

IVIN™ USB Programmer's Reference Manual
Abbreviated Summary Version
DLL Version 0.8, for Windows

Revision March 18, 2008

Copyright 2008 © IVA Corporation

All Rights Reserved

IVIN is a Trademarks of IVA Corporation
(filename IvUsbProgRef0p8aSummary.doc)

Table of Contents

1.0 Introduction 3

2.0 IVIN Architecture 5

3.0 Demos and DLL's 6

 Application Programmer's Interface DLL Driver Files 6

 Console Demos..... 6

 Windows Demo..... 7

4.0 Routines 8

 Basic Routine Summary 8

 Other Routines Summary 8

 Comments Applicable to All Routines..... 9

5.0 Pixel Data Formats 9

1.0 Introduction

The IVIN USB SDK consists of demonstration applications and sources along with the IVIN USB application programmer's interface DLL, named "ivin.dll". The necessary USB device drivers are installed automatically when the standard IVIN USB Windows demo application CD is installed.

The api presented by ivin.dll provides three types of functions, 1) those that provide access and control of the IVIN USB camera, starting with the characters ivIn..., 2) those that provide simplified direct access to hardware video overlays for live video display via Direct X, starting with the characters ivDx..., and 3) those that provide simplified access to AVI stream files, starting with the characters ivAvi..., early releases only include a subset of the ivIn... functions.

Following is a VERY simple (no error checking, etc.) example program that can be run from a console window to acquire continuous live video to a memory buffer, and to grab a frame to "capture.bmp" when the user types "p":

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>          // for _kbhit(), _getch()
#include "ivin.h"

// global image and image info buffers
unsigned char imageRGB[640*480*3]; // XSIZE*YSIZE*0RGB data buffer
IVINIMAGEINFO imagedoneInfo;      // and associated image info buffer

//***** main program *****/
int main(int argc, char *argv[])
{
    int k;
    int status;
    int deviceCount;
    int stop;
    int bytesprovided = 0;
    IVINH *phIvIn = NULL;
    int frameNumber;

    printf("ivin usb example program v0.8\n");

    // Find our how may cameras are accessible. Only use camera #0 for now.
    deviceCount = ivInDeviceCount(NULL); // tcpip address, may be NULL
                                           // for local camera.
    printf("ivin usb device count = %d, available devices 0 to %d\n",
           deviceCount, deviceCount ? deviceCount - 1 : 0);

    if (deviceCount == 0)
    {
        printf("no cameras present, exiting\n");
        exit(1);
    }

    // Get a handle to camera #0
    phIvIn = ivInOpen(
        NULL, // tcpip address, can be NULL for local camera
        0,    // camera device #, 0-based, 0 is first camera
        0,    // session i/d, not needed for local camera
        NULL, // password code, not needed for local camera
        NULL, // inifile of camera initialization parameters, not needed
        NULL, // section name in inifile, not needed
    );
}
```

```

        &status); // return status
if (phIvIn == NULL)
{
    printf("ivInOpen failed, status = %d\n", status);
    exit(1);
}
else
    printf("ivInOpen succeeded\n");

// Now, let's set camera to some reasonable operating values. We should
// really check each status return individually, but to make it more
// readable, not done

// set clock rate, default is 20mHz which is kinda slow
ivInSetSensorClock(phIvIn, IVIN_CLOCK_33MHZ);

// context A (preview, normal mode) will be 640x480, YUV
ivInSetSensorOutput(phIvIn, CONTEXTA, IVIN_SIZE_640X480, 0);

// set the liquid lens focus, 1200 is usually infinity
ivInSetFocus(phIvIn, 1200);

// And start the camera.
stop = ivInStartX(phIvIn, 1); // note mode 1 for auto buffer management

// imagedoneInfo struct needs imagedoneInfo.dwSize to be set
imagedoneInfo.dwSize = sizeof(imagedoneInfo);

// and start looping
printf("Acquiring %d x %d\n", 640, 480);
while (!stop)
{
    if (_kbhit()) // then user is requesting something
    {
        k = _getch(); // get the keyboard character
        if (k == 'p') // snaP a picture
        {
            if (bytesprovided != 0)
            {
                ivInStoreColorBmp(imageRGB,
                    "capture.bmp", 640, 480);
                printf("\nsaved %d x %d capture.bmp\n",
                    640, 480);
            }
            else
                printf("\nno image to save\n");
        }
        else // only other choice is exit
        {
            printf("\nexiting acquisition loop\n");
            break; // exit
        }
    }

    // Get newest frame to image buffer
    bytesprovided = ivInGetNewestFrameX(
        phIvIn, // handle
        imageRGB, // unsigned char *dstImageBuff
        sizeof(imageRGB), // DWORD dstImageMaxSize
        &imagedoneInfo, // IVINIMAGEINFO *imageInfo

```

```

        imagedoneInfo.dwSize,    // DWORD imageInfoMaxSize
        1,                      // int format, 1 is .bmp format bgr
        1000,                   // int timeout msec
        &frameNumber,           // int *frameNumber
        &status);               // int *status

    if (!bytesprovided)        // no image returned
    {
        printf("\nivInGetNewestFrame exception status %d\n",
            status);
        continue;
    }
    // print frame number to verify that we're looping
    printf("\r%d", frameNumber);
}

// we gracefully exited from main loop
while (_kbhit()) // purge kb
    _getch();

// stop video acquisition
status = ivInStop(phIvIn, 1, INFINITE);
if (status)
    printf("\nivInStop failed...");
else
    printf("\nivInStop succeeded...");

// close ivin
if (phIvIn)
{
    status = ivInClose(phIvIn);
    if (status)
        printf("error %d calling ivInClose\n", status);
}

return 0;
}

```

2.0 IVIN Architecture

IVIN is an acronym for Intelligent Video Input device. In contrast to a standard analog or digital video camera, the IVIN incorporates processing capabilities within the camera module itself in order to efficiently implement features such as auto-focus and jpeg compression.

The IVIN2M-USB is a two-board stack which consists of the IVIN2M camera board and the IVIN-USB interface board. The IVIN2M is a 2Mpix (1600x1200) CMOS camera and includes an on-board control microprocessor, JPEG compression engine, auto-white balance, auto-exposure, auto-focus, and liquid lens controller. The IVIN-USB is a USB-2 interface with on-board control microprocessor that manages the movement of data and the command and control of the camera.

The camera can be quickly switched between two contexts, called CONTEXTA and CONTEXTB, with different parameter settings for each context. Usually, CONTEXTA is used for continuous live video, often referred to as preview mode, while CONTEXTB is set up to support higher resolution single frame capture. For example, CONTEXTA can be set up for 320x240, 30FPS (Frames Per Second) live video to observe a scene and position/focus the camera, while CONTEXTB can be set up for 1600x1200 JPEG to enable capture of a single high

quality image upon command. This is desirable, because even though USB-2 has relatively high bandwidth, it still does not have sufficient bandwidth to acquire high resolution images at frame rates suitable for real-time viewing for the purposes of positioning and focusing.

Only CONTEXTB supports JPEG, so in some cases, CONTEXTB is used for continuous video when the desired resolution and frame rate can only be maintained over USB if the data is first compressed. Even high quality JPEG provides at least a 10:1 reduction in image size, so 1600x1200 images can easily be continuously acquired at the maximum rate that the sensor can operate at (currently about 12 FPS at 1600x1200, faster as image size is reduced).

The image sensor array is 1600x1200 pixels. Using the `ivInSetSensorOutput` function, the output image size can be reduced, using high precision interpolation, NOT by discarding pixels. So, for example, a 640x480 output image may be generated, which incorporates contributions from the entire 1600x1200 pixel sensor. Because ALL sensor pixels are used, no frame rate speedup is afforded by this reduction (an exception to this is discussed later). In addition, the `ivInSetSensorFieldOfView` function can be used to select a subrectangle of the 1600x1200 pixels, which WILL provide a frame rate speedup that is proportional to the reduction in total pixels acquired. The `ivInSetSensorOutput` can still be used to reduce the final image size. In the special case that the output size is less than or equal to half the field of view (for example 800x600 or less for a 1600x1200 field of view), then dual A/D's are used to double the acquisition rate. Hence, for example, an 800x600 (or less) output image from a 1600x1200 field of view can run at twice the frame rate as a larger output size.

3.0 Demos and DLL's

Application Programmer's Interface DLL Driver Files

In order to build an application, the file "ivIn.h" must be #included in the application. The application must be linked with "ivIn.lib". The file "ivIn.dll" must be accessible to the application program.

More to be added later.

Console Demos

ivintest0

This program (ivintest0.cpp) demonstrates usage of the ivin usb camera dll (ivIn.dll). It uses the initially released `ivInDrvToAppQ/ivInAppToDrvQ` functions to do buffer management. The newer `ivInGetNewestFrame(X)` function is simpler and may be preferable, see ivintest2 for that version.

After some one-time startup calls, the program enters a continuous video acquisition loop, displaying some real-time info as it loops.

The following actions can be performed by hitting the indicated keys:

- ? - display this information
- x - exit
- p - capture image to capture.bmp
- d - display a bunch of interesting statistics
- e/E - freeze/unFreeze auto exposure
- l/L - freeze/unFreeze flicker filter
- w/W - freeze/unFreeze auto white balance
- a/A - freeze/unFreeze all of the above
- f/F - decrease/increase focus value

Note that this example (and ivintest1, ivintest2, and ivintest3/s) also includes (in the updateHelper function) code to update the usb driver and or camera eeprom if necessary. However, example0 and example1 do not, so make sure to run either ivintest0, 1, 2, or 3/S first to insure that drivers and firmware are at the correct level..

ivintest1

This example is very similar to ivintest0, but because ivInYuvToBgr24 was sped up by a factor of approx 10x, that call is now included directly in image acquisition loop.

ivintest2

This example is very similar to ivintest1, but instead of using ivInDrvToAppQ/ivInAppToDrvQ for buffer management, it uses ivInGetNewestFrameX, which makes for a simpler program.

ivintest3

This example is based on ivintest2, but demonstrates how to use multiple cameras, only one in use at a time. A future release will support multiple cameras simultaneously in use.

ivintest3s

This example is a slightly modified version of ivintest3, which was used to investigate and fix a bug exposed by ivintest3. The bug was a problem in ivin.dll which was fixed, so it no longer is a problem in any example program.

example0


This is the very simple example program shown at the beginning of the USB SDK manual. Note that neither example0 or example1 include code to update the usb driver and or camera eeprom if necessary, so make sure to run either ivintest0, 1, 2, or 3/S first to insure that drivers and firmware are at the correct level..

example1

This example is similar to example0, but shows how to acquire continuous JPEG images, which is useful if higher image resolution is required than can be supported by the available USB bandwidth. The associated file maxSettings.h shows typical maximum bandwidth settings for most resolutions with raw vs. JPEG formats.

Windows Demo

More to be added later.



4.0 Routines

Basic Routine Summary

ivInOpen	open a handle to the camera, acquire necessary resources
ivInClose	close the camera handle, release acquired resources
ivInSetSensorClock	set the camera's pixel clock rate
ivInSetSensorFieldOfView	set the camera sensor's field of view
ivInSetSensorOutput	set the camera's output size and raw/jpeg
ivInSetSensorColorMono	set color/mono
ivInSetAuto	set autoexposure parameters
ivInGetAuto	get autoexposure parameters
ivInSetFocus	set liquid lens focus value
ivInStartX	start video acquisition extended version
ivInGetNewestFrameX	get newest video frame extended version
ivInCapture	capture a single frame
ivInStop	stop video acquisition
ivInYuvToBgr24	convert raw YUV to BGR24 data
ivInYuvTo0Rgb32	convert raw YUV to 0RGB32 data
ivInStoreColorBmp	store BGR24 data to bmp file
ivInGetColorJpegX	retrieve raw jpeg and add standard header
ivInStoreColorJpeg	retrieve raw jpeg, add standard header, store to file

Other Routines Summary

ivInGetDllVersion	get ivin.dll version number
ivInDeviceCount	get number of cameras
ivInGetDeviceInfo	get specific camera device information
ivInGetSecurity	get info needed to connect to remote camera
ivInUpdateUsbDriver	update usb driver to current required version
ivInWriteCameraEeprom	update camera firmware to current required version
ivInVerifyCameraEeprom	verify camera firmware
ivInSetSensorClockX	set the camera's pixel clock rate extended version
ivInSetSensorFieldOfViewX	set the sensor field of view extended version
ivInSetSensorOutputX	set camera output size extended version
ivInSetUsb	set usb transfer type
ivInGetEffectiveGain	convert raw gain info to standard number
ivInStart	old version of ivInStart (recommend using ivInStartX)
ivInGetNewestFrame	old version of get newest video frame (recommend using ivInStartX)

ivInAppToDrvrQ	application to driver command queue
ivInDrvrToAppQ	driver to application command queue
ivInGetColorJpeg	old version of retrieve raw jpeg and add standard header (recommend using ivInGetColorJpegX)
ivInLoadIniFile	load camera parameters from ini file

Comments Applicable to All Routines

Most functions require a handle to the camera named IVINH which is obtained with a call to ivInOpen. The exceptions are ivInGetDllVersion, ivInDeviceCount, ivInGetDeviceInfo, and ivInGetSecurity which are used to find out information about the cameras attached to a system that may need to be known before ivInOpen is called. ivInClose should always be called when done to release the resources acquired by ivInOpen. ivInOpen does quite a few setup operations to the camera, and should generally be done once at a global application level, NOT in some constructor/destructor that is called everytime something happens.

Most functions return an int which is 0 for success, or a non-zero error code elaborated in IVIN.H. An exception is ivInOpen, which returns the handle IVINH.

The file "ivin.h" contains constant values, function prototypes, error values, and structure definitions for all the routines discussed below. **Always refer to the current "ivin.h" supplied with a particular release for the last word on any such details.**

5.0 Pixel Data Formats

Image pixel data memory formats are shown in the following tables. For a more detailed discussion about the relationship between YUV and BGR24 and 0RGB24 data, see the following section titled "Composite Color Data".

BGR24

Byte 0	Blue	B7	B6	B5	B4	B3	B2	B1	B0
Byte 1	Green	G7	G6	G5	G4	G3	G2	G1	G0
Byte 2	Red	R7	R6	R5	R4	R3	R2	R1	R0

0RGB32

Byte 0	Blue	B7	B6	B5	B4	B3	B2	B1	B0
Byte 1	Green	G7	G6	G5	G4	G3	G2	G1	G0
Byte 2	Red	R7	R6	R5	R4	R3	R2	R1	R0
Byte 3	Zero	0	0	0	0	0	0	0	0

YUV 4:2:2

Byte 0	Y1	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 1	U	U7	U6	U5	U4	U3	U2	U1	U0
Byte 2	Y2	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 3	V	V7	V6	V5	V4	V3	V2	V1	V0