Artemis SDK

Version 3.55.0.0

Introduction

The Artemis Software Development Kit (SDK) provides easy access to the functions in the Artemis camera driver DLL. Using the SDK is much more straightforward than managing the low-level USB communication, and has the added benefit of protecting applications from any future changes to the low-level protocol, as well as providing a common software interface to a range of different camera types.

Installation

Installation of the SDK is simply a matter of running the installer program, ArtemisSDK.exe. The installation includes the SDK itself and a sample application, Snapshot, which is a very simple image capture program.

Software environment

The Artemis library and SDK were developed using Microsoft Visual C/C++ V6.0. Provided that the relevant calling conventions are adhered to, it should be possible to use the SDK with other versions of C/C++, and it should also be possible to adapt the SDK for use with other languages such as Visual Basic if necessary - however a VB interface is not part of this SDK.

Basic Principles

1. Connection to driver

Since there is an essentially common interface to a number of cameras in the Artemis range, the first thing an application has to do is to select the DLL it is to use. For the standard Artemis cameras, this is ArtemisHSC.dll and so the SDK has to be connected to the driver using the call:

```
    ArtemisLoadDLL("ArtemisHSC.dll");
```

For the USB1 cameras the relevant dll is loaded using:

```
    ArtemisLoadDLL("ArtemisCCD.dll");
```

This will return TRUE if successful, or FALSE if the DLL (or a DLL which it requires) cannot be loaded. At the end of the session it is good practice, although not essential, to disconnect from the driver using:

```
    ArtemisUnLoadDLL( );
```

2. Connection to camera

Once the driver is selected, you must choose which camera to connect to. Each driver can potentially communicate with multiple cameras of the same type, each of which is given an index number when it is connected to the USB. The index number is not necessarily the same as the order in which the devices are connected. To connect to a camera, use the function

```
    ArtemisConnect(int Device);
```

which returns a handle that must be used in all subsequent communication with the camera. The Device number can be a positive integer if you know the index number of the camera you are connecting to, or it can be -1 to connect to the first available camera.

3. Taking an exposure

With the connection established you can interrogate the camera to discover its sensor dimensions, or simply trigger an exposure which will default to full-frame. While the exposure is in progress you can query the camera's status, which will indicate whether the camera is exposing, downloading, idle, etc. During the download you can ask how much of the image has been received. Finally you can retrieve the downloaded image for display, saving, processing, etc.

4. Colour information

If the camera is programmed with colour information then this can be retrieved using the function ArtemisColourProperties, which always returns ARTEMIS_OK:

```
int ArtemisColourProperties(
        ArtemisHandle hCam,
        ARTEMISCOLOURTYPE *colourType,
        int *normalOffsetX, int *normalOffsetY,
        int *previewOffsetX, int *previewOffsetY);
```

The function returns two sets of bayer offsets, one for imaging with preview mode disabled (normalOffsetX and normalOffsetY) and one for imaging with preview mode enabled (previewOffsetX and previewOffsetY), since in general the bayer offsets are different for the two imaging modes. The offsets are the pixel (x,y) coordinates of that red filter array element which is closest to (0,0), and they are guaranteed to be always either 0 or 1. The offsets are only changed if the return value of *colourType is ARTEMIS_COLOUR_RGGB (see the definition of ARTEMISCOLOURTYPE below).

5. Shutter control

There is no function for opening and closing the shutter, but the API provides a means to keep the shutter closed for dark frames. The function ArtemisSetDarkMode (see below) enables/disables dark mode, ie the mode in which the shutter is to be kept closed during exposures, and the function ArtemisGetDarkMode can be used to determine whether or not dark mode is enabled. Additionally, the cameraflags parameter in the ARTEMISPROPERTIES structure which is returned by the ArtemisProperties function indicates whether or not the camera has a mechanical shutter (see the definition of ARTEMISPROPERTIESCAMERAFLAGS below).

Definitions

There follows a list of definitions and functions included in the SDK, with a brief description of each one and its parameters.

Prototypes and definitions for the functions can be found in the files ArtemisHSCAPI.h and .cpp (or in the files ArtemisCCDAPI.h and .cpp for the USB1 cameras).

Error codes returned by various functions:

```
enum ARTEMISERROR
{
     ARTEMIS_OK = 0,
     ARTEMIS_INVALID_PARAMETER,
     ARTEMIS_NOT_CONNECTED,
     ARTEMIS_NOT_IMPLEMENTED,
     ARTEMIS_NO_RESPONSE,
     ARTEMIS_INVALID_FUNCTION,
};
```

Other enumerated types

```
enum ARTEMISCOLOURTYPE
{
     ARTEMIS_COLOUR_UNKNOWN = 0,
     ARTEMIS_COLOUR_NONE,
     ARTEMIS_COLOUR_RGGB
};

enum ARTEMISPRECHARGEMODE
{
     PRECHARGE_NONE = 0,          // Precharge ignored
     PRECHARGE_ICPS,              // In-camera precharge subtraction
     PRECHARGE_FULL,              // Precharge sent with image data
};

enum ARTEMISPROPERTIESCCDFLAGS
{
     ARTEMIS_PROPERTIES_CCDFLAGS_INTERLACED =1, // CCD is interlaced type
     ARTEMIS_PROPERTIES_CCDFLAGS_DUMMY=0x7FFFFFFF // force size to 4 bytes
};

enum ARTEMISPROPERTIESCAMERAFLAGS
{
// Camera has readout FIFO fitted
     ARTEMIS_PROPERTIES_CAMERAFLAGS_FIFO =1,
// Camera has external trigger capabilities
     ARTEMIS_PROPERTIES_CAMERAFLAGS_EXT_TRIGGER =2,
// Camera can return preview data
     ARTEMIS_PROPERTIES_CAMERAFLAGS_PREVIEW =4,
// Camera can return subsampled data
     ARTEMIS_PROPERTIES_CAMERAFLAGS_SUBSAMPLE =8,
// Camera has a mechanical shutter
     ARTEMIS_PROPERTIES_CAMERAFLAGS_HAS_SHUTTER =16,
// Camera has a guide port
     ARTEMIS_PROPERTIES_CAMERAFLAGS_HAS_GUIDE_PORT =32,
// Camera has GPIO capability
     ARTEMIS_PROPERTIES_CAMERAFLAGS_HAS_GPIO =64,
// force size to 4 bytes
     ARTEMIS_PROPERTIES_CAMERAFLAGS_DUMMY=0x7FFFFFFF
};
```

```
enum ARTEMISCAMERASTATE
{
     CAMERA_ERROR = -1,
     CAMERA_IDLE = 0,
     CAMERA_WAITING,
     CAMERA_EXPOSING,
     CAMERA_READING,
     CAMERA_DOWNLOADING,
     CAMERA_FLUSHING,
};
```

Parameters for ArtemisSendMessage

```
enum ARTEMISSENDMSG
{
     ARTEMIS_LE_LOW              =0,
     ARTEMIS_LE_HIGH             =1,
     ARTEMIS_GUIDE_NORTH         =10,
     ARTEMIS_GUIDE_SOUTH         =11,
     ARTEMIS_GUIDE_EAST          =12,
     ARTEMIS_GUIDE_WEST          =13,
     ARTEMIS_GUIDE_STOP          =14,
};
```

Parameters for ArtemisGet/SetProcessing

```
enum ARTEMISPROCESSING
{
     ARTEMIS_PROCESS_LINEARISE   =1,// compensate for JFET nonlinearity
     ARTEMIS_PROCESS_VBE         =2,// adjust for 'Venetian Blind effect'
};
```

camera/CCD properties

```
struct ARTEMISPROPERTIES
{
     int Protocol;
     int nPixelsX;
     int nPixelsY;
     float PixelMicronsX;
     float PixelMicronsY;
     int ccdflags;
     int cameraflags;
     char Description[40];
     char Manufacturer[40];
};
```

Handle type definition

```
typedef void* ArtemisHandle;
```

Interface Functions

```
// Return API version. XYY X=major, YY=minor
extern  int ArtemisAPIVersion();

// ** Deprecated - use ArtemisDeviceName instead
// Get USB Identifier of Nth FTDI device. Return false if no such device.
// pName must be at least 40 chars long.
extern  bool ArtemisFTName(int Device, char *pName);

// ** Deprecated - use ArtemisDeviceSerial instead
// Get USB Serial number of Nth FTDI device. Return false if no such
device.
// pName must be at least 40 chars long.
extern  bool ArtemisFTSerial(int Device, char *pName);

// ** Deprecated - use ArtemisDeviceIsCamera instead
// Return true if Nth FTDI device exists and is a camera.
extern  bool ArtemisIsCamera(int Device);

// Get USB Identifier of Nth USB device. Return false if no such device.
// pName must be at least 40 chars long.
extern  bool ArtemisDeviceName(int Device, char *pName);

// Get USB Serial number of Nth USB device. Return false if no such device.
// pName must be at least 40 chars long.
extern  bool ArtemisDeviceSerial(int Device, char *pName);

// Return true if Nth USB device exists and is a camera.
extern  bool ArtemisDeviceIsCamera(int Device);

// Disconnect from given device.
// Returns true if disconnected as requested
extern  bool ArtemisDisconnect(ArtemisHandle hCam);

// Connect to given device. If Device=-1, connect to first available
// Returns handle if connected as requested, else NULL
extern  ArtemisHandle ArtemisConnect(int Device);

// Disconnect all connected devices
extern  bool ArtemisDisconnectAll();

// Returns TRUE if currently connected to a device
extern  bool ArtemisIsConnected(ArtemisHandle hCam);

// Fills in pProp with camera properties
extern  int ArtemisProperties(ArtemisHandle hCam, struct ARTEMISPROPERTIES
*pProp);

// Displays the Artemis setup dialog, if any
extern  int ArtemisSetupDialog();

// Abort exposure, if one is in progress
extern  int ArtemisAbortExposure(ArtemisHandle hCam);

// Set the start x,y coords for imaging subframe.
// X,Y in unbinned coordinates
extern  int ArtemisSubframePos(ArtemisHandle hCam, int x, int y);

// Set the width and height of imaging subframe
// W,H in unbinned coordinates
extern  int ArtemisSubframeSize(ArtemisHandle hCam, int w, int h);
```

```
// set the pos and size of imaging subframe inunbinned coords
extern  int ArtemisSubframe(ArtemisHandle hCam, int x, int y, int w, int
h);

// Get the pos and size of imaging subframe
extern  int ArtemisGetSubframe(ArtemisHandle hCam, int *x, int *y, int *w,
int *h);

// Set the x,y binning factors
extern  int ArtemisBin(ArtemisHandle hCam, int x, int y);

// Get the x,y binning factors
extern  int ArtemisGetBin(ArtemisHandle hCam, int *x, int *y);

// Set the Precharge mode
extern  int ArtemisPrechargeMode(ArtemisHandle hCam, int mode);

// Clear the VRegs
extern  int ArtemisClearVRegs(ArtemisHandle hCam);

// Set the FIFO usage flag
extern  int ArtemisFIFO(ArtemisHandle hCam, bool bEnable);

// Start an exposure
extern  int ArtemisStartExposure(ArtemisHandle hCam, float Seconds);

// Prematurely end an exposure, collecting image data.
extern  int ArtemisStopExposure(ArtemisHandle hCam);

// Retrieve the downloaded image as a 2D array of type VARIANT
extern  int ArtemisGetImageArray(ArtemisHandle hCam, VARIANT *pImageArray);

// Retrieve image dimensions and binning factors.
// x,y are actual CCD locations. w,h are pixel dimensions of image
extern  int ArtemisGetImageData(ArtemisHandle hCam, int *x, int *y, int *w,
int *h, int *binx, int *biny);

// Upload a compressed object file to the Artemis PIC
extern  int ArtemisReflash(ArtemisHandle hCam ,char *objfile);

// Return true if amp switched off during exposures
extern  bool ArtemisGetAmplifierSwitched(ArtemisHandle hCam);

// Set whether amp is switched off during exposures
extern  int ArtemisSetAmplifierSwitched(ArtemisHandle hCam, bool
bSwitched);

// Return duration of last exposure, in seconds
extern  float ArtemisLastExposureDuration(ArtemisHandle hCam);

// Return time remaining in current exposure, in seconds
extern  float ArtemisExposureTimeRemaining(ArtemisHandle hCam);

// Return ptr to static buffer containing time of start of last exposure
extern  char* ArtemisLastStartTime(ArtemisHandle hCam);

// Return true if an image is ready to be retrieved
extern  bool ArtemisImageReady(ArtemisHandle hCam);

// Switch off all guide relays
extern  int ArtemisStopGuiding(ArtemisHandle hCam);

// Activate a guide relay for a short interval
extern  int ArtemisPulseGuide(ArtemisHandle hCam, int axis, int milli);
```

```
// Activate a guide relay
extern  int ArtemisGuide(ArtemisHandle hCam, int axis);

// Set download thread to high or normal priority
extern  int ArtemisHighPriority(ArtemisHandle hCam, bool bHigh);

// Retrieve the current camera state
extern  int ArtemisCameraState(ArtemisHandle hCam);

// Percentage downloaded
extern  int ArtemisDownloadPercent(ArtemisHandle hCam);

// Return pointer to internal image buffer (actually unsigned shorts)
extern  void* ArtemisImageBuffer(ArtemisHandle hCam);

// Set the CCD amplifier on or off
extern  int ArtemisAmplifier(ArtemisHandle hCam, bool bOn);

// Set the webcam Long Exposure control
extern  int ArtemisWebcamLE(ArtemisHandle hCam, bool bHigh);

// Reset the camera PIC
extern  int ArtemisReset(ArtemisHandle hCam);

// Get current image processing options
extern  int ArtemisGetProcessing(ArtemisHandle hCam);

// Set current image processing options
extern  int ArtemisSetProcessing(ArtemisHandle hCam, int options);

// Set External Trigger mode (if supported by camera). True=wait for
trigger.
extern int ArtemisTriggeredExposure(ArtemisHandle hCam, bool
bAwaitTrigger);

// Set preview mode (if supported by camera). True=preview mode enabled.
extern  int ArtemisSetPreview(ArtemisHandle hCam, bool bPrev);

// Set subsampling mode (if supported by camera). True=subsampling enabled.
extern  int ArtemisSetSubSample(ArtemisHandle hCam, bool bSub);

// Return true if dark mode is set - ie the shutter is kept closed during
exposures
extern  bool ArtemisGetDarkMode(ArtemisHandle hCam);

// Enable/disable dark mode - ie the shutter is to be kept closed during
exposures
extern  int ArtemisSetDarkMode(ArtemisHandle hCam, bool bEnable);

// Return colour properties
extern  int ArtemisColourProperties(ArtemisHandle hCam, ARTEMISCOLOURTYPE
*colourType, int *normalOffsetX, int *normalOffsetY, int *previewOffsetX,
int *previewOffsetY);
```

```
// Send a packet of data to a peripheral, and receive a reply.
// pSendData points to an 8 byte array which is sent to the peripheral.
// pRecvData points to an 8 byte array which is filled with the
peripheral's response.
// Returns ARTEMIS_OK if the peripheral responds, in which case pRecvData
contains its reply.
// Returns ARTEMIS_NO_RESPONSE if the peripheral doesn't respond.
// If the peripheral does not respond, pRecvData is not guaranteed to be
preserved.
// pSendData and pRecvData may be the same but must not be NULL.
extern  int ArtemisPeripheral(ArtemisHandle hCam, int PeripheralID,
unsigned char *pSendData, unsigned char *pRecvData);

/////////////////////////////////////////////////
// Diagnostic Functions

// Ping the camera, return the result. -1 on error.
extern  int ArtemisDiagnosticPing(ArtemisHandle hCam, int send);

// Ping the FIFO, return the result. -1 on error.
extern  int ArtemisDiagnosticPingFIFO(ArtemisHandle hCam, int send);

// Set the CCD clocks running (firmware doesn't return!)
extern  int ArtemisDiagnosticRunCCD(ArtemisHandle hCam);

// Measure the precharge level
extern  int ArtemisDiagnosticPrecharge(ArtemisHandle hCam);

// Connects to kernel only, safe to use before firmware
// has been uploaded.
// Returns handle if connected as requested
extern  ArtemisHandle ArtemisDiagnosticConnect(int Device);

// Miscellaneous commands to set voltages etc.
// Not to be called if the CCD is installed!
extern  int ArtemisDiagnosticCommand(ArtemisHandle hCam, int cmd);

// Return the last FT USB error condition seen
// Calling this function clears the internal error state
extern  int ArtemisDiagnosticUSBError(ArtemisHandle hCam);

/////////////////////////////////////////////////
// Access LE/Guide ports from 3rd-party software
// msg is the command to execute
// unit is the camera number
// returns:
//  0  OK
//  1  camera busy
//  2  no camera active
//  3  invalid command
extern  int ArtemisSendMessage(int msg, int unit);
```

```
//////////////////////////////////////////////////
// Access peripherals from 3rd-party software
// peripheral is the peripheral's device ID
// send and recv are 8-byte buffers for message-passing
// unit is the camera number
// returns:
//  0  OK
//  1  camera busy
//  2  no camera active
//  3  invalid command
//  4  no response fropm peripheral
extern  int ArtemisSendPeripheralMessage(int peripheral, char *send, char
*recv, int unit);

//////////////////////////////////////////////////
// Get camera description, for 3rd-party software
// recv is a 40-byte buffer for the data
// info tells which data to return:
//  0  camera description from firmware
//  1  FTDI device name
//  2  FTDI device serial number
// unit is the camera number
// returns:
//  0  OK
//  1  camera busy
//  2  no camera active
extern  int ArtemisGetDescription(char *recv, int info, int unit);

//////////////////////////////////////////////////
// Check camera licensing info.
// returns:
// true if camera is licensed, or if no licence required.
// false if camera licence is invalid.
extern  bool ArtemisIsLicensed(ArtemisHandle hCam);

//////////////////////////////////////////////////
// Get camera's input voltage.
// returns:
// vInput=input voltage
// Error code on error
extern  int ArtemisDiagnosticInputVoltage(ArtemisHandle hCam, float*
vInput);

//////////////////////////////////////////////////
// Get camera's ClkSub and Amp voltages.
// returns:
// vClkSub=Clk Sub voltage
// vAmp=Amp voltage
// Error code on error
extern  int ArtemisDiagnosticClkSubAmpVoltage(ArtemisHandle hCam, float*
vClkSub, float* vAmp);

//////////////////////////////////////////////////
// Try to load the Artemis DLL.
// Returns true if loaded ok.
extern bool ArtemisLoadDLL(char *FileName);

//////////////////////////////////////////////////
// Unload the Artemis DLL.
extern void ArtemisUnLoadDLL();
```

The following functions are defined for the HSC (High speed USB) cameras only:

```
//////////////////////////////////////////////////
// Return the temperature indicated by a given sensor.
// A sensor number of 0 will return the number of sensors in the
// temperature parameter.
// A sensor number of 1 or more will return the temperature in
// degrees Celsius * 100
// Error code on error
extern  int ArtemisTemperatureSensorInfo(ArtemisHandle hCam, int sensor,
int* temperature);

//////////////////////////////////////////////////
// Returns information on the Peltier cooler.
// flags: b0-4 capabilities
// b0 0 = no cooling 1=cooling
// b1 0= always on 1= controllable
// b2 0 = on/off control not available 1= on off cooling control
// b3 0= no selectable power levels 1= selectable power levels
// b4 0 = no temperature set point cooling 1= set point cooling
// b5-7 report what's actually happening
// b5 0=normal control 1=warming up
// b6 0=cooling off 1=cooling on
// b7 0= no set point control 1=set point control
// level is the current cooler power level.
// minlvl is the minimum power level than can be set on order to prevent
// rapid warming.
// maxlvl is the maximum power level than can be set or returned, can be
// used to scale power to a percentage.
// setpoint is the current temperature setpoint, in degrees Celsius * 100
// Error code on error
extern  int ArtemisCoolingInfo(ArtemisHandle hCam, int* flags, int* level,
int* minlvl, int* maxlvl, int* setpoint);

//////////////////////////////////////////////////
// Sets the cooling setpoint or power level.
// If the camera supports setpoint cooling (b4 in flags
// from ArtemisCoolingInfo) then setpoint is the desired temperature
// in degrees Celsius * 100. Otherwise if the camera supports power
// level control (b3 in flags) then setpoint is the desired power
// level.
// Error code on error
extern  int ArtemisSetCooling(ArtemisHandle hCam, int setpoint);

//////////////////////////////////////////////////
// Starts the warm-up sequence.
// Error code on error
extern  int ArtemisCoolerWarmUp(ArtemisHandle hCam);

//////////////////////////////////////////////////
// Reconnects the camera's USB controller.
// Error code on error
extern  int ArtemisReconnectUSB(ArtemisHandle hCam);

//////////////////////////////////////////////////
// Return camera type and firmware serial number
// flags low byte=camera type, 1=4021, 2=11002
extern  int ArtemisCameraSerial(ArtemisHandle hCam, int* flags,
int* serial);
```