -600;
m_bihIris.biCompression = BI_RGB;
m_bihIris.biSizeImage = 2592 * 600;
m_bihIris.biClrUsed = 256;
m_bihIris.biClrImportant = 256;
```

```
m_Graph->SetGrayScalePalette( &m_bihIris, (ULONG)&m_bihIris.pal );
```

Now m_bihIris may be passed to any GDI API that requires a BITMAPINFO structure.

HRESULT Start();

The Start method finishes building the DirectShow graph and starts streaming. Make sure you have set up your callback before calling Start to make sure you don't miss any frames.

HRESULT Stop();

The Stop method stops streaming and tears down the DirectShow graph.

HRESULT Close();

The Close method releases any resources allocated in Initialize.

HRESULT Trigger();

The Trigger method signals the camera to begin the iris search process. Firing the trigger can take a long time (a second or more), so this API returns immediately and fires the trigger in a separate thread.

HRESULT TriggerReady();

The TriggerReady command returns S_OK if the trigger process has completed, and S_FALSE if the triggering is still in process.

HRESULT WriteI2C(uint8_t device, uint8_t reg, uint8_t data);

The WriteI2C method sends a request to the camera to write a value to a register in an I2C device. Please refer to the hardware documentation to know which devices are writable.

uint8_t ReadI2C(uint8_t device, uint8_t reg);

The ReadI2C method sends a request to the camera to read a value from a register in an I2C device. Please refer to the hardware documentation to know which devices are readable.

6.3 Sample Code

Videology provides a C++ sample application to get the developer started with the setup and configuration of a viewer for the IDentity-1 iris scanner. The sample code demonstrates the instantiation of the COM extensions to control the camera from a GUI. If you prefer to use C#, you can import the component's type library (IrisApi.tlb) directly.

This process takes care of the iris image acquisition part. Next, the developer can proceed to integrate their biometric SDK of choice for a final iris recognition solution.

The Sample Viewer is a simple viewer that grabs the live video from the scanner into a preview object, triggers the camera and saves the retrieved iris images in 8-bit bitmaps format.

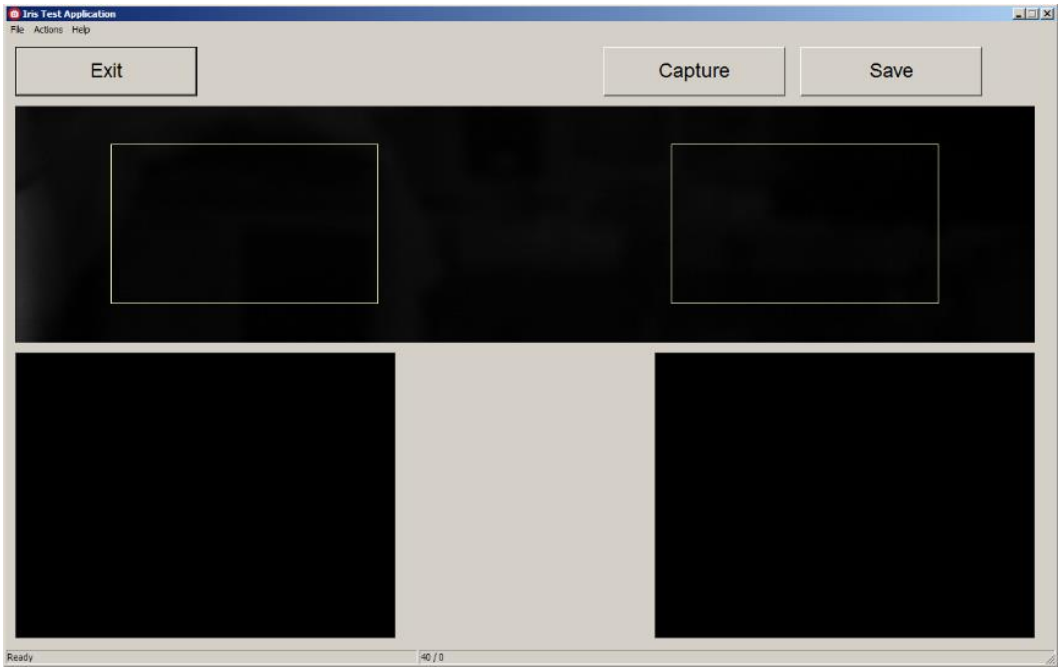


Figure 10: IDentity1 Sample GUI

Capture button: Sends a trigger to the camera in auto capture mode.

Save button: Saves the iris images displayed in the bottom windows of the GUI to the local application folder. The format is 8-bit bitmap with the naming convention "Snapshot[L/R][date][time].bmp"

We recommend the implementation of an image quality handshake between the iris scanner and your biometric SDK. That way, only images with a passing grade are processed for template creation, which is usually controlled by a template meter. Our handshake flow chart is described in figure 6.

6.4 I2C Registers

The IDentity-1 iris scanner has a set of configurable registers for IR LED control (single & dual wavelength), LED indicator control (red/green), frequency mode, start trigger, manual/auto mode, reset and debug. They are accessible through the ReadI2C() and WriteI2C() API functions.

Videology is committed to make your integration simple. Please consult with us if you have any integration or image acquisition questions.

Register #	***Value***	***Description***
0x00	0x0 (default), 0x01	Reset
0x01	0x0 (default), 0x01	Start
0x02	0x0 = Manual 0x1 = Auto (default)	Auto Mode
0x03	0x00 – 0xFF	Auto Max Captures (Max # of captures in auto mode)
0x04	0x0=OFF, 0x1=ON	Debug Mode: Enters RS232 Mode
0x05	0x0=Single (default), 0x1=Dual	Capture Mode: Single or Dual Wavelength
0x06	0x00 – 0xFF	Dual Wavelength ON Interval (# of captures per each wavelength)

0x07	0x0 = 60Hz, 0x01 = 50Hz	Frequency Mode (60/50Hz)
0x08	0x0 = IR1, 0x1 = IR2	IR_SELECT
0x09	0x00 – 0xFF	IR1_DURATION (unit = 1/125th of frame time) @60Hz = 264us @50Hz = 320us
0x0a	0x00 – 0xFF	IR1_OFFSET (unit = 1/125th of frame time) @60Hz = 264us @50Hz = 320us
0x0b	0x0=OFF, 0x1=ON	IR1_CONTINUOUS
0x0c	0x00 – 0xFF	IR2 DURATION (unit = 1/125th of frame time) @60Hz = 264us @50Hz = 320us
0x0d	0x00 – 0xFF	IR2_OFFSET (unit = 1/125th of frame time) @60Hz = 264us @50Hz = 320us
0x0e	0x0=OFF, 0x1=ON	IR2_CONTINUOUS
0x0f	0=OFF, 1=ON	LED_CONTROL
0x10	0=OFF, 1=ON	GREEN LED
0x11	0=OFF, 1=ON	RED LED
0x12	0x00 – 0xFF	LED delay (10ms multiplier)
0x13		Reserved
0x14		Reserved
0x15		Reserved
0x16		Reserved
0x17		Reserved
0x18		Reserved

0x19		Reserved
0x1a		Reserved
0x1b		Reserved
0x1c		Reserved
0x1d		Reserved
0x1e		Reserved
0x1f	0x00 – 0xFF	Firmware Version Number

Figure 11: IDentity1 I2C Register Table

Table of Figures

Technical Support

Europe: Tel: +31 (0) 413 256 261 E-mail: info@videology.nl

USA: Tel: +1 401 949 5332 E-mail: sales@videologyinc.com

Product website: https://www.videologyinc.com/cameras/irisID.htm