The Artemis
Software Development Kit

 Version 4.0.0.0

Introduction

The Artemis Software Development Kit (SDK) provides easy access to the functions in the Artemis camera control DLL ArtemisSci.dll, which provides a common interface to all the cameras in the Artemis range.

Installation

The SDK comprises two files in addition to this one: ArtemisSciAPI.h and ArtemisSciAPI.cpp. In order to control a camera, ArtemisSci.dll and associated underlying DLLs must have been installed using the Artemis software installation package.

Software environment

The Artemis library and SDK were developed using Microsoft Visual C/C++ V6. 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.

Basic Principles

Loading the DLL

The first thing an application has to do is to load the DLL, as follows -

```
ArtemisLoadDLL("ArtemisSci.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 unload the DLL as follows -

```
ArtemisUnLoadDLL( );
```

Connecting to the camera

Once the DLL is loaded, you must choose which camera to connect to. The DLL can potentially communicate with a number of different cameras, 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);
```

This 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.

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.

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).

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 ArtemisSciAPI.h and ArtemisSciAPI.cpp.

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,
     ARTEMIS_NOT_INITIALIZED,
     ARTEMIS_OPERATION_FAILED,
};
```

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];
};

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,
// Camera has a window heater
     ARTEMIS_PROPERTIES_CAMERAFLAGS_HAS_WINDOW_HEATER =128,
};

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

Other enumerated types

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

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

Handle type definition

```
typedef void* ArtemisHandle;
```

Interface Functions

DLL management

```
// Load the Artemis DLL.
// Returns true if loaded ok.
extern bool ArtemisLoadDLL(char *FileName);

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

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

USB device information

```
// 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);
```

Connecting and disconnecting

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

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

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

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

Camera Information

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

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

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

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

Exposure settings

```
// 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);
```

Exposure control

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

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

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

// 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);

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

Exposure information and image data

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

// Set a window message to be posted on completion of image download
// hWnd=NULL for no message.
extern  int ArtemisExposureReadyCallback(ArtemisHandle hCam, HWND hWnd, int
msg, int wParam, int lParam);

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

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

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

// 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);

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

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

Cooling and temperature

```
// 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);

// Return 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);

// Set the window heater power
extern  int ArtemisSetWindowHeaterPower(ArtemisHandle hCam, int
windowHeaterPower);

// Get the window heater power
extern  int ArtemisGetWindowHeaterPower(ArtemisHandle hCam, int*
windowHeaterPower);
```

GPIO

```
// Get the number of GPIO lines and the value of the input on each line
// (value of input on nth line given by value of nth bit in lineValues)
extern  int ArtemisGetGpioInformation(ArtemisHandle hCam, int* lineCount,
int* lineValues);

// Set the GPIO line directions
// (nth line is set as an input/output if nth bit of directionMask is 1/0
extern  int   ArtemisSetGpioDirection(ArtemisHandle hCam, int
directionMask);

//Set GPIO output line values
// (nth line (if it's an output) is set to high/low if nth bit of
lineValues is 1/0
extern  int ArtemisSetGpioValues(ArtemisHandle hCam, int lineValues);
```

Guide ports

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

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

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

Low-level control

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

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

// 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);

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

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

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

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