ArtemisHSC.dll Guide:

Introduction:

This document is a simple guide to the common functions in the ArtemisHSC.dll. Only the important functions in the dll are listed below. Any function not listed is either deprecated or only used for specialist purposes.

DLL Info:

Here are the functions which give you information about the DLL itself:

API Version: `int ArtemisAPIVersion()`

Returns the version of the API. This number comes from the services itself.

DLL Version: `int ArtemisDLLVersion()`

Returns the version of the DLL. This number is set in the DLL. It is important to check that the DLL and API version match. If not, there is a strong possibility that things won't work properly.

Device Info:

These functions will give you information from the USB device itself.

Device Present: `BOOL ArtemisDeviceIsPresent(int iDevice)`

Returns a bool to indicate whether the given device is present.

Device Name: `BOOL ArtemisDeviceName(int iDevice, char *pName)`

Sets the supplied ‘pName’ variable to the name of the given device. Return true if iDevice is found, false otherwise.

Device Serial: `BOOL ArtemisDeviceSerial(int iDevice, char *pSerial)`

Sets the supplied ‘pSerial’ variable to the serial number of the given device. Returns true if iDevice is found, false otherwise.

Connecting / Disconnect to cameras:

These functions are used to connect and disconnect from the cameras:

Connect: `ArtemisHandle ArtemisConnect(int iDevice)`

Connects to the given camera. The ArtemisHandle is actually a 'void *' and will be needed for all camera specific methods. It will return 0 if this method fails. You can call this method with '-1', in which case, you will received the first camera that is not currently in use (by any application). Note: It is possible for different applications to connect to the same camera.

Disconnect: `BOOL ArtemisDisconnect(ArtemisHandle hCam)`

This method is used to release the camera when done. It will allow other applications to connect to the camera as listed above.

Disconnect All: `BOOL ArtemisDisconnectAll()`

This method is used to release all the cameras when done. It will allow other applications to connect to the camera as listed above. Although it used to be best to call this function when the application is closing, it is no longer necessary as the DLL itself will remove all connections when shutting down.

Is Connected: `BOOL ArtemisIsConnected(ArtemisHandle hCam)`

Used to check that the given camera is still connected. The most likely reason for this happening is that the camera has been unplugged.

Refresh Devices Count: `int ArtemisRefreshDevicesCount()`

The refresh devices count tells you how many times the camera list has changed on the service. The camera list changes every time a USB device is connected or removed. Therefore, the purpose of this method is to tell the user that the cameras have changed and that it's worth checking to make sure their camera(s) are still connected.

Device Is Camera: `BOOL ArtemisDeviceIsCamera(int iDevice)`

Used to check that a device is a camera before connecting to it. The only alternative is a test bench, so this function is usually of no use.

Camera Info

Once you have connected to a camera, you can use these methods to find out information about the camera

Camera Serial: `int ArtemisCameraSerial(ArtemisHandle hCam, int* flags, int* serial)`

This method is used to find out the serial number of the camera (as written in the EEPROM). Note that this number is likely to be different to the Device Serial which is supplied by the USB device.

This method returns an ArtemisErrorCode as listed below.

Properties: `int ArtemisProperties(ArtemisHandle hCam, struct ARTEMISPROPERTIES *pProp)`

Sets the supplied ARTEMISPROPERTIES to the value for the given camera.

The properties contains the following:

Value	Description
`int Protocol`	The firmware version of the camera
`int nPixelsX`	The width of the sensor in pixels
`int nPixelsY`	The height of the sensor in pixels
`float PixelMicronsX`	The width of each pixel in microns
`float PixelMicronsY`	The height of each pixel in microns
`int ccdflags`	The value is '1' if the sensor is interlaced. 0 otherwise
`int cameraflags`	Represents the properties of the camera: 1 = Has FIFO 2 = Has External Trigger 4 = Can return preview data 8 = Camera can subsample 16 = Has Mechanical Shutter 32 = Has Guide Port 64 = Has GPIO capabilities 128 = Has Window Heater 256 = Can download 8-bit image 512 = Can Overlap exposure 1024 = Has Filter Wheel
`char Description[40]`	The name of the type of camera
`char Manufacturer[40]`	The manufacturer of the camera. Usually Atik Cameras.

This method returns an ArtemisErrorCode as listed below.

Colour Properties: `int ArtemisColourProperties(ArtemisHandle hCam, ARTEMISCOLOURTYPE *colourType, int *normalOffsetX, int *normalOffsetY, int *previewOffsetX, int *previewOffsetY)`

Gives the colour properties of the given camera.

The offsets (Normal / Preview – X / Y) give you information about the Bayer matrix used. The ARTEMISCOLOURTYPE is as follows:

Value	Title	Description
0	ARTEMIS_COLOUR_UNKNOWN	Unknown colour type
1	ARTEMIS_COLOUR_NONE	No colour (Mono camera)
2	ARTEMIS_COLOUR_RGGB	Colour Camera (Bayer Matrix)

This method returns an ArtemisErrorCode as listed below.

Exposures:

These functions are used to start / stop and read the exposures:

Start Exposure: `int ArtemisStartExposure(ArtemisHandle hCam, float Seconds);`

As you might expect, this method is used to start an exposure on the camera. The function will return an error code as listed below. Note: If you have previously stopped the exposure, then you have to wait for the camera state (listed below) to return to Idle before another exposure can be made.

Start Exposure MS: `int ArtemisStartExposureMS(ArtemisHandle hCam, int ms)`

Same as Start Exposure, except the supplied time is in milliseconds instead of seconds.

Stop Exposure: `int ArtemisStopExposure(ArtemisHandle hCam)`

Used to cancel the current exposure.

Image Ready: `bool ArtemisImageReady(ArtemisHandle hCam)`

Let's you know when the image is ready. The value is set to 'false' when 'Start Exposure' is called and only returns true once the exposure has finished.

Camera State: `int ArtemisCameraState(ArtemisHandle hCam)`

Returns the current state of the camera. Values are listed below:

Value	Title	Description
0	CAMERA_IDLE	The default state of the camera. Indicates that the camera is ready to start and exposure
1	CAMERA_WAITING	This is a temporary state between receiving a start exposure command, and the exposure actually starting.
2	CAMERA_EXPOSING	The camera is exposing
4	CAMERA_DOWNLOADING	The service is reading the image off the sensor
5	CAMERA_FLUSHING	Stop Exposure has been called, and we are waiting for everything to stop. Some cameras will still need to clear the sensor, so this can take a while.
-1	CAMERA_ERROR	Something has gone wrong with the camera

Exposure Time Remaining: `float ArtemisExposureTimeRemaining(ArtemisHandle hCam);`

The time (in seconds) that the exposure has left. Note: This does not include the download time which can be significant on some cameras.

Download Percent: `int ArtemisDownloadPercent(ArtemisHandle hCam);`

How much of the image has been download from the sensor. Note: Some cameras will do the whole thing in one go, so the value will jump from 0% to 100%. Other take longer and will reflect how long is left.

Image Data: `int ArtemisGetImageData(ArtemisHandle hCam, int *x, int *y, int *w, int *h, int *binx, int *biny);`

Will populate the given values with the details of the image:

Value	Description
x	The x start position of the image. If you are not subframing the value will be 0.
Y	The y start position of the image. If you are not subframing the value will be 0.
W	The width of the image.
H	The height of the image
xBin	The x-bin value
yBin	The y-Bin value.

The function will return an error code as listed below.

Image Buffer: `void* ArtemisImageBuffer(ArtemisHandle hCam);`

Returns a pointer to the image buffer. You should call ArtemisGetImageData first to find out the dimensions of the image. This will return 0 if there is no image.

Exposure Settings:

These functions are used to set / get the exposure settings:

Set Bin: `int ArtemisBin(ArtemisHandle hCam, int x, int y);`

Used to set the binning values of the exposure. The function will return an error code as listed below.

Get Bin: `int ArtemisGetBin(ArtemisHandle hCam, int *x, int *y);`

This function will populate the given parameters with the current binning values. The function will return an error code as listed below.

Get Max Bin: `int ArtemisGetMaxBin(ArtemisHandle hCam, int *x, int *y);`

This function will populate the given parameters with the maximum binning values for this camera. The function will return an error code as listed below.

Set Sub Frame: `int ArtemisSubframe(ArtemisHandle hCam, int x, int y, int w, int h);`

This function will set the subframe values. The function will return an error code as listed below.

Set Sub Frame Pos: `int ArtemisSubframePos(ArtemisHandle hCam, int x, int y);`

This function will set the x, y position of the subframe. The function will return an error code as listed below.

Set Sub Frame Size: `int ArtemisSubframePos(ArtemisHandle hCam, int x, int y);`

This function will set the size of subframe. The function will return an error code as listed below.

Get Sub Frame: `int ArtemisGetSubframe(ArtemisHandle hCam, int *x, int *y, int *w, int *h);`

This function will populate the given values with the current subframe values. The function will return an error code as listed below.

Set Preview: `int ArtemisSetPreview(ArtemisHandle hCam, bool bPrev);`

Preview mode will produce images at a faster rate, but at a cost of quality. This method is used to set the camera into normal / preview mode. Passing bPrev = true will set the camera into preview mode, 'bPrev = false' sets the camera into normal mode. The function will return an error code as listed below.

Cooling:

These function will allow you to set the cooling of the camera:

Sensor Info: `int ArtemisTemperatureSensorInfo(ArtemisHandle hCam, int sensor, int* temperature);`

This function has two purposes. Firstly, if you call this function with 'sensor = 0', then temperature will actually be set to the number of sensors. Once you know how many sensors there are, you can call this method with a '1-based' sensor index, and temperature will be set to the temperature reading of that sensor. The temperature is in 1/100 of a degree (Celcius), so a value of -1000 is actually -10C The function will return an error code as listed below.

Set Cooling: `int ArtemisSetCooling(ArtemisHandle hCam, int setpoint);`

This function is used to set the temperature of the camera. The setpoint is in 1/100 of a degree (Celcius). So, to set the cooling to -10C, you need to call the function with setpoint = -1000. The function will return an error code as listed below.

Cooling Info: `int ArtemisCoolingInfo(ArtemisHandle hCam, int* flags, int* level, int* minlvl, int* maxlvl, int* setpoint);`

Gives the current state of the cooling. The function will return an error code as listed below.

Warm Up: `int ArtemisCoolerWarmUp(ArtemisHandle hCam);`

Tells the camera to start warming up. Note: It is very important that this function is called at the end of operation. Letting the sensor warm up naturally can cause damage to the sensor. It's not unusual for the temperature to go further down before going up. The function will return an error code as listed below.

Internal Filter Wheel:

These function allow you to control the internal filter wheel of any Atik camera (i.e. AtikOne). To see if a camera has an internal filter wheel, check the camera flags of the ArtemisProperties. (1024)

Filter Wheel Info: `int ArtemisFilterWheelInfo(ArtemisHandle hCam, int *numFilters, int *moving, int *currentPos, int *targetPos);`

Will populate the given values with the current filter wheel info. A moving value of 0 indicates the filter wheel is stationary. The function will return an error code as listed below.

Filter Wheel Move: `int ArtemisFilterWheelMove(ArtemisHandle hCam, int targetPos);`

Will set the filter wheel to the given position. The target position is 0-based. The function will return an error code as listed below.

External Filter Wheel:

To control an external filter wheel (EFW2) you can use the following commands:

EFW Device Present: `bool ArtemisEFWIsPresent(int i);`

Tells you if the device is present. Can be used to loop over the available cameras.

EFW Get Device Details: `int ArtemisEFWGetDeviceDetails(int i, ARTEMISEFWTYPE * type, char * serialNumber);`

Will give you the serial number and type (EFW1 or EFW2) of the given filter wheel. The function will return an error code as listed below.

EFW Connect: `ArtemisHandle ArtemisEFWConnect(int i);`

Connects to the given filter wheel. The ArtemisHandle is actually a 'void *' and will be needed for all filter wheel specific methods. It will return 0 if this method fails. You can call this method with '-1', in which case, you will received the first filter wheel that is not currently in use (by any application). Note: It is possible for different applications to connect to the same filter wheel.

EFW Disconnect: `int ArtemisEFWDisconnect( ArtemisHandle handle);`

This method is used to release the filter wheel when done. It will allow other applications to connect to the filter wheel (using -1) as listed above. The function will return an error code as listed below.

EFW Get Details: `int ArtemisEFWGetDetails(ArtemisHandle handle, ARTEMISEFWTYPE * type, char * serialNumber);`

Will give you the serial number and type (EFW1 or EFW2) of the given filter wheel. The function will return an error code as listed below.

EFW Nmr Position: `int ArtemisEFWNmrPosition( ArtemisHandle handle, int * nPosition);`

Sets nPosition to the number of filter wheel positions. The function will return an error code as listed below.

EFW Get Position: `int ArtemisEFWGetPosition( ArtemisHandle handle, int * iPosition, bool * isMoving);`

Sets 'iPosition' to the current filter wheel position and sets 'isMoving' to true if the wheel is moving, and false otherwise. The function will return an error code as listed below.

EFW Set Position: `int ArtemisEFWSetPosition( ArtemisHandle handle, int iPosition);`

Sets the filter wheel to 'iPosition'. The function will return an error code as listed below.

Guiding:

These functions allow you to control the guiding

Guide: `int ArtemisGuide(ArtemisHandle hCam, int axis)`

Allows you to set the guiding for a specific direction. The axis takes the following values: 1 = North, 2 = South, 3 = East, 4 = West. The function will return an error code as listed below.

Guide Port: `int ArtemisGuidePort(ArtemisHandle hCam, int nibble)`

Allows you to set the guiding in multiple directions at the same time. The nibble is a bit flag value with bits: 1 = North, 2 = South, 3 = East, 4 = West. The function will return an error code as listed below.

Pulse Guide: `int ArtemisPulseGuide(ArtemisHandle hCam, int axis, int milli)`

Allows you to set the guiding for a specific direction for a set amount of time. The axis takes the following values: 1 = North, 2 = South, 3 = East, 4 = West. The function will return an error code as listed below.

Stop Guide: `int artfn ArtemisStopGuiding(ArtemisHandle hCam)`

Stops the guiding. The function will return an error code as listed below.

Artemis Error Codes:

Here are the following error codes:

Value	Title	Description
0	ARTEMIS_OK	The function call has been successful
1	ARTEMIS_INVALID_PARAMETER	One or more of the parameters supplied are inconsistent with what was expected. One example of this would be calling any camera function with an invalid ArtemisHandle. Another might be to set a subframe to an unreachable area.
2	ARTEMIS_NOT_CONNECTED	Returned when calling a camera function on a camera which is no longer connected.
3	ARTEMIS_NOT_IMPLEMENTED	Returned for functions that are no longer used.
4	ARTEMIS_NO_RESPONSE	Returned if a function times out for any reason
5	ARTEMIS_INVALID_FUNCTION	Returned when trying to call a camera specific function on a camera which doesn't have that feature. (Such as the cooling functions on cameras without cooling).
6	ARTEMIS_NOT_INITIALISED	Returned if trying to call a function on something that hasn't been initialised. The only current example is the lens control
7	ARTEMIS_OPERATION_FAILED	Returned if a function couldn't complete for any other reason.