ANDOR

Software Guide

SDK
Software Development Kit

Software Development Kit V3.14

Rev 3.1
23 August 2018

© Andor 2018

Table of Contents

1. INSTALLATION 5
1.1. Technical Support 5
1.2. Rationale 6
1.3. Structure 6
1.4. Key Features 6
1.5. Installation 7
1.5.1. Windows Installation 7
1.5.2. Linux Installation 8
1.6. Getting Started 10
1.6.1. Windows Getting Started 10
1.6.2. Linux Getting Started 12
1.7. Microsoft Application Verifier..12
2. API (APPLICATION PROGRAM INTERFACE) 13
2.1. Overview 13
2.2. Function Listing 13
2.3. API Description..14
2.3.1. Library Initialization 14
2.3.2. Opening a Camera Handle 14
2.3.3. Integer Features 15
2.3.4. Floating Point Features 16
2.3.5. Boolean Features 16
2.3.6. Enumerated Features 17
2.3.7. Command Features 18
2.3.8. String Features 18
2.3.9. Buffer Management 18
2.3.10. Feature Access Control 19
2.3.11. Feature Notifications..20
2.4. Error Codes 22
3. FUNCTION REFERENCE..25
3.1. Function Listing 25
3.1.1. AT_Open 25
3.1.2. AT_OpenDevice 25
3.1.3. AT_Close 25
3.1.4. AT_IsImplemented..25
3.1.5. AT_IsReadOnly 26
3.1.6. AT_IsWritable 26

3.1.7. AT_IsReadable 26
3.1.8. AT_RegisterFeatureCallback 26
3.1.9. AT_UnregisterFeatureCallback 27
3.1.10. AT_InitialiseLibrary 27
3.1.11. AT_FinaliseLibrary 27
3.1.12. AT_SetInt 27
3.1.13. AT_GetInt 27
3.1.14. AT_GetIntMax 28
3.1.15. AT_GetIntMin 28
3.1.16. AT_SetFloat 28
3.1.17. AT_GetFloat 28
3.1.18. AT_GetFloatMax 28
3.1.19. AT_GetFloatMin 28
3.1.20. AT_SetBool 29
3.1.21. AT_GetBool 29
3.1.22. AT_Command 29
3.1.23. AT_SetString 29
3.1.24. AT_GetString 29
3.1.25. AT_GetStringMaxLength 29
3.1.26. AT_SetEnumIndex 30
3.1.27. AT_SetEnumString 30
3.1.28. AT_GetEnumIndex 30
3.1.29. AT_GetEnumCount 30
3.1.30. AT_GetEnumStringByIndex 31
3.1.31. AT_IsEnumIndexAvailable 31
3.1.32. AT_IsEnumIndexImplemented 31
3.1.33. AT_QueueBuffer 31
3.1.34. AT_WaitBuffer 31
3.1.35. AT_Flush 32
4. CAMERA FEATURES 33
4.1. Camera Support 33
4.2. Feature Availibility 33
4.3. Feature Reference 36
4.3.1. Zyla 36
4.3.2. Neo 41
4.3.3. iStar-sCMOS 46
4.3.4. Apogee 52

4.3.5. Sona 56
4.3.6. Marana 61
4.3.7. Balor 65
4.3.8. SimCam 70
4.4. Image Format 73
4.5. Pixel Encoding 75
4.5.1. Mono12Packed 75
4.5.2. Mono12 75
4.5.3. Mono16 75
4.5.4. Mono32 76
4.5.5. Mono8 (Limited Availability) 76
4.6. Metadata 77
4.7. Area of Interest 79
4.8. PixelEncoding and PreAmpGainControl 80
4.9. Sensor Cooling 81
4.10. Comparison of SDK2 and SDK3 82
5. TUTORIAL 83
5.1. Further Examples 86
5.1.1. Initialise Library and Open Camera 86
5.1.2. Simple Single Frame Acquisition 86
5.1.3. Using a Feature 87
5.1.4. Circular Buffer 87
5.1.5. Pixel Encoding 89
5.1.6. Call-Backs 89
6. ADDITIONAL LIBRARIES 91
6.1. ATUTILITY 91
6.2. AT_InitialiseUtilityLibrary 91
6.3. AT_FinaliseUtilityLibrary 91
6.4. AT_ConvertBuffer 91
6.5. Convert Buffer Example 92
6.6. AT_ConvertBufferUsingMetadata 94
6.7. AT_GetWidthFromMetadata 94
6.8. AT_GetHeightFromMetadata 95
6.9. AT_GetStrideFromMetadata 95
6.10. AT_GetPixelEncodingFromMetadata 96
6.11. AT_GetTimeStampFromMetadata 97

1. INSTALLATION

1.1. TECHNICAL SUPPORT

If you have any questions regarding the use of this equipment, please contact the representative, from whom your system was purchased, or:

7 Millennium Way
Springvale Business Park
Belfast
BT12 7AL
Northern Ireland

Tel: +44 (0)28 9023 7126
Fax: +44 (0)28 9031 0792
e-mail: scmossupport@andor.com

1.2. RATIONALE

Andor SDK Version 3, herein referred to as SDK3, has been designed from the ground up to simplify integration of the Andor camera range into your application. Modern scientific digital cameras have become feature rich devices which can be tailored to the particular application into which they are applied. Andor understands that the integration of the camera is just one component of a larger system solution, and, as such a more consistent and scalable API is required to allow the application developer to both quickly prototype basic acquisition functionality and to provide a clear path to exposing the full feature set.

1.3. STRUCTURE

In SDK3, the features that are configurable on the camera have been made independent of the API. There are no API functions for configuring specific features. For example, there is no SetExposureTime function in the API. Instead, the feature that is to be configured is passed into the API function as a parameter. By doing this the number of functions in the SDK3 API is reduced dramatically which in turn reduces the time required to learn the API. This also allows the developer to write generic code that will work for multiple features, again shortening development time. In this manual the API and the camera features are described in separate sections.

1.4. KEY FEATURES

- Simplified API to help reduce development time
- Full access to the current state and limits of camera features
- Support for querying the availability of camera features
- Observer interface to camera features
- Handle parameter in each function to facilitate multiple camera support
- Simple Queue / Wait interface for acquisition buffer management
- Built in software simulated camera (SimCam)

1.5. INSTALLATION

The following sections will describe how to install the software and hardware in order to make your sCMOS camera ready to use.

1.5.1. WINDOWS INSTALLATION

Ensure that you do not install the PCI Express frame grabber card before running any of the software installations.

1. Installation of SDK3 and interface drivers:

[Note: You must have administrator access on your PC to perform the installation.]

- Run the setup.exe file on the cd or from download.
- Select the installation directory or accept the default when prompted by the installer.
- Click on the Install button to confirm and continue with the installation.
- During the installation a number of other windows will pop up as the Camera Link and CXP drivers, and SDK3 are installed. Click on the Finish button when prompted.

2. Install camera interface card as appropriate:

- Shut down your PC.
- Install the PCI Express CL frame grabber card into a free PCI Express slot on your motherboard.
    - Minimum x4 PCIe for Zyla 3-Tap and Neo 3-Tap
    - Minimum x8 PCIe for Zyla 10-Tap
- Install the PCI Express CXP frame grabber card into a free PCI Express slot on your motherboard.
    - Minimum x8 PCIe for Sona or Balor CXP
- Install the PCI Express USB3 card into a free PCI Express slot on your motherboard.
    - Minimum x1 PCIe
    - **Windows 7**: the drivers are installed automatically during start up.
    - **Windows 10**: install the latest drivers from the following location: https://www.startech.com/uk/Cards-Adapters/USB-3.0/Cards/2-port-PCI-Express-USB-3-Card~PEXUSB3S25#dnlds
- Power on the PC.

1.5.2. LINUX INSTALLATION

The SDK3 and Camera Link drivers are distributed as a Linux tar file named andor-sdk3-A.B.C.D.tgz where A.B.C.D is the distribution version code.

1. Installation of SDK3 and Camera Link drivers:

- After you have downloaded the tar file you should open a terminal window and change to the download directory.
- Untar the download file by typing:

```
tar -xf andor-sdk3-A.B.C.D.tgz
```

  where A.B.C.D is replaced by the version information of the file you have downloaded e.g.

```
tar –xf andor-sdk3-3.14.30005.0.tgz
```

- This creates a sub-folder “andor”.
- Change to the “andor” directory and type:

```
sudo ./install_andor
```

- If the install script is unable to determine the platform you will be prompted to enter the platform i.e.

```
Platform cannot be automatically determined. Please select platform to install:
1. 32-bit
2. 64-bit
3. Exit
Selection:
```

- Enter appropriate selection, e.g. 2
- The following warning will then be displayed:

```
This setup will install several libraries into,
 /usr/local/lib and
 /usr/local/bin
Continue (y/n)?
```

- Enter “y” to continue.
- If the installation is successful you should see the following:

```
Bitflow Installation successful
Additional manual configuration required
See the 'BitflowManualConfig.txt' in the 'doc' folder

Andor Installation successful
See the 'doc' directory for further information.
```

- For CL cameras: To finish the installation, perform the manual steps described in the BitflowManualConfig.txt document. These steps are required to complete the configuration of the cameralink card.
- For USB3 cameras: To finish the installation, perform the manual steps described in the USBManualConfig.txt document.
- For CXP cameras: To finish the installation, perform the following steps:
  - The following lines should be added to your ~/.bashrc file or equivalent:

```
$ export LD_LIBRARY_PATH="/usr/local/lib:$LD_LIBRARY_PATH"
$ export BITFLOW_INSTALL_DIRS="/usr/local/bf"
```

- The following lines should be added to your /etc/rc.local file or equivalent:

```
$ sudo /sbin/modprobe v4l2-common
$ sudo /sbin/modprobe videodev
$ sudo /sbin/insmod /usr/local/mod/bitflow.ko fwDelay1=200 customFlags=1
  cxpHWmap=0x37FF83
$ sudo chmod a+rw /dev/video*
```

- Ensure the *rc.local* file will be executed on start-up:

```
$ sudo chmod +x /etc/rc.local
```

- Reboot your PC or source your ~/.bashrc file to trigger the changes to /etc/rc.local

1.6. GETTING STARTED

This section will demonstrate how to create a basic SDK3 application that will test the software & hardware installation, and will help to confirm communication between PC and camera.

1.6.1. WINDOWS GETTING STARTED

Running the examples that came with the installation

In the installation directory there is an examples directory. In that directory there are two further directories, ‘acquisition’ and ‘serialnumber’. These directories contain the executables and the required DLLs to initialise and communicate with the sCMOS camera.

- The **acquisition** example will initialise and take a single acquisition with the camera and display the counts of the first 20 pixels.
- The **serial number** example will initialise and print out the serial number of the camera.

Creating your own applications

With this installation you can create an application with an Embarcadero or Microsoft compatible compiler. Perform the following steps to create your application.

1. Create a simple console application with either Embarcadero C++ Builder or Microsoft Visual Studio.
2. Add the SDK3 installation directory to the include path for the project. E.g. C:\Program Files\Andor SDK3
3. Add the appropriate library from the SDK3 installation directory to your project.
    - atcore.lib for the Embarcadero compiler
    - atcorem.lib for the Microsoft compiler
4. Copy all the DLL’s from the SDK3 installation directory to the directory that the executable is going to run from.
5. Type or copy the code shown below into your projects main file.
6. Compile and run the program. The program should initialise the first camera found and display it’s serial number.
7. If the serial number is not displayed then follow the comments in the code listing for hints on tracking down any issues.

> ***NOTE***
> It is assumed that there is at least 1 sCMOS camera attached. It is acceptable to have no sCMOS cameras attached if using the software simulated camera (SimCam).

```
#include "atcore.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
  int i_retCode;
  i_retCode = AT_InitialiseLibrary();
  if (i_retCode != AT_SUCCESS) {
    //error condition, check atdebug.log file
  }
  AT_64 iNumberDevices = 0;
  i_returnCode = AT_GetInt(AT_HANDLE_SYSTEM, L"DeviceCount", &iNumberDevices);
  if (iNumberDevices <= 0) {
    // No cameras found, check all redistributable binaries
    // have been copied to the executable directory or are in the system path
    // and check atdebug.log file
  }
  else {
    AT_H Hndl;
    i_retCode = AT_Open(0, &Hndl);
    if (i_retCode != AT_SUCCESS) {
      //error condition - check atdebug.log
    }

    AT_WC szValue[64];
    i_retCode= AT_GetString(Hndl, L"SerialNumber", szValue, 64);
    if (i_retCode == AT_SUCCESS) {
      //The serial number of the camera is szValue
      wcout << L"The serial number is " << szValue << endl;
    }
    else {
      //Serial Number feature was not found, check the error code for information
    }
    AT_Close(Hndl);
  }
  AT_FinaliseLibrary();

  return 0;
}
```

1.6.2. LINUX GETTING STARTED

Running the examples that came with the installation
In the installation directory there is an examples directory. In that directory there are two further directories, ‘listdevices’ and ‘image’. These directories contain source and makefiles. To build the examples change into the appropriate directory and type “make”.

1. To run the listdevices example type “./listdevices” from the “listdevices” directory. You should see the following output:

```
Found 3 Devices.
Device 0 : DC-152Q-FI
Device 1 : SIMCAM CMOS
Device 2 : SIMCAM CMOS
Press any key and enter to exit.
```

2. To run the image example type “./image” from the “image” directory. A bitmap file “image.bmp” should be created in the “image” directory containing a single image acquired from the camera.

Creating your own applications
The example directories contain example source code and makefiles that show how to create your own application. Also see windows example code shown in Section 1.6.1.

1.7. MICROSOFT APPLICATION VERIFIER

It has been found that the SDK3 cannot be used with the Microsoft Application Verifier. This is due to how libusb handles disconnected or inactive devices during initialisation which, though it is later handled within libusb, causes Microsoft Application Verifier to throw an exception.

2. API (APPLICATION PROGRAM INTERFACE)

2.1. OVERVIEW

The SDK3 API can be divided into several sets of functions, each controlling a particular aspect of camera control. There are sections in the API for opening a handle to a camera, for buffer management and for accessing the features that every camera exposes. Each feature that a camera exposes to the user has a particular type that represents how that feature is controlled. The feature types are:

- Integer
- Floating Point
- Boolean
- Enumerated
- Command
- String

For example:
- Exposure Time feature: Floating Point
- Acquisition Start feature: Command

Each of these feature types, the management of multiple cameras and buffer management are described in the sections below. The character type used by the API is a 16 bit wide character defined by the AT_WC type, which is used to represent all feature names, enumerated options and string feature values.

Wide Characters

An example of converting wide character strings to char strings can be found in the appendix.

2.2. FUNCTION LISTING

```
int AT_InitialiseLibrary();
int AT_FinaliseLibrary();

int AT_Open(int DeviceIndex, AT_H* Handle);
int AT_OpenDevice(AT_WC* Device, AT_H* Handle);
int AT_Close(AT_H Hndl);

typedef int (*FeatureCallback)(AT_H Hndl, AT_WC* Feature, void* Context);
int AT_RegisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback,
void* Context);
int AT_UnregisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback,
void* Context);

int AT_IsImplemented(AT_H Hndl, AT_WC* Feature, AT_BOOL* Implemented);
int AT_IsReadOnly(AT_H Hndl, AT_WC* Feature, AT_BOOL* ReadOnly);
int AT_IsReadable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Readable);
int AT_IsWritable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Writable);

int AT_SetInt(AT_H Hndl, AT_WC* Feature, AT_64 Value);
int AT_GetInt(AT_H Hndl, AT_WC* Feature, AT_64* Value);
int AT_GetIntMax(AT_H Hndl, AT_WC* Feature, AT_64* MaxValue);
int AT_GetIntMin(AT_H Hndl, AT_WC* Feature, AT_64* MinValue);

int AT_SetFloat(AT_H Hndl, AT_WC* Feature, double Value);
int AT_GetFloat(AT_H Hndl, AT_WC* Feature, double* Value);
int AT_GetFloatMax(AT_H Hndl, AT_WC* Feature, double* MaxValue);
int AT_GetFloatMin(AT_H Hndl, AT_WC* Feature, double* MinValue);

int AT_SetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL Value);
int AT_GetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL* Value);

int AT_SetEnumIndex(AT_H Hndl, AT_WC* Feature, int Value);
int AT_SetEnumString(AT_H Hndl, AT_WC* Feature, AT_WC* String);
int AT_GetEnumIndex(AT_H Hndl, AT_WC* Feature, int* Value);
int AT_GetEnumCount(AT_H Hndl, AT_WC* Feature, int* Count);
int AT_IsEnumIndexAvailable(AT_H Hndl, AT_WC* Feature, int Index, AT_BOOL* Available);
int AT_IsEnumIndexImplemented(AT_H Hndl, AT_WC* Feature, int Index, AT_BOOL*
Implemented);
int AT_GetEnumStringByIndex(AT_H Hndl, AT_WC* Feature, int Index, AT_WC* String, int
StringLength);

int AT_Command(AT_H Hndl, AT_WC* Feature);

int AT_SetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value);
int AT_GetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value, int StringLength);
int AT_GetStringMaxLength(AT_H Hndl, AT_WC* Feature, int* MaxStringLength);

int AT_QueueBuffer(AT_H Hndl, AT_U8* Ptr, int PtrSize);
int AT_WaitBuffer(AT_H Hndl, AT_U8** Ptr, int* PtrSize, unsigned int Timeout);
int AT_Flush(AT_H Hndl);
```

2.3. API DESCRIPTION

2.3.1. LIBRARY INITIALIZATION

The first API function call made by any application using SDK3 must be:

```
AT_InitialiseLibrary()
```

This allows SDK to setup its internal data structures and to detect any cameras that are attached. AT_InitialiseLibrary takes no parameters.

Before your application closes or when you no longer wish to access the API you should call the function:

```
AT_FinaliseLibrary()
```

This cleans up any data structures held internally by SDK.

2.3.2. OPENING A CAMERA HANDLE

To access the features provided by a camera and to acquire images you must first open a handle. A camera handle, represented by the data type AT_H, is a reference to the particular camera that you wish to control and is passed as the first parameter to most other functions in the SDK. In multi-camera environments the handle becomes particularly useful as it allows cameras to be controlled simultaneously in a thread safe manner. To open a handle to a camera you should pass the index of the camera that you wish to access, to the function:

```
AT_Open(int DeviceIndex, AT_H* Handle)
```

The handle will be returned in the Handle parameter which is passed by address. To open the first camera you should pass a value of 0 to the DeviceIndex parameter, for the second camera pass a value of 1 etc.

Once you have finished with the camera it should be closed using the function:

```
AT_Close(AT_H Handle)
```

The only parameter to this function is the handle of the camera that you wish to release.

```
AT_OpenDevice(AT_WC* Device, AT_H* Handle)
```

An alternative to the AT_Open function, AT_OpenDevice can be used to open a device from a specific device library. The Device parameter should be passed a descriptor for the device as outlined below. The handle will be returned in the Handle parameter which is passed by address.

The device descriptor contains a Library and an Index property in the format
Library:[LibraryName], Index:[DeviceIndex]

So for example to open the first Zyla camera, from the atdevregcam library, the following call should be used:
AT_OpenDevice(L"Library:regcam, Index:0", &Hndl);

The library name to use for each of the supported camera types is as follows:
Neo: regcam
Zyla: regcam
Apogee: apogee
Simcam: simcam

System Handle

There are some features of the system that are not connected to a specific camera but are global properties. For example, the Device Count feature stores a count of the number of devices that are currently connected. To access these features you do not need to open a handle to a camera, instead you should use the system handle represented by the constant AT_HANDLE_SYSTEM. You do not need to retrieve this handle using the AT_Open function; it is predefined and can be used immediately after the AT_InitialiseLibrary function has completed.

These system features can also be accessed on a particular device library to allow access to, for example, the software version of a specific library. To do this you should call the API function using the system handle and prefix the system feature name with the name of the library and a forward slash. For example, to get the software version of the library, atdevregcam, used to control the Zyla camera use:

AT_GetString(AT_HANDLE_SYSTEM, L"Regcam/SoftwareVersion", …

Thread Safety

SDK3 is thread safe when accessing different devices on different threads and also when accessing device features from the same device on different threads.

2.3.3. INTEGER FEATURES

Integer features are those that can be represented by a single integer value. For example, the number of images in the sequence that you wish to acquire is represented by the Integer feature FrameCount. To set an Integer feature use the function:

```
AT_SetInt(AT_H Hndl, AT_WC* Feature, AT_64 Value)
```

The first parameter is the handle to the camera that is exposing the desired feature; the second parameter is a wide character string indicating the name of the feature that you wish to modify. The full list of feature strings is available in the Feature Reference section. The third parameter contains the value that you want to assign to the feature. The function will return a value indicating whether the function successfully applied the value. **Section 2.4 Error Codes** lists the possible error codes that can be returned from the Integer Type functions.

To get the current value for an Integer feature use the function:

```
AT_GetInt(AT_H Hndl, AT_WC* Feature, AT_64 * Value)
```

The first two parameters are the camera handle and the feature name, the same as those passed to the AT_SetInt function. The third parameter is the address of the variable into which you want to store the Integer value.

Integer features can sometimes be restricted in the range of values that they can be set to. This range of possible values can be determined by using the functions:

```
AT_GetIntMax(AT_H Hndl, AT_WC* Feature, AT_64 * MaxValue)
AT_GetIntMin(AT_H Hndl, AT_WC* Feature, AT_64 * MinValue)
```

These functions work similarly to the AT_GetInt function except that the third parameter returns either the highest allowable value or the lowest allowable value. Using these maximum and minimum values you can check which values are allowed by AT_SetInt without having to monitor its return code. This can be useful, for example, when you wish to limit the range of possible values that the user can enter in a GUI application. Note that the maximum and minimum of an Integer feature may change if other dependent features values are modified, for example, the maximum frame rate will decrease as the exposure time is increased. You can use the feature notification mechanism described in a later section to find out when this happens.

2.3.4. FLOATING POINT FEATURES

Floating Point type features work in a similar way to Integer features, in that they have a Set function and Get function and GetMin and GetMax functions. Floating Point features represent those features that are expressed with a value that contains a decimal point. As an example, the Exposure Time feature is exposed through the Floating Point functions. The functions are:

```
AT_SetFloat(AT_H Hndl, AT_WC* Feature, double Value)
AT_GetFloat(AT_H Hndl, AT_WC* Feature, double * Value)
AT_GetFloatMax(AT_H Hndl, AT_WC* Feature, double * MaxValue)
AT_GetFloatMin(AT_H Hndl, AT_WC* Feature, double * MinValue)
```

The first parameter to each of these functions is the camera handle, the second parameter is the name of the feature and the third parameter contains either the value that you wish to set or the address of a variable that will return the current value, maximum or minimum of the feature. The list of possible error codes is described in **Section 2.4 Error Codes**. Note that the maximum and minimum of a Floating Point feature may change if other dependent features values are modified, you can use the feature notification mechanism described in later section to find out when this happens.

2.3.5. BOOLEAN FEATURES

Boolean features can only be set to one of two possible values, representing the logical states true and false. True is represented by the value AT_TRUE and false by the value AT_FALSE. An example of a boolean feature is the Sensor Cooling feature which can be used to switch the cooler on the camera on or off. To change the state of a boolean feature use the function:

```
AT_SetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL Value)
```

The first parameter is a handle to the camera being used, the second parameter is the string descriptor of the feature to change and the third parameter is the value. So to enable a boolean feature pass a value of AT_TRUE, to disable a boolean feature, pass a value of AT_FALSE in the third parameter. To retrieve the current state of a boolean feature, use the function:

```
AT_GetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL* Value)
```

For this function the third parameter contains the address of the variable into which you want the state stored. A value of AT_FALSE means the feature is disabled, while a value of AT_TRUE means the feature is enabled.

2.3.6. ENUMERATED FEATURES

Enumerated features are used to represent those features that can be assigned one value from a set of possible options. For example, the triggering mode that you wish to use with the camera is set using the TriggerMode enumerated feature. The triggering mode setting can be chosen from a number of options, for example, internal, external or external start. The enumerated feature functions allow you to:

- Determine how many options are available
- Select which option you wish to use
- Retrieve a human readable representation of each option.

Enumerated options can be set either by their text value or by index, using the functions:

```
AT_SetEnumIndex(AT_H Hndl, AT_WC* Feature, int Index)
AT_SetEnumString(AT_H Hndl, AT_WC* Feature, AT_WC* String)
```

The first function changes the current item to the one that lies at the position specified by the Index parameter, where an Index of 0 is the first item. The second function lets you specify the string descriptor for the particular option that you wish to use. String Descriptors for all features are described in Section 4.3, Feature Reference. As for all feature access functions the first two parameters are the camera handle and the string descriptor of the feature that you wish to modify. The choice of which function to use will depend on your particular application and they can both be used in the same program.

Enumerated Indexes

The particular index that maps to an enumerated option may be different across SDK versions and across different cameras. To ensure best compatibility for your application you should use strings wherever possible and avoid assuming that a specific option is found at a particular index.

To find out which option is currently selected for an enumerated feature, you can use the function:

```
AT_GetEnumIndex(AT_H Hndl, AT_WC* Feature, int* Value)
```

The third parameter is the address of the variable where you want the currently selected index stored.

To find out how many options there are available for the feature, use the function:

```
AT_GetEnumCount(AT_H Hndl, AT_WC* Feature, int* Count)
```

The third parameter, on return, will contain the number of possible options. If you attempt to select an option using AT_SetEnumIndex with an index either below zero or above or equal to this count an error will be returned.

You can retrieve the string descriptor for any option by calling the function:

```
AT_GetEnumStringByIndex(AT_H Hndl, AT_WC* Feature, int Index, AT_WC* String,
           int StringLength)
```

The third parameter is the index of the option that you want to receive the descriptor for, the fourth parameter is a user allocated buffer to receive the descriptor and the fifth parameter is the length of the allocated buffer.

Enumerated Index Availability

In some situations one or more of the options listed for an enumerated feature may be either permanently or temporarily unavailable. An option may be permanently unavailable if the camera does not support this option, or temporarily unavailable if the current value of other features do not allow this option to be selected. To find out which options are available you can use the functions:

```
AT_IsEnumIndexAvailable(AT_H Hndl, AT_WC* Feature, int Index, AT_BOOL* Available)
AT_IsEnumIndexImplemented(AT_H Hndl, AT_WC* Feature, int Index,
            AT_BOOL* Implemented)
```

Both functions take the index of the option that you want to interrogate in the third parameter. The first function determines if the option is only temporarily unavailable, the second function determines if the feature is permanently unavailable. The fourth parameter returns the availability status, a value of AT_FALSE means unavailable, and a value of AT_TRUE means the option is available. If you try to select an option that is unavailable using either of the set functions then an error will be returned.

2.3.7. COMMAND FEATURES

Command features are those that represent a single action. For example, to start the camera acquiring you will use the Command feature Acquisition Start. These commands do not require any extra parameters and are simply called by passing the string descriptor of the command to the function:

```
AT_Command(AT_H Hndl, AT_WC* Feature)
```

The function call is blocking so when the function returns, the action is complete.

2.3.8. STRING FEATURES

String features are those that can be represented by 1 or more characters. An example of a String feature is the Serial Number of the camera. In many cases these features are read only but if they are writable, you can set the value using the function:

```
AT_SetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value)
```

The first parameter contains the camera handle, the second parameter is the string descriptor of the feature, and the third parameter is the character string that you want to assign to the feature.

To retrieve the value of a String feature use the function:

```
AT_GetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value, int StringLength)
```

In this case the third parameter should be a caller allocated character string that will be used to receive the string. The fourth parameter is the length of the caller allocated buffer. To determine what length of string that should be allocated to receive the string value you can use the function:

```
AT_GetStringMaxLength(AT_H Hndl, AT_WC* Feature, int* MaxStringLength)
```

The maximum length of the String feature will be returned in the third parameter.

2.3.9. BUFFER MANAGEMENT

SDK maintains two queues for each camera, which are used to manage the transfer of image data to the application. Both queues operate in a First-in-First-out (FIFO) basis and are used to store the addresses of blocks of memory allocated by the application.

Queuing
The first queue, the input queue, which is written into by the application and read from by the SDK, is used to store the memory buffers that have not yet been filled with image data. This queue is accessed using the function:

```
AT_QueueBuffer(AT_H Hndl, AT_U8 * Ptr, int PtrSize)
```

The first parameter contains the handle to the camera. The second parameter is the address of an application allocated buffer large enough to store a single image. The PtrSize parameter should contain the size of the buffer that is being queued. The required size of the buffer can be obtained by reading the value of the ImageSizeBytes integer feature. The AT_QueueBuffer function can be called multiple times with different buffers, to allow a backlog of buffers to be stored by the SDK. By doing this the SDK can be copying image data into these buffers while the application is processing previous images.

Buffer Alignment

Any buffers queued to the SDK using the AT_QueueBuffer function should have their address aligned to an 8-byte boundary. The examples shown later in this manual demonstrate how this can be done if your compiler does not do this for you automatically when creating the buffer.

Waiting

The second queue is the output queue and is used to store the application defined buffers after they have had images copied into them. In this case the SDK adds buffers to this queue which can then be retrieved by the application. This application can retrieve the processed buffers from this queue by using the function:

```
AT_WaitBuffer(AT_H Hndl, AT_U8 ** Ptr, int* PtrSize, unsigned int Timeout)
```

The AT_WaitBuffer function will retrieve the next buffer from the output queue and return the address in the second parameter; the size of the buffer will also be returned in the PtrSize parameter. As both of these parameters are outputs from the function, they are passed in by address. If there are no buffers currently in the output queue, the AT_WaitBuffer function will put the calling thread to sleep until a buffer arrives. The thread will sleep until either a buffer arrives or the time specified by the Timeout parameter expires. The Timeout parameter is specified in milliseconds and can be any value between 0 and the constant AT_INFINITE. If a value of zero is used then the function will simply test the output queue for available buffers and return immediately. If the value is AT_INFINITE, then the function will sleep indefinitely until data arrives at the output queue; any value in between will be used as a millisecond timeout for the function.

Flushing

The input and output queues are not automatically flushed when an acquisition either completes normally or is stopped prematurely. Any buffers remaining in the input queue will be used during the next acquisition and any buffers in the output queue are still available to be retrieved by the application. If you wish to clear the two queues at any time then you should call the function below. If this function is not called after an acquisition is complete, then the remaining buffers will be used the next time an acquisition is started and may lead to undefined behaviour.

```
AT_Flush(AT_H Hndl)
```

Acquisition Control

See the descriptions of the Acquisition Start and Acquisition Stop command features for information on running an acquisition on the camera.

2.3.10. FEATURE ACCESS CONTROL

The individual access rights of features can be determined using a set of functions that apply to all features, independent of their type. The four access characteristics of a feature are:

- Whether a feature is implemented by a camera.

- Whether a feature is read only.
- Whether a feature can currently be read.
- Whether a feature can currently be modified.

The first two access rights are permanent characteristics of the feature, the second two access rights may change during the running of the program. For example, if other features are modified in such a way as to affect this feature. If a feature is not implemented by a camera then any attempt to access that feature will return the error code `AT_ERR_NOTIMPLEMENTED`. Any attempt to modify a read only feature will result in the error code `AT_ERR_READONLY`. If a feature cannot be currently read then any attempt to get the current value will return the `AT_ERR_NOTREADABLE` error code and any attempt to modify a value that cannot currently be written to will return the error code `AT_ERR_NOTWRITABLE`.

To determine if a feature has been implemented use the function:

```
AT_IsImplemented(AT_H Hndl, AT_WC* Feature, AT_BOOL* Implemented)
```

The first two parameters are, as usual, the handle to the camera and the string descriptor of the feature. The third parameter is an output parameter which returns with a value indicating whether the feature is implemented or not. If Implemented contains the value AT_TRUE, on return from the function then the feature is implemented, if it returns with the value AT_FALSE, then the feature is not implemented.

To determine if a feature is read only, use the function:

```
AT_IsReadOnly(AT_H Hndl, AT_WC* Feature, AT_BOOL* ReadOnly)
```

This function works in a similar way to AT_IsImplemented, that is, if the ReadOnly parameter returns with the value AT_TRUE then the feature is read only, a value of AT_FALSE indicates that it can be modified. SerialNumber is an example of a feature that is read only.

To determine if a feature is currently readable, use the function:

```
AT_IsReadable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Readable)
```

The Readable parameter indicates whether the feature is currently readable in the same manner as the ReadOnly parameter to the function AT_IsReadOnly.

To determine if a feature is currently writable use the function:

```
AT_IsWritable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Writable)
```

An example of the use of this function is to allow a GUI application to disable access to features when they cannot be modified, for example, whilst an acquisition is running.

2.3.11. FEATURE NOTIFICATIONS

Sometimes a feature may change its value or its other characteristics, not as a direct result of the user modifying the feature, but indirectly through modification of a separate feature. For example, if the Trigger Mode feature is set to External trigger, then the Frame Rate feature’s writable access characteristic will be disabled (see Feature Access Control above), as the frame rate is now controlled by the rate at which the external trigger is applied, and not by the application setting.

To allow the application to receive notification when this type of indirect change occurs, there are functions provided in the API that allow the application to create a callback function and attach it to a feature. Whenever the feature changes in any way, this callback will be triggered, allowing the application to carry out any actions required to respond to the change. For example, if an application provides a GUI interface that allows users to modify features, then the callback can update the GUI with any changes. This facilitates use of the Observer design pattern in your application. The callback will also be triggered if the feature is modified directly by the application.

The definition of the callback function implemented by the application should be in the format:

```
int AT_EXP_CONV MyCallback(AT_H Hndl, const AT_WC* Feature, void* Context)
{
  // Perform action
}
```

There are three parameters sent to the function that allow the application to determine the reason for the call-back. The first parameter indicates which camera caused the call-back. By using this parameter you can make use of the same call-back function for multiple cameras. The second parameter holds the string descriptor of the feature that has been modified in some way, and allows the same call-back function to be used with multiple features. The final parameter is an application defined context parameter that was passed in as a parameter at the time that the call-back function was registered. The Context parameter is not parsed in any way by the SDK and can be used to store any information that the application wishes.

Note that the AT_EXP_CONV modifier must be present and ensures that the correct calling convention is used by the SDK.

To register the call-back function, use the function:

```
AT_RegisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback, void*
Context)
```

The first parameter is the camera handle, the second parameter is the string descriptor of the feature that you wish to receive notifications for. The third parameter is your call-back function that you have defined as described above and the fourth parameter in the Context parameter that will be passed to the call-back each time it is called. Whenever the application no longer requires notifications for a particular feature, it should release the call-back by calling the function:

```
AT_UnregisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback,
void* Context)
```

The same parameters should be passed to this function as were passed to the AT_RegisterFeatureCallback.

You need to register a call-back individually for each feature that you are interested in, but the same call-back function can be used for all or some features, or, a separate call-back function can be provided for each feature.

Notes on implementing call-backs

A call-back should complete any work required in the minimal amount of time as it holds up the thread that caused the call-back. If possible the application should delegate any work to a separate application thread if the action will take a significant amount of time.

The call-back function should not attempt to modify the value of any feature as this can cause lockup.

2.4. ERROR CODES

Find below the available return codes and their values for each feature type and the buffer control functions.

Device Connection	Description
AT_SUCCESS (0)	Function call has been successful
AT_ERR_NOTINITIALISED (1)	Function called with an uninitialized handle
AT_ERR_CONNECTION (10)	Error connecting to or disconnecting from hardware
AT_ERR_INVALIDHANDLE (12)	Invalid device handle passed to function
AT_ERR_NULL_HANDLE (21)	Null device handle passed to function
AT_ERR_NOMEMORY (37)	No memory has been allocated for the current action
AT_ERR_DEVICEINUSE (38)	Function failed to connect to a device because it is already being used

String Feature	Description
AT_SUCCESS (0)	Function call has been successful
AT_ERR_NOTIMPLEMENTED (2)	Feature has not been implemented for the chosen camera
AT_ERR_READONLY (3)	Feature is read only
AT_ERR_NOTWRITABLE (5)	Feature is currently not writable
AT_ERR_NOTREADABLE (4)	Feature is currently not readable
AT_ERR_EXCEEDEDMAXSTRINGLENGTH (9)	String value provided exceeds the maximum allowed length
AT_ERR_NULL_FEATURE (20)	NULL feature name passed to function
AT_ERR_NULL_READABLE_VAR (23)	Readable not set
AT_ERR_NULL_WRITABLE_VAR (25)	Writable not set
AT_ERR_NULL_ISAVAILABLE_VAR (31)	Available not set
AT_ERR_NULL_VALUE (28)	NULL value returned from function
AT_ERR_NULL_STRING (29)	NULL string returned from function
AT_ERR_NULL_MAXSTRINGLENGTH (32)	Max string length is NULL
AT_ERR_INVALIDHANDLE (12)	Invalid device handle passed to function
AT_ERR_NOMEMORY (37)	No memory has been allocated for the current action
AT_ERR_COMM (17)	An error has occurred while communicating with hardware

Integer Feature	Description
AT_SUCCESS (0)	Function call has been successful
AT_ERR_OUTOFRANGE (6)	Value is outside the maximum and minimum limits
AT_ERR_NOTIMPLEMENTED (2)	Feature has not been implemented for the chosen camera
AT_ERR_READONLY (3)	Feature is read only
AT_ERR_NOTWRITABLE (5)	Feature is currently not writable
AT_ERR_NOTREADABLE (4)	Feature is currently not readable
AT_ERR_NULL_FEATURE (20)	NULL feature name passed to function
AT_ERR_NULL_READABLE_VAR (23)	Readable not set
AT_ERR_NULL_WRITABLE_VAR (25)	Writable not set
AT_ERR_NULL_ISAVAILABLE_VAR (31)	Available not set
AT_ERR_NULL_VALUE (28)	NULL value returned from function
AT_ERR_NULL_MINVALUE (26)	NULL min value
AT_ERR_NULL_MAXVALUE (27)	NULL max value
AT_ERR_INVALIDHANDLE (12)	Invalid device handle passed to function
AT_ERR_NOMEMORY (37)	No memory has been allocated for the current action
AT_ERR_COMM (17)	An error has occurred while communicating with hardware

Float Feature	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_OUTOFRANGE (6)`	Value is outside the maximum and minimum limits
`AT_ERR_NOTIMPLEMENTED (2)`	Feature has not been implemented for the chosen camera
`AT_ERR_READONLY (3)`	Feature is read only
`AT_ERR_NOTWRITABLE (5)`	Feature is currently not writable
`AT_ERR_NOTREADABLE (4)`	Feature is currently not readable
`AT_ERR_NULL_FEATURE (20)`	NULL feature name passed to function
`AT_ERR_NULL_READABLE_VAR (23)`	Readable not set
`AT_ERR_NULL_WRITABLE_VAR (25)`	Writable not set
`AT_ERR_NULL_ISAVAILABLE_VAR (31)`	Available not set
`AT_ERR_NULL_VALUE (28)`	NULL value returned from function
`AT_ERR_NULL_MINVALUE (26)`	NULL min value
`AT_ERR_NULL_MAXVALUE (27)`	NULL max value
`AT_ERR_INVALIDHANDLE (12)`	Invalid device handle passed to function
`AT_ERR_NOMEMORY (37)`	No memory has been allocated for the current action
`AT_ERR_COMM (17)`	An error has occurred while communicating with hardware

Boolean Feature	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_OUTOFRANGE (6)`	The value passed to the function was not a valid boolean value i.e. 0 or 1.
`AT_ERR_NOTIMPLEMENTED (2)`	Feature has not been implemented for the chosen camera
`AT_ERR_READONLY (3)`	Feature is read only
`AT_ERR_NOTWRITABLE (5)`	Feature is currently not writable
`AT_ERR_NOTREADABLE (4)`	Feature is currently not readable
`AT_ERR_NULL_FEATURE (20)`	NULL feature name passed to function
`AT_ERR_NULL_READABLE_VAR (23)`	Readable not set
`AT_ERR_NULL_WRITABLE_VAR (25)`	Writable not set
`AT_ERR_NULL_ISAVAILABLE_VAR (31)`	Available not set
`AT_ERR_NULL_VALUE (28)`	NULL value returned from function
`AT_ERR_INVALIDHANDLE (12)`	Invalid device handle passed to function
`AT_ERR_NOMEMORY (37)`	No memory has been allocated for the current action
`AT_ERR_COMM (17)`	An error has occurred while communicating with hardware

Enumerated Feature	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_OUTOFRANGE (6)`	The index passed to the function was either less than zero or greater than or equal to the number of implemented options.
`AT_ERR_NOTIMPLEMENTED (2)`	Feature has not been implemented for the chosen camera
`AT_ERR_READONLY (3)`	Feature is read only
`AT_ERR_NOTWRITABLE (5)`	Feature is currently not writable
`AT_ERR_NOTREADABLE (4)`	Feature is currently not readable
`AT_ERR_INDEXNOTAVAILABLE (7)`	Index is currently not available
`AT_ERR_INDEXNOTIMPLEMENTED (8)`	Index is not implemented for the chosen camera
`AT_ERR_STRINGNOTAVAILABLE(18)`	Index / String is not available
`AT_ERR_STRINGNOTIMPLEMENTED (19)`	Index / String is not implemented for the chosen camera
`AT_ERR_NULL_FEATURE (20)`	NULL feature name passed to function
`AT_ERR_NULL_READABLE_VAR (23)`	Readable not set
`AT_ERR_NULL_WRITABLE_VAR (25)`	Writable not set
`AT_ERR_NULL_ISAVAILABLE_VAR (31)`	Available not set
`AT_ERR_NULL_VALUE (28)`	NULL value returned from function
`AT_ERR_NULL_COUNT_VAR (30)`	NULL feature count
`AT_ERR_NULL_IMPLEMENTED_VAR (22)`	Feature not implemented
`AT_ERR_INVALIDHANDLE (12)`	Invalid device handle passed to function
`AT_ERR_NOMEMORY (37)`	No memory has been allocated for the current action
`AT_ERR_COMM (17)`	An error has occurred while communicating with hardware

Command Feature	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_NOTIMPLEMENTED (2)`	Feature has not been implemented for the chosen camera
`AT_ERR_NOTWRITABLE (5)`	Feature is currently not executable
`AT_ERR_NULL_FEATURE (20)`	NULL feature name passed to function
`AT_ERR_NULL_READABLE_VAR (23)`	Readable not set
`AT_ERR_NULL_WRITABLE_VAR (25)`	Writable not set
`AT_ERR_NULL_ISAVAILABLE_VAR (31)`	Available not set
`AT_ERR_NULL_VALUE (28)`	NULL value returned from function
`AT_ERR_INVALIDHANDLE (12)`	Invalid device handle passed to function
`AT_ERR_NOMEMORY (37)`	No memory has been allocated for the current action
`AT_ERR_COMM (17)`	An error has occurred while communicating with hardware

Buffer Control	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_TIMEDOUT (13)`	The AT_WaitBuffer function timed out while waiting for data arrive in output queue
`AT_ERR_BUFFERFULL (14)`	The input queue has reached its capacity
`AT_ERR_INVALIDSIZE (15)`	The size of a queued buffer did not match the frame size
`AT_ERR_INVALIDALIGNMENT (16)`	A queued buffer was not aligned on an 8-byte boundary
`AT_ERR_HARDWARE_OVERFLOW (100)`	The software was not able to retrieve data from the card or camera fast enough to avoid the internal hardware buffer bursting.
`AT_ERR_NOMEMORY (37)`	No memory has been allocated for the current action
`AT_ERR_NODATA (11)`	No Internal Event or Internal Error
`AT_ERR_COMM (17)`	An error has occurred while communicating with hardware
`AT_ERR_NULL_QUEUE_PTR (34)`	Pointer to queue is NULL
`AT_ERR_NULL_WAIT_PTR (35)`	Wait pointer is NULL
`AT_ERR_NULL_PTRSIZE (36)`	Pointer size is NULL

Feature Callback	Description
`AT_SUCCESS (0)`	Function call has been successful
`AT_ERR_NULL_FEATURE (20)`	NULL feature name passed to function
`AT_ERR_NULL_EVCALLBACK (33)`	EvCallBack parameter is NULL
`AT_ERR_NOTIMPLEMENTED (2)`	Feature has not been implemented for the chosen camera
`AT_ERR_INVALIDHANDLE (12)`	The size of a queued buffer did not match the frame size

3. FUNCTION REFERENCE

3.1. FUNCTION LISTING

This section provides a description of various reference functions available in SDK3.

3.1.1. AT_OPEN

```
int AT_Open(int DeviceIndex, AT_H* Handle)
```

Description

This function is used to open up a handle to a particular camera. The DeviceIndex parameter indicates the index of the camera that you wish to open and the handle to the camera is returned in the Handle parameter. This Handle parameter must be passed as the first parameter to all other functions to access the features or to acquire data from the camera.

3.1.2. AT_OPENDEVICE

```
int AT_OpenDevice(AT_WC* Device, AT_H* Handle)
```

Description

This function is used to open up a handle to a particular device and can be used instead of the AT_Open function. The Device parameter indicates the descriptor for the device that you wish to open and the handle to the device is returned in the Handle parameter. This Handle parameter must be passed as the first parameter to all other functions to access the features or to acquire data from the device. The format of the Device descriptor is:

Library:[LibraryName], Index:[DeviceIndex]

When specifying the library name do not include the atdev prefix or the library file extension.

3.1.3. AT_CLOSE

```
int AT_Close(AT_H Handle)
```

Description

This function is used to close a previously opened handle to a camera. The Handle parameter is the handle that was returned from the AT_Open function. The function should be called when you no longer wish to access the camera from your application usually at shutdown.

3.1.4. AT_ISIMPLEMENTED

```
int AT_IsImplemented(AT_H Hndl, AT_WC* Feature, AT_BOOL* Implemented)
```

Description

This function can be used to determine whether the camera has implemented the feature specified by the Feature parameter. On return the Implemented parameter will contain the value AT_FALSE or AT_TRUE. In the case that the feature is implemented the value of Implemented will be AT_TRUE, otherwise it will be AT_FALSE.

3.1.5. AT_ISREADONLY

```
int AT_IsReadOnly(AT_H Hndl, AT_WC* Feature, AT_BOOL* ReadOnly)
```

Description
This function can be used to determine whether the feature specified by the Feature parameter can be modified. On return the ReadOnly parameter will contain the value AT_FALSE or AT_TRUE. In the case that the feature cannot be modified the value of ReadOnly will be AT_TRUE, otherwise it will be AT_FALSE.

3.1.6. AT_ISWRITABLE

```
int AT_IsWritable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Writable)
```

Description
This function can be used to determine whether the feature specified by the Feature parameter can currently be modified. On return the Writable parameter will contain the value AT_FALSE or AT_TRUE. In the case that the feature is currently writable the value of Writable will be AT_TRUE, otherwise it will be AT_FALSE. This function differs from the AT_IsReadOnly function in that a feature that is not writable may only be temporarily unavailable for modification because of the values of other features, whereas a feature that is read only is permanently un-modifiable.

3.1.7. AT_ISREADABLE

```
int AT_IsReadable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Readable)
```

Description
This function can be used to determine whether the feature specified by the Feature parameter can currently be read. On return the Readable parameter will contain the value AT_FALSE or AT_TRUE. In the case that the feature is currently readable the value of Readable will be AT_TRUE, otherwise it will be AT_FALSE. A feature may become unavailable for reading based on the value of other features.

3.1.8. AT_REGISTERFEATURECALLBACK

```
int AT_RegisterFeatureCallback(AT_H Hndl, AT_WC* Feature,
       FeatureCallback EvCallback, void* Context)
```

Description
To retrieve a notification each time the value or other properties of a feature changes you can use this function to register a callback function. The Feature that you wish to receive notifications for is passed into the function along with the function that you wish to get called. The fourth parameter is a caller defined parameter that can be used to provide contextual information when the callback is called. The callback function should have the signature shown below.

```
int AT_EXP_CONV MyFunction(AT_H Hndl, AT_WC* Feature, void* Context)
```

When called, the Feature that caused the callback is returned which allows you to use a single callback function to handle multiple features. The context parameter is the same as that used when registering the callback and is sent unmodified. As soon as this callback is registered a single callback will be made immediately to allow the callback handling code to perform any Initialisation code to set up monitoring of the feature.

3.1.9. AT_UNREGISTERFEATURECALLBACK

```
int AT_UnregisterFeatureCallback(AT_H Hndl, AT_WC* Feature,
                  FeatureCallback EvCallback, void* Context)
```

Description
This function is used to un-register a callback function previously registered using AT_RegisterFeatureCallback. The same parameters that were passed to the register function should be passed to this unregister function. Once this function is called, no more callbacks will be sent to this callback function for the specified Feature.

3.1.10. AT_INITIALISELIBRARY

```
int AT_InitialiseLibrary()
```

Description
This function is used to prepare the SDK internal structures for use and must be called before any other SDK functions have been called.

3.1.11. AT_FINALISELIBRARY

```
int AT_FinaliseLibrary()
```

Description
This function will free up any resources used by the SDK and should be called whenever the program no longer needs to use any SDK functions. AT_InitialiseLibrary may be called again later by the same process if camera control is again required.

3.1.12. AT_SETINT

```
int AT_SetInt(AT_H Hndl, AT_WC* Feature, AT_64 Value)
```

Description
This function will modify the value of the specified feature, if the feature is of integer type. The function will return an error if the feature is read only or currently not writable or if the feature is either not an integer feature or is not implemented by the camera.

3.1.13. AT_GETINT

```
int AT_GetInt(AT_H Hndl, AT_WC* Feature, AT_64 * Value)
```

Description
This function will return the current value for the specified feature. The function will return an error if the feature is currently not readable or if the specified feature is either not an integer feature or is not implemented by the camera.

3.1.14. AT_GETINTMAX

```
int AT_GetIntMax(AT_H Hndl, AT_WC* Feature, AT_64 * MaxValue)
```

Description
This function will return the maximum allowable value for the specified integer type feature.

3.1.15. AT_GETINTMIN

```
int AT_GetIntMin(AT_H Hndl, AT_WC* Feature, AT_64 * MinValue)
```

Description
This function will return the minimum allowable value for the specified integer type feature.

3.1.16. AT_SETFLOAT

```
int AT_SetFloat(AT_H Hndl, AT_WC* Feature, double Value)
```

Description
This function will modify the value of the specified feature, if the feature is of float type. The function will return an error if the feature is read only or currently not writable or if the feature is either not a float type feature or is not implemented by the camera.

3.1.17. AT_GETFLOAT

```
int AT_GetFloat(AT_H Hndl, AT_WC* Feature, double * Value)
```

Description
This function will return the current value for the specified feature. The function will return an error if the feature is currently not readable or if the specified feature is either not a float type feature or is not implemented by the camera.

3.1.18. AT_GETFLOATMAX

```
int AT_GetFloatMax(AT_H Hndl, AT_WC* Feature, double * MaxValue)
```

Description
This function will return the maximum allowable value for the specified float type feature.

3.1.19. AT_GETFLOATMIN

```
int AT_GetIntMin(AT_H Hndl, AT_WC* Feature, double * MinValue)
```

Description
This function will return the minimum allowable value for the specified float type feature.

3.1.20. AT_SETBOOL

```
int AT_SetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL Value)
```

Description
This function will set the value of the specified boolean feature. A value of AT_FALSE indicates false and a value of AT_TRUE indicates true. An error will be returned if the feature is read only, currently not writable, not a boolean feature or is not implemented by the camera.

3.1.21. AT_GETBOOL

```
int AT_GetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL * Value)
```

Description
This function will return the current value of the specified boolean feature. If a value of AT_FALSE is returned then the feature is currently set to false. If a value of AT_TRUE is returned then the feature is currently set to true. An error will be returned if the feature is currently not readable, not a boolean feature or is not implemented by the camera.

3.1.22. AT_COMMAND

```
int AT_Command(AT_H Hndl, AT_WC* Feature)
```

Description
This function will trigger the specified command feature to execute. An error will be returned if the feature is currently not writable, not a command feature or is not implemented by the camera.

3.1.23. AT_SETSTRING

```
int AT_SetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value)
```

Description
This function will set the value of the specified string feature. The string should be null terminated. An error will be returned if the feature is read only, currently not writable, not a string feature or is not implemented by the camera.

3.1.24. AT_GETSTRING

```
int AT_GetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value, int StringLength)
```

Description
This function will return the current value of the specified string feature. The length of the string in which you want the value returned must be provided in the fourth parameter and the string should include enough space for the null terminator. An error will be returned if the feature is currently not readable, not a string feature or is not implemented by the camera.

3.1.25. AT_GETSTRINGMAXLENGTH

```
int AT_GetStringMaxLength(AT_H Hndl, AT_WC* Feature, int* MaxStringLength)
```

Description
This function will return the maximum length of the specified string feature. This value can be used to determine what size of string to allocate when retrieving the value of the feature using the AT_GetString function.

3.1.26. AT_SETENUMINDEX

```
int AT_SetEnumIndex(AT_H Hndl, AT_WC* Feature, int Value)
```

Description
This function sets the currently selected index of the specified enumerated feature. The index is zero based and should be in the range 0 to Count-1, where Count has been retrieved using the AT_GetEnumCount function. An error will be returned if the feature is read only, currently not writable, the index is outside the allowed range, not an enumerated feature, or the feature is not implemented by the camera. In some cases an index within the range may not be allowed if its availability depends on other features values, in this case an error will be returned if this index is applied.

3.1.27. AT_SETENUMSTRING

```
int AT_SetEnumString(AT_H Hndl, AT_WC* Feature, AT_WC* String)
```

Description
This function directly sets the current value of the specified enumerated feature. The String parameter must be one of the allowed values for the feature and must be currently available.

3.1.28. AT_GETENUMINDEX

```
int AT_GetEnumIndex(AT_H Hndl, AT_WC* Feature, int* Value)
```

Description
This function retrieves the currently selected index of the specified enumerated feature. The function will return an error if the feature is currently not readable or if the specified feature is either not an enumerated type feature or is not implemented by the camera.

3.1.29. AT_GETENUMCOUNT

```
int AT_GetEnumCount(AT_H Hndl, AT_WC* Feature, int* Count)
```

Description
This function returns the number of indexes that the specified enumerated feature can be set to.

3.1.30. AT_GETENUMSTRINGBYINDEX

```
int AT_GetEnumStringByIndex(AT_H Hndl, AT_WC* Feature, int Index, AT_WC* String,
                              int StringLength)
```

Description

This function returns the text representation of the specified enumerated feature index. The index should be in the range 0... Count-1, where Count has been retrieved using the AT_GetEnumCount function. The length of the String parameter should be passed in to the fifth parameter.

3.1.31. AT_ISENUMINDEXAVAILABLE

```
int AT_IsEnumIndexAvailable(AT_H Hndl, AT_WC* Feature, int Index,
              AT_BOOL* Available)
```

Description

This function indicates whether the specified enumerated feature index can currently be selected. The availability of enumerated options may depend on the value of other features.

3.1.32. AT_ISENUMINDEXIMPLEMENTED

```
int AT_IsEnumIndexImplemented(AT_H Hndl, AT_WC* Feature, int Index,
                AT_BOOL* Implemented)
```

Description

This function indicates whether the camera supports the specified enumerated feature index. For consistency across the camera range, some enumerated features options may appear in the list even when they are not supported, this function will let you filter out these options.

3.1.33. AT_QUEUEBUFFER

```
int AT_QueueBuffer(AT_H Hndl, AT_U8* Ptr, int PtrSize)
```

Description

This function configures the area of memory into which acquired images will be stored. You can call this function multiple times to set up storage for consecutive images in a series. The order in which buffers are queued is the order in which they will be used on a first in, first out (FIFO) basis. The PtrSize parameter should be equal to the size of an individual image in number of bytes. This function may be called before the acquisition starts, after the acquisition starts or a combination of the two. Any buffers queued using this function should not be modified or deallocated by the calling application until they are either returned from the AT_WaitBuffer function, or the AT_Flush function is called.

3.1.34. AT_WAITBUFFER

```
int AT_WaitBuffer(AT_H Hndl, AT_U8** Ptr, int* PtrSize, unsigned int Timeout)
```

Description

This function is used to receive notification whenever a previously queued image buffer contains data. The address of the buffer that is now available is returned in the Ptr parameter. The PtrSize parameter will return with the size of the returned image buffer. The Timeout parameter can be specified to indicate how long in milliseconds you wish to wait for the next available image. The function will put the calling thread to sleep until either an image becomes available or the Timeout elapses.

3.1.35. AT_FLUSH

```
int AT_Flush(AT_H Hndl)
```

Description

This function is used to flush out any remaining buffers that have been queued using the AT_QueueBuffer function. It should always be called after the AT_Command(L"AcquisitionStop") function has been called. If this function is not called after an acquisition is complete, then the remaining buffers will be used the next time an acquisition is started, and may lead to undefined behaviour.

4. CAMERA FEATURES

4.1. CAMERA SUPPORT

SDK3 currently supports the Andor sCMOS and Apogee families of cameras. The features that are available for these cameras are outlined below. There is also a software module called SimCam that simulates limited functionality of a camera. The SimCam module can be useful to prototype an application where availability of a real camera may be limited. To use SimCam, you should copy the atdevsimcam.dll file into your application directory. On initialisation of the SDK there will be two SimCam cameras available. The camera platforms that support each feature are listed in the table column below.

4.2. FEATURE AVAILIBILITY

Key
CL – CameraLink only
Dep – Deprecated
WC – Water Cooled only

Feature	System	Neo	Zyla	iStar	Sona	Marana	Balor	Apogee	SimCam
AccumulateCount		✓	✓	✓					
AcquiredCount								✓	
AcquisitionStart		✓	✓	✓	✓	✓	✓		✓
AcquisitionStop		✓	✓	✓	✓	✓	✓		✓
AlternatingReadoutDirection			✓						
AOIBinning		✓	✓	✓	✓	✓	✓		
AOIHBin			✓	✓	✓	✓	✓	✓	✓
AOIHeight		✓	✓	✓	✓	✓	✓	✓	✓
AOILayout			✓	✓	✓	✓	✓	✓	
AOILeft		✓	✓	✓	✓	✓	✓	✓	✓
AOIStride		✓	✓	✓	✓	✓	✓		
AOITop		✓	✓	✓	✓	✓	✓	✓	✓
AOIVBin			✓	✓	✓	✓	✓	✓	✓
AOIWidth		✓	✓	✓	✓	✓	✓	✓	✓
AuxiliaryOutSource		✓	✓		✓	✓	✓		
AuxOutSourceTwo			✓	✓	✓	✓	✓		
BackoffTemperatureOffset								✓	
Baseline		✓	✓	✓	✓	✓	✓		
BitDepth		✓	✓	✓	✓	✓	✓	✓	
BufferOverflowEvent		✓	✓	✓	✓	✓			
BytesPerPixel		✓	✓	✓	✓	✓	✓		
CameraAcquiring		✓	✓	✓	✓	✓	✓		✓
CameraFamily					✓	✓	✓	✓	
CameraMemory								✓	
CameraModel		✓	✓	✓	✓	✓	✓		✓
CameraName		✓	✓	✓	✓	✓	✓	✓	
CameraPresent					✓	✓	✓		
ColourFilter								✓	
ControllerID		CL	CL						
CoolerPower								✓	
CycleMode		✓	✓	✓	✓	✓	✓		✓
DDGIOCEnable				✓					
DDGIOCNumberOfPulses				✓					
DDGIOCPeriod				✓					
DDGOutputDelay				✓					
DDGOutputEnable				✓					
DDGOutputStepEnable				✓					
DDGOpticalWidthEnable				✓					
DDGOutputPolarity				✓					
DDGOutputSelector				✓					
DDGOutputWidth				✓					

Feature	System	Neo	Zyla	iStar	Sona	Marana	Balor	Apogee	SimCam
DDGStepCount				✓					
DDGStepDelayCoefficientA				✓					
DDGStepDelayCoefficientB				✓					
DDGStepDelayMode				✓					
DDGStepEnabled				✓					
DDGStepUploadProgress				✓					
DDGStepUploadRequired				✓					
DDGStepWidthCoefficientA				✓					
DDGStepWidthCoefficientB				✓					
DDGStepWidthMode				✓					
DDGStepUploadModeValues				✓					
DDR2Type								✓	
DeviceCount	✓								
DeviceVideoIndex		CL	CL						
DisableShutter								✓	
DriverVersion								✓	
ElectronicShutteringMode		✓	✓	✓	✓	✓	✓		✓
EventEnable		✓	✓	✓	✓	✓			
EventsMissedEvent		✓	✓	✓	✓	✓			
EventSelector		✓	✓	✓	✓	✓			
ExposedPixelHeight			✓	✓					
ExposureTime		✓	✓	✓	✓	✓	✓		✓
ExposureEndEvent		✓	✓	✓	✓	✓			
ExposureStartEvent		✓	✓	✓	✓	✓			
ExternalIOReadout								✓	
ExternalTriggerDelay			✓	✓	✓	✓	✓		
FanSpeed		✓	✓	✓	✓	✓	✓	✓	✓
FastAOIFrameRateEnable		✓	✓	✓					
FirmwareVersion		✓	✓	✓	✓	✓	✓	✓	
FrameGenFixedPixelValue					✓	✓	✓		
FrameGenMode					✓	✓	✓		
ForceShutterOpen								✓	
FrameCount		✓	✓	✓	✓	✓	✓	✓	✓
FrameInterval								✓	
FrameIntervalTiming								✓	
FrameRate		✓	✓	✓	✓	✓	✓	✓	✓
FullAOIControl		✓	✓	✓	✓	✓	✓		
GateMode				✓					
HeatSinkTemperature								✓	
I2CAddress				✓					
I2CByte				✓					
I2CByteCount				✓					
I2CByteSelector				✓					
I2CRead				✓					
I2CWrite				✓					
ImageSizeBytes		✓	✓	✓	✓	✓	✓		✓
InputVoltage								✓	
InsertionDelay				✓					
InterfaceType		✓	✓	✓	✓	✓	✓	✓	
IOControl								✓	
IODirection								✓	
IOState								✓	
IOInvert		✓	✓	✓	✓	✓	✓		
IOSelector		✓	✓	✓	✓	✓	✓	✓	
IRPreFlashEnable								✓	
KeepCleanEnable								✓	
KeepCleanPostExposure Enable								✓	
LineScanSpeed			✓						
LogLevel	✓								
LUTIndex									

Feature	System	Neo	Zyla	iStar	Sona	Marana	Balor	Apogee	SimCam
LUTValue									
MaxInterfaceTransferRate		✓	✓	✓	✓	✓	✓		
MCPGain				✓					
MCPIntelligate				✓					
MCPVoltage				✓					
MetadataEnable		✓	✓	✓	✓	✓			
MetadataFrame		✓	✓	✓	✓	✓			
MetadataTimestamp		✓	✓	✓	✓	✓			
MicrocodeVersion								✓	
MultitrackBinned			✓	✓	✓	✓	✓		
MultitrackCount			✓	✓	✓	✓	✓		
MultitrackEnd			✓	✓	✓	✓	✓		
MultitrackSelector			✓	✓	✓	✓	✓		
MultitrackStart			✓	✓	✓	✓	✓		
Overlap		✓	✓	✓	✓	✓	✓	✓	
PIVEnable				✓					
PixelCorrection									✓
PixelEncoding		✓	✓	✓	✓	✓	✓		✓
PixelHeight		✓	✓	✓	✓	✓	✓	✓	✓
PixelReadoutRate		✓	✓	✓	✓	✓	✓	✓	✓
PixelWidth		✓	✓	✓	✓	✓	✓	✓	✓
PortSelector								✓	
PreAmpGain		Dep							✓
PreAmpGainChannel		Dep							✓
PreAmpGainControl		Dep							
PreAmpGainValue								✓	
PreAmpGainSelector		Dep							✓
PreAmpOffsetValue								✓	
PreTriggerEnable				✓					
ReadoutTime		✓	✓	✓	✓	✓	✓		
RollingShutterGlobalClear			✓						
RowNExposureEndEvent		✓	✓		✓	✓			
RowNExposureStartEvent		✓	✓		✓	✓			
RowReadTime			✓		✓	✓	✓		
ScanSpeedControlEnable			✓						
SensorCooling		✓	✓	✓	✓	✓	✓	✓	✓
SensorHeight		✓	✓	✓	✓	✓	✓	✓	✓
SensorModel								✓	
SensorReadoutMode			✓						
SensorType								✓	
SensorTemperature		✓	✓	✓	✓	✓	✓	✓	✓
SensorWidth		✓	✓	✓	✓	✓	✓	✓	✓
SerialNumber		✓	✓	✓	✓	✓	✓		✓
ShutterAmpControl								✓	
ShutterMode			✓	✓	✓	✓	✓	✓	
ShutterOutputMode			✓	Dep					
ShutterState								✓	
ShutterStrobePeriod								✓	
ShutterStrobePosition								✓	
ShutterTransferTime			✓	✓	✓	✓	✓		
SimplePreAmpGainControl		✓	✓	✓	Dep	Dep	Dep		
SoftwareTrigger		✓	✓	✓	✓	✓	✓		
SoftwareVersion	✓								
SpuriousNoiseFilter		✓	✓	✓	✓	✓	✓		
StaticBlemishCorrection		✓	✓	✓					
Synchronous Triggering									✓
TargetSensor Temperature		Dep			✓	✓	✓		✓
Temperature Control		✓	WC		✓	✓	✓		

Feature	System	Neo	Zyla	iStar	Sona	Marana	Balor	Apogee	SimCam
Temperature Status		✓	✓	✓	✓	✓	✓	✓	
TimestampClock		✓	✓	✓	✓	✓			
TimestampClock Frequency		✓	✓	✓	✓	✓			
TimestampClock Reset		✓	✓	✓	✓	✓			
TransmitFrames								✓	
TriggerMode		✓	✓	✓	✓	✓	✓	✓	✓
UsbProductId								✓	
UsbDeviceId								✓	
VerticallyCentreAOI		✓	✓	✓	✓	✓	✓		

4.3. FEATURE REFERENCE

4.3.1. ZYLA

Feature	Type	Description
AccumulateCount	Integer	Sets the number of images that should be summed to obtain each image in sequence.
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AlternatingReadoutDirection	Boolean	Configures whether the sensor will be read out in alternating directions. See SensorReadDirection.
AOIBinning	Enumerated	Sets up pixel binning on the camera. Options: • **1x1** • **2x2** • **3x3** • **4x4** • **8x8** See **Section 4.7 Area of Interest.**
AOIHBin	Integer	Configures the Horizontal Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Options: • **Image** • **Multitrack**
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest**.
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest**.
AuxiliaryOutSource	Enumerated	Configures which signal appears on the auxiliary output pin. Options: • **FireRow1** • **FireRowN** • **FireAll** • **FireAny**
AuxOutSourceTwo	Enumerated	AuxOutSourceTwo is a configurable output available to the user on the D-

Feature	Type	Description
		type Options: • **ExternalShutterControl** • **FrameClock** • **RowClock** • **ExposedRowClock**
Baseline	Integer	Returns the baseline level of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. Supported Bit Depth will be dependent on the camera. Options Neo/Zyla: • **11 Bit or 12 Bit** • **16 Bit**
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Floating Point	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system. Register a callback to this feature to be notified if the camera is disconnected.
ControllerID	String	Returns a unique identifier for the camera controller device. i.e. Frame grabber over Cameralink
CoolerPower	Double	Percentage of maximum power being used by the cooler.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the 'AcquisitionStop' command is issued. Options: • **Fixed** • **Continuous**
DeviceCount	Integer	Returns the number of cameras detected.
DeviceVideoIndex	Integer	Returns the /dev/videoN number in Linux only
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • **Rolling** • **Global**
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
EventSelector	Enumerated	Selects the acquisition events you wish to enable or disable using the EventEnable feature. Options: • **ExposureEndEvent** • **ExposureStartEvent** • **RowNExposureEndEvent** • **RowNExposureStartEvent** • **EventsMissedEvent** • **BufferOverflowEvent**
ExposedPixelHeight	Integer	Configures the exposure window in pixels.
ExposureTime	Floating	The requested exposure time in seconds. *Note: In some modes the*

Feature	Type	Description
	Point	*exposure time can also be modified while the acquisition is running.*
ExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExternalTriggerDelay	Floating Point	Sets the delay time between the camera receiving an external trigger and the acquisition start.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **On**
FastAOIFrameRateEnable	Boolean	Enables faster framerates at small AOIs.
FirmwareVersion	String	Returns the camera firmware version
FrameCount	Integer	Configures the number of images to acquire in the sequence while **CycleMode** is **Fixed**. The value of FrameCount must be any value which is a multiple of AccumulateCount. This ensures the accumulation contains the correct number of frames. When this feature is unavailable then the camera does not currently support fixed length series, therefore you must explicitly abort the acquisition once you have acquired the amount of frames required.
FrameRate	Floating Point	Configures the frame rate in Hz at which each image is acquired during any acquisition sequence. This is the rate at which frames are acquired by the camera which may be different from the rate at which frames are delivered to the user. For example when AccumulateCount has a value other than 1, the apparent frame rate will decrease proportionally.
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InterfaceType	String	Returns the camera interface type. Current types: Options: • **USB3** • **CL 3 Tap** • **CL 2x5 Tap** • **CL 10 Tap**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin that you subsequently wish to configure using the *IO Invert* Feature. Options: • **Fire 1** • **Fire N** • **Aux Out 1** • **Arm** • **External Trigger** • **Fire N and 1** (deprecated)
LineScanSpeed	Floating Point	Configures the number of rows read per second.
LogLevel	Enumerated	Sets the level of logging used. Options: • **NONE** • **FATAL** • **ERROR** • **WARN** • **INFO** *(default)* • **DEBUG**

Feature	Type	Description
		- **VERB** The log level is set independently for each device library using the system handle. See **Opening a Camera Handle**. Changing the LogLevel changes the verbosity of output. The log file can be found in the temporary folder of the operating system. The default level can also be changed by setting an environment variable named after the library, e.g. atdevregcamloglevel, to one of the options above.
LUTIndex	Integer	Sets the position in the LUT to read/write a new pixel map
LUTValue	Integer	Sets the value in LUT in position specified by "LUT Index"
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not. Default state is set to true.
MultitrackCount	Integer	It is in the range 1-256. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: - **Mono12** - **Mono12Packed** - **Mono16** - **Mono32** See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PreAmpGainControl feature.
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: - **280 MHz** - **100 MHz**
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
ReadoutTime	Floating Point	This feature will return the time in seconds to readout data from a sensor.
RollingShutterGlobalClear	Boolean	Enables Rolling Shutter Global Clear readout mode.
RowNExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE of ROW N in Rolling Shutter.
RowNExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE of ROW N in Rolling Shutter.
RowReadTime	Floating	Configures the time in seconds to read a single row.

Feature	Type	Description
	Point	
ScanSpeedControlEnable	Boolean	Configures whether the LineReadSpeed and RowReadTime can be altered.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is disabled by default at power up and must be enabled for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature where available for example on the Neo camera.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorReadoutMode	Enumerated	Configures the direction in which the sensor will be read out. Options • **Bottom Up Sequential** • **Bottom Up Simultaneous** • **Centre Out Simultaneous** • **Outside In Simultaneous** • **Top Down Sequential** • **Top Down Simultaneous**
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
ShutterMode	Enumerated	Controls the behavior of the shutter. Options: • **Open** • **Closed** • **Auto**
ShutterOutputMode	Enumerated	Controls the mode the external trigger will run in. External Shutter signal can either be set to high (open) or low (closed). ShutterOutput can be triggered by setting AuxOutSourceTwo to ExternalShutterControl. Options: • **Open** • **Closed**
ShutterTransferTime	Floating Point	Sets the time at which the shutter will be opened before an exposure starts. Only available whenever the ShutterMode is set to 'Auto'.
SimplePreAmpGainControl	Enumerated	Wrapper Feature to simplify selection of the sensitivity and dynamic range options. This feature should be used as a replacement for the PreAmpGainControl feature as some of the options in the PreAmpGainControl feature are not supported on all cameras. Supported Bit Depth will be dependent on the camera. See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PixelEncoding feature. Options: • **11-bit (high well capacity)** **Or** **12-bit (high well capacity)** • **11-bit (low noise)** **Or** **12-bit (low noise)** • **16-bit (low noise & high well capacity)**
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to generate each frame on the camera whenever the trigger mode is set to *Software*.
SoftwareVersion	String	Returns the version of the SDK.
SpuriousNoiseFilter	Boolean	Enables or Disables the Spurious Noise Filter
StaticBlemishCorrection	Boolean	Enables or Disables Static Blemish Correction
Temperature Control	Enumerated	Allows the user to set the target temperature of the sensor based on a list of valid temperatures.
Temperature Status	Enumerated	Reports the current state of cooling towards the Target Sensor Temperature. [Read Only]

Feature	Type	Description
		Options: • **Cooler Off** • **Stabilised** • **Cooling** • **Drift** • **Not Stabilised** • **Fault**
TimestampClock	Integer	Reports the current value of the camera’s internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera’s internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera’s internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Allows the user to configure the camera trigger mode at a high level. If the trigger mode is set to *Advanced* then the *Trigger Selector* and *Trigger Source* feature must also be set. Options: • **Internal** • **Software** (Not available on Apogee) • **External** • **External Start** • **External Exposure**
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.2. Neo

Feature	Type	Description
AccumulateCount	Integer	Sets the number of images that should be summed to obtain each image in sequence.
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AOIBinning	Enumerated	Sets up pixel binning on the camera. Options: • **1x1** • **2x2** • **3x3** • **4x4** • **8x8** See **Section 4.7 Area of Interest.**
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest**.
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels.

Feature	Type	Description
		See **Section 4.7 Area of Interest**.
AuxiliaryOutSource	Enumerated	Configures which signal appears on the auxiliary output pin. Options: • **FireRow1** • **FireRowN** • **FireAll** • **FireAny**
Baseline	Integer	Returns the baseline level of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. . Supported Bit Depth will be dependent on the camera. Options: • **11 Bit or 12 Bit** • **16 Bit**
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Floating Point	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system. Register a callback to this feature to be notified if the camera is disconnected.
ControllerID	String	Returns a unique identifier for the camera controller device. i.e. Frame grabber over Cameralink
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the 'AcquisitionStop' command is issued. Options: • **Fixed** • **Continuous**
DeviceVideoIndex	Integer	Returns the /dev/videoN number in Linux only
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • **Rolling** • **Global**
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
EventSelector	Enumerated	Selects the acquisition events you wish to enable or disable using the EventEnable feature. Options: • **ExposureEndEvent** • **ExposureStartEvent** • **RowNExposureEndEvent** • **RowNExposureStartEvent** • **EventsMissedEvent** • **BufferOverflowEvent**
ExposureTime	Floating Point	The requested exposure time in seconds. *Note: In some modes the exposure time can also be modified while the acquisition is running.*
ExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.

Feature	Type	Description
ExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low** • **On**
FastAOIFrameRateEnable	Boolean	Enables faster framerates at small AOIs.
FirmwareVersion	String	Returns the camera firmware version
FrameCount	Integer	Configures the number of images to acquire in the sequence while **CycleMode** is **Fixed**. The value of FrameCount must be any value which is a multiple of AccumulateCount. This ensures the accumulation contains the correct number of frames. When this feature is unavailable then the camera does not currently support fixed length series, therefore you must explicitly abort the acquisition once you have acquired the amount of frames required.
FrameRate	Floating Point	Configures the frame rate in Hz at which each image is acquired during any acquisition sequence. This is the rate at which frames are acquired by the camera which may be different from the rate at which frames are delivered to the user. For example when AccumulateCount has a value other than 1, the apparent frame rate will decrease proportionally.
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InterfaceType	String	Returns the camera interface type. Current types: Options: • **CL 3 Tap** • **CL 2x5 Tap** • **CL 10 Tap**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin that you subsequently wish to configure using the *IO Invert* Feature. Options: • **Fire 1** • **Aux Out 1** • **Arm** • **External Trigger** • **Fire N and 1** (deprecated)
LineScanSpeed	Floating Point	Configures the number of rows read per second.
LogLevel	Enumerated	Sets the level of logging used. Options: • **NONE** • **FATAL** • **ERROR** • **WARN** • **INFO** *(default)* • **DEBUG** • **VERB** The log level is set independently for each device library using the system handle. See **Opening a Camera Handle**. Changing the LogLevel changes the verbosity of output. The log file can be found in the temporary folder of the operating system. The default level can also be changed by setting an

Feature	Type	Description
		environment variable named after the library, e.g. atdevregcamloglevel, to one of the options above.
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MicrocodeVersion	String	Returns a revision code for the internal USB firmware within the camera head. Only available on USB interface.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not. Default state is set to true.
MultitrackCount	Integer	It is in the range 1-256. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: **Mono12** **Mono12Packed** **Mono16** **Mono32** See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PreAmpGainControl feature.
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options Neo: • **280 MHz** • **200 MHz** • **100 MHz**
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
PreAmpGain	Enumerated	Configures the gain applied to the gain channel selected through the *Pre Amp Gain Selector* Feature. Options: • **x1** • **x2** • **x10** • **x30**
PreAmpGainChannel	Enumerated	Configures which pre amp gain channel(s) will be used for reading out the sensor. Options:

Feature	Type	Description
		- **High** - **Low** - **Both**
PreAmpGainControl	Enumerated	Wrapper Feature to simplify access to the PreAmpGain, PreAmpGainChannel and PreAmpGainSelector feaures. See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PixelEncoding feature. This feature is deprecated and should be replaced by the SimplePreAmpGainControl feature as some of the options may not be supported. Options: - **Gain 1 (11 bit)** - **Gain 2 (11 bit)** - **Gain 3 (11 bit)** - **Gain 4 (11 bit)** - **Gain 1 Gain 3 (16 bit)** - **Gain 1 Gain 4 (16 bit)** - **Gain 2 Gain 3 (16 bit)** - **Gain 2 Gain 4 (16 bit)**
PreAmpGainSelector	Enumerated	Selects the gain channel that you subsequently wish to configure using the *Pre Amp Gain* Feature. Options: - **High** - **Low**
ReadoutTime	Floating Point	This feature will return the time in seconds to readout data from a sensor.
RollingShutterGlobalClear	Boolean	Enables Rolling Shutter Global Clear readout mode.
RowNExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE of ROW N in Rolling Shutter.
RowNExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE of ROW N in Rolling Shutter.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is disabled by default at power up and must be enabled for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature where available for example on the Neo camera.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
SimplePreAmpGainControl	Enumerated	Wrapper Feature to simplify selection of the sensitivity and dynamic range options. This feature should be used as a replacement for the PreAmpGainControl feature as some of the options in the PreAmpGainControl feature are not supported on all cameras. Supported Bit Depth will be dependent on the camera. See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PixelEncoding feature. Options: - **11-bit (high well capacity)** **Or** **12-bit (high well capacity)** - **11-bit (low noise)** **Or** **12-bit (low noise)** - **16-bit (low noise & high well capacity)**
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to generate each frame on the camera whenever the trigger mode is set to *Software*.
SoftwareVersion	String	Returns the version of the SDK.
SpuriousNoiseFilter	Boolean	Enables or Disables the Spurious Noise Filter
StaticBlemishCorrection	Boolean	Enables or Disables Static Blemish Correction
TargetSensor	Floating	Configures the temperature to which the sensor will be cooled. To be used

Feature	Type	Description
Temperature	Point	for cameras with no correction data (-50->25). Otherwise *TemperatureControl* should be used.
Temperature Control	Enumerated	Allows the user to set the target temperature of the sensor based on a list of valid temperatures.
Temperature Status	Enumerated	Reports the current state of cooling towards the Target Sensor Temperature. [Read Only] Options: • **Cooler Off** • **Stabilised** • **Cooling** • **Drift** • **Not Stabilised** • **Fault**
TimestampClock	Integer	Reports the current value of the camera's internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera's internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera's internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Allows the user to configure the camera trigger mode at a high level. If the trigger mode is set to *Advanced* then the *Trigger Selector* and *Trigger Source* feature must also be set. Options: • **Internal** • **Software** (Not available on Apogee) • **External** • **External Start** • **External Exposure**
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.3. ISTAR-SCMOS

Feature	Type	Description
AccumulateCount	Integer	Sets the number of images that should be summed to obtain each image in sequence.
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AOIBinning	Enumerated	Sets up pixel binning on the camera. Options: • **1x1** • **2x2** • **3x3** • **4x4** • **8x8** See **Section 4.7 Area of Interest.**
AOIHBin	Integer	Configures the Horizontal Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Options: • **Image** • **Kinetics**

Feature	Type	Description
		- **TDI** - **Multitrack**
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest**.
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest**.
AuxOutSourceTwo	Enumerated	AuxOutSourceTwo is a configurable output available to the user on the D-type (Zyla) and SMA (iStar-sCMOS). Options: - **ExternalShutterControl** - **FrameClock** - **RowClock** - **ExposedRowClock**
BackoffTemperatureOffset	Float	The Backoff temperature offset of the cooler subsystem.
Baseline	Integer	Returns the baseline level of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. . Supported Bit Depth will be dependent on the camera. Options: - **11 Bit or 12 Bit** - **16 Bit**
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Floating Point	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system. Register a callback to this feature to be notified if the camera is disconnected.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the 'AcquisitionStop' command is issued. Options: - **Fixed** - **Continuous**
DDGIOCEnable	Boolean	Enable or disable the integrate on chip option.
DDGIOCNumberOfPulses	Integer	Configures the number of pulses generated when the integrate on chip option is enabled.
DDGIOCPeriod	Integer	Configures the period (in picoseconds) of the pulses generated when the integrate on chip option is enabled.
DDGOutputDelay	Integer	Configures the delay (in picoseconds) for the currently selected DDG output.
DDGOutputEnable	Boolean	Enable or disable the currently selected DDG output.
DDGOutputStepEnable	Boolean	Enable or disable DDG step mode for the currently selected DDG output.
DDGOpticalWidthEnable	Boolean	Enable or disable the optical gate width option.
DDGOutputPolarity	Enumerated	Configures the polarity for the currently selected DDG output.

Feature	Type	Description
		Options: - **Negative** - **Positive**
DDGOutputSelector	Enumerated	Selects the DDG output that you subsequently wish to configure using the DDG output Features. Options: - **Gater** - **Output A** - **Output B** - **Output C**
DDGOutputWidth	Integer	Configures the width (in picoseconds) for the currently selected DDG output.
DDGStepCount	Integer	Configures the number of frames that will use the same step value. If you are using accumulations and you require each of the accumulated frames to use the same step values then this feature should be set to the number of accumulations.
DDGStepDelayCoefficientA	Floating Point	Configures coefficient A for the delay DDG step.
DDGStepDelayCoefficientB	Floating Point	Configures coefficient B for the delay DDG step.
DDGStepDelayMode	Enumerated	Configures the delay DDG step mode. Options: - **Off** - **Constant** - **Exponential** - **Logarithmic**
DDGStepEnabled	Boolean	Indicates when DDG step is enabled.
DDGStepUploadProgress	Integer	Indicates the current progess of the DDG step upload.
DDGStepUploadRequired	Boolean	Indicates if a DDG step upload is required.
DDGStepWidthCoefficientA	Floating Point	Configures coefficient A for the width DDG step.
DDGStepWidthCoefficientB	Floating Point	Configures coefficient B for the width DDG step.
DDGStepWidthMode	Enumerated	Configures the width DDG step mode. Options: - **Off** - **Constant** - **Exponential** - **Logarithmic**
DDGStepUploadModeValues	Command	Uploads the DDG step values to the DDG frame buffer.
DeviceCount	Integer	Returns the number of cameras detected.
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. - For pulsed or fast moving images *Global* shuttering is recommended. - For the highest frame rates and best noise performance *Rolling* is recommended. Options: - **Rolling** (not available in iStar-sCMOS) - **Global**
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
EventSelector	Enumerated	Selects the acquisition events you wish to enable or disable using the EventEnable feature. Options: - **ExposureEndEvent** - **ExposureStartEvent**

Feature	Type	Description
		- **EventsMissedEvent** - **BufferOverflowEvent**
ExposedPixelHeight	Integer	Configures the exposure window in pixels.
ExposureTime	Floating Point	The requested exposure time in seconds. *Note: In some modes the exposure time can also be modified while the acquisition is running.*
ExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExternalTriggerDelay	Floating Point	Sets the delay time between the camera receiving an external trigger and the acquisition start.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: - **Off** - **On**
FastAOIFrameRateEnable	Boolean	Enables faster framerates at small AOIs.
FirmwareVersion	String	Returns the camera firmware version
FrameCount	Integer	Configures the number of images to acquire in the sequence while **CycleMode** is **Fixed**. The value of FrameCount must be any value which is a multiple of AccumulateCount. This ensures the accumulation contains the correct number of frames. When this feature is unavailable then the camera does not currently support fixed length series, therefore you must explicitly abort the acquisition once you have acquired the amount of frames required.
FrameRate	Floating Point	Configures the frame rate in Hz at which each image is acquired during any acquisition sequence. This is the rate at which frames are acquired by the camera which may be different from the rate at which frames are delivered to the user. For example when AccumulateCount has a value other than 1, the apparent frame rate will decrease proportionally.
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
GateMode	Enumerated	Selects the photocathode gating mode. Options: - **CWOn** - **CWOff** - **FireOnly** - **GateOnly** - **FireAndGate** - **DDG**
I2CAddress	Integer	Configures the address of the I^2C device.
I2CByte	Integer	Get/Set the I^2C byte currently selected by the I2CByteSelector.
I2CByteCount	Integer	Configures the number of I^2C bytes to read/write.
I2CByteSelector	Integer	Selects the I^2C byte that you subsequently wish to Get/Set using the I2CByte Feature.
I2CRead	Command	Performs the configured I^2C read.
I2CWrite	Command	Performs the configured I^2C write.
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InsertionDelay	Enumerated	Configures the length of the insertion delay. Options: - **Normal** - **Fast**
InterfaceType	String	Returns the camera interface type. Current types: Options Neo/Zyla:

Feature	Type	Description
		- **USB3** - **CL 3 Tap** - **CL 2x5 Tap** - **CL 10 Tap** Options iStar-sCMOS: - **USB3** Options Apogee: - **USB2** - **Ethernet**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin that you subsequently wish to configure using the *IO Invert* Feature. Options Neo/Zyla: - **Fire 1** - **Aux Out 1** - **Arm** - **External Trigger** - **Fire N and 1** (deprecated) - **Aux Out 2** - **External Trigger**
LogLevel	Enumerated	Sets the level of logging used. Options: - **NONE** - **FATAL** - **ERROR** - **WARN** - **INFO** *(default)* - **DEBUG** - **VERB** The log level is set independently for each device library using the system handle. See **Opening a Camera Handle**. Changing the LogLevel changes the verbosity of output. The log file can be found in the temporary folder of the operating system. The default level can also be changed by setting an environment variable named after the library, e.g. atdevregcamloglevel, to one of the options above.
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MCPGain	Integer	Controls the voltage applied across the microchannel plate.
MCPIntelligate	Boolean	Enables or disables the MCP Intelligate mode. Only available when normal insertion delay is selected.
MCPVoltage	Integer	Indicates the current MCP voltage.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not.

Feature	Type	Description
		Default state is set to true.
MultitrackCount	Integer	It is in the range 1-256. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PIVEnable	Boolean	Enables PIV mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Neo, Zyla, iStar-sCMOS and SimCam Options: **Mono12** **Mono12Packed** **Mono16** **Mono32** SimCam only Options: **Mono8** **RGB8Packed** **Mono12Coded** **Mono12CodedPacked** **Mono22Parallel** **Mono22PackedParallel** See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PreAmpGainControl feature.
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options sCMOS: • **280 MHz** • **200 MHz** (Neo Only - deprecated) • **100 MHz** Options SimCam: • **550 MHz** Options Apogee: • **Normal** • **Fast** (Not Available on AltaE)
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
PreTriggerEnable	Boolean	Enable or disable the pre-trigger input.
ReadoutTime	Floating Point	This feature will return the time in seconds to readout data from a sensor.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is disabled by default at power up and must be enabled for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature where available for example on the Neo camera.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
ShutterMode	Enumerated	Controls the behavior of the shutter. Options: • **Open** • **Closed** • **Auto**
ShutterTransferTime	Floating	Sets the time at which the shutter will be opened before an exposure

Feature	Type	Description
	Point	starts. Only available whenever the ShutterMode is set to 'Auto'.
SimplePreAmpGainControl	Enumerated	Wrapper Feature to simplify selection of the sensitivity and dynamic range options. This feature should be used as a replacement for the PreAmpGainControl feature as some of the options in the PreAmpGainControl feature are not supported on all cameras. Supported Bit Depth will be dependent on the camera. See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PixelEncoding feature. Options: • **11-bit (high well capacity)** **Or** **12-bit (high well capacity)** • **11-bit (low noise)** **Or** **12-bit (low noise)** • **16-bit (low noise & high well capacity)**
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to generate each frame on the camera whenever the trigger mode is set to *Software*.
SoftwareVersion	String	Returns the version of the SDK.
SpuriousNoiseFilter	Boolean	Enables or Disables the Spurious Noise Filter
StaticBlemishCorrection	Boolean	Enables or Disables Static Blemish Correction
Temperature Status	Enumerated	Reports the current state of cooling towards the Target Sensor Temperature. [Read Only] Options:: • **Cooler Off** • **Stabilised** • **Cooling**: • **Drift** • **Not Stabilised** • **Fault**
TimestampClock	Integer	Reports the current value of the camera's internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera's internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera's internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Allows the user to configure the camera trigger mode at a high level. If the trigger mode is set to *Advanced* then the *Trigger Selector* and *Trigger Source* feature must also be set. Options: • **Internal** • **Software** (Not available on Apogee) • **External** • **External Start** • **External Exposure**
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.4. APOGEE

Feature	Type	Description

Feature	Type	Description
AcquiredCount	Integer	Dynamically incrementing count during an image sequence.
AOIHBin	Integer	Configures the Horizontal Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Options: • **Image** • **Kinetics** • **TDI** • **Multitrack**
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest**.
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. **Section 4.7 Area of Interest**.
BackoffTemperatureOffset	Float	The Backoff temperature offset of the cooler subsystem.
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. . Supported Bit Depth will be dependent on the camera. Options: • **12 Bit** (Not available AltaF/Aspen/Ascent) • **16 Bit** For AltaU/E this is determined by PixelReadoutRate: • **"Normal" -> "16-bit"** • **"Fast" -> "12-bit"**
CameraFamily	String	Returns the family of the camera.
CameraMemory	Integer	Returns the amount of available memory for storing images, in bytes.
CameraName	String	Returns the name of the camera.
ColourFilter	Enumerated	Controls which colour filter is enabled. Options: • **None** • **Bayer** • **TRUESENSE**
CoolerPower	Double	Percentage of maximum power being used by the cooler.
DDR2Type	String	Returns the amount of DDR2 Memory installed.
DeviceCount	Integer	Returns the number of cameras detected.
DisableShutter	Boolean	Enable or Disable shutter during acquisition. Overridden by ForceShutterOpen.
DriverVersion	String	Returns USB driver version if USB Interface. Returns cURL library version if Ethernet interface.
ExternalIOReadout	Boolean	When TRUE, the readout of the camera is no longer started by the external shutter. Instead, Pin 5 "External Readout Start" is used to start the readout. The default value of this variable after initialization is FALSE. Not available on Ascent. Must use IOControl to cause the camera toconsider Pin 5 to be "External Readout Start" and not user-defined.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low** • **Medium** • **High**

Feature	Type	Description
FirmwareVersion	String	Returns the camera firmware version
ForceShutterOpen	Boolean	Choose whether to force the shutter to open. Overrides DisableShutter.
FrameCount	Integer	Configures the number of images to acquire in the sequence while **CycleMode** is **Fixed**. The value of FrameCount must be any value which is a multiple of AccumulateCount. This ensures the accumulation contains the correct number of frames. When this feature is unavailable then the camera does not currently support fixed length series, therefore you must explicitly abort the acquisition once you have acquired the amount of frames required.
FrameInterval	Floating Point	The interval in seconds between the end of readout of one image to the beginning of exposure of the next.
FrameIntervalTiming	Boolean	Configures whether the timing of image acquisition is determined by FrameInterval or FrameRate
FrameRate	Floating Point	Configures the frame rate in Hz at which each image is acquired during any acquisition sequence. This is the rate at which frames are acquired by the camera which may be different from the rate at which frames are delivered to the user. For example when AccumulateCount has a value other than 1, the apparent frame rate will decrease proportionally.
HeatSinkTemperature	Float	Returns the current Heatsink Temperature. Not available for Ascent.
InputVoltage	Float	Returns the operating input voltage to the camera.
InterfaceType	String	Returns the camera interface type. Current types: Options: • **USB2** • **Ethernet**
IOControl	Enum	Configures whether selected IO is default or user defined. Options • **Default** • **User**
IODirection	Boolean or Enumerated	Configures whether selected IO is input or output. Cannot be changed for AltaF or Ascent. Options • **Input / 0** • **Output / 1**
IOState	Boolean	Configures whether selected IO is enabled or disabled.
IOSelector	Enumerated	Selects the IO Pin that you subsequently wish to configure using the *IO Invert* Feature. Options Apogee: • **External Trigger** (I/O Signal 1: TriggerNormal) • **Fire** (I/O Signal 2: Shutter Output) • **Shutter Strobe** (I/O Signal 3: Shutter Strobe) • **External Exposure** (I/O Signal 4 ExternalShutter) • **External Readout** (I/0 Signal 5: ExternalIOReadout)
IRPreFlashEnable	Boolean	When TRUE, the camera normalizes the sensor before an image is taken with a flash of IR. Not available for Ascent or other Interline transfer CCDs.
KeepCleanEnable	Boolean	Enableds/Disables any flushing command sent by the driver.
KeepCleanPostExposure Enable	Boolean	Enables/Disables the camera control firmware to/from immediately beginning an internal flush cycle after an exposure.
LogLevel	Enumerated	Sets the level of logging used. Options: • **NONE**

Feature	Type	Description
		- **FATAL** - **ERROR** - **WARN** - **INFO** *(default)* - **DEBUG** - **VERB** The log level is set independently for each device library using the system handle. See **Opening a Camera Handle**. Changing the LogLevel changes the verbosity of output. The log file can be found in the temporary folder of the operating system. The default level can also be changed by setting an environment variable named after the library, e.g. atdevregcamloglevel, to one of the options above.
MicrocodeVersion	String	Returns a revision code for the internal USB firmware within the camera head. Only available on USB interface.
Overlap	Boolean	Enables overlap readout mode.
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: - **Normal** - **Fast** (Not Available on AltaE)
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
PortSelector	Integer	Configures the port being used. Not implemented if fixed Single readout. Min is always 0. Max is {0, 1, 3} depending on {Single, Dual, Quad} readout. If the camera is switchable between Single and Dual readout then the value of PortSelector required to set the gain/offset for Single readout is camera-dependent (i.e. it could be 0 or 1).
PreAmpGainValue	Integer	This is the value of the pre-amplifier gain, for the currently-selected ADC/channel pair as selected by PortSelector if PortSelector is implemented. Only effective for Fast PixelReadoutRate for AltaU/AltaE/AltaF/Aspen.
PreAmpOffsetValue	Integer	This is the value of the pre-amplifier offset, for the currently-selected ADC/channel pair as selected by PortSelector, if PortSelector is implemented. Only effective for Fast PixelReadoutRate for AltaU/AltaE/AltaF/Aspen.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is disabled by default at power up and must be enabled for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature where available for example on the Neo camera.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorModel	String	Returns the sensor model installed in the camera.
SensorType	Enumerated	Returns true for CCD, False for CMOS. Options: - **CCD** - **CMOS**
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
ShutterAmpControl	Boolean	Disables the CCD voltage while the shutter strobe is high.
ShutterMode	Enumerated	Controls the behavior of the shutter.

Feature	Type	Description
		Options: • **Open** • **Closed** • **Auto**
ShutterState	Boolean	Returns whether shutter is opened or closed.
ShutterStrobePeriod	Double	Sets the period of the shutter strobe on pin3. Must use IOControl to cause the camera to consider Pin 3 to be "Shutter Strobe Output" and not user-defined.
ShutterStrobePosition	Double	Sets the delay from the time the exposure begins to the time the rising edge of the shutter strobe period appears on pin 3. Must use IOControl to cause the camera to consider Pin 3 to be "Shutter Strobe Output" and not user-defined.
SoftwareVersion	String	Returns the version of the SDK.
Temperature Status	Enumerated	Reports the current state of cooling towards the Target Sensor Temperature. [Read Only] Options: • **Cooler Off** • **Stabilised** • **Cooling** • **Backoff**
TransmitFrames	Boolean	If false all image data will be concatenated and obtained as a single download at the end of the sequence. If true each image (or image row in the case of TDI) from a sequence will be available for download after it is read out (digitised).
TriggerMode	Enumerated	Allows the user to configure the camera trigger mode at a high level. If the trigger mode is set to *Advanced* then the *Trigger Selector* and *Trigger Source* feature must also be set. Options: • **Internal** • **Software** (Not available on Apogee) • **External** • **External Start** • **External Exposure**
UsbProductId	Integer	Returns the USB Product ID associated with the camera system. Only available for USB interface.
UsbDeviceId	Integer	Returns the USB Device ID associated with the camera system. Only available on USB interface.

4.3.5. SONA

Feature	Type	Description
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AOIBinning	Enumerated	Configures symmetric binning of the sensor area of interest in pixels. Options: • 1x1 • 2x2 • 3x3 • 4x4 • 8x8

Feature	Type	Description
AOIHBin	Integer	Configures the horizontal binning of the sensor area of interest in pixels. See **Section 4.7 Area of Interest.**
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Configures the active readout mode. Options: • Image – a single AOI is read from the sensor. • Multitrack – multiple tracks of the AOIWidth can be read out.
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest.**
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**.
AuxiliaryOutSource	Enumerated	Configures which signal appears on the auxiliary output pin. Options: • FireRow1 – High when the sensor is exposing row 1. • FireRowN – High when the sensor is exposing the last row in the AOI. • FireAll – High when all rows are exposing at the same time. • FireAny – High when any row of the sensor is exposing.
AuxOutSourceTwo	Enumerated	Allows the user to select which option is set for Aux 1. Options: • ExternalShutterControl • FrameClock • RowClock • ExposedClock
Baseline	Integer	Returns the baseline count value of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. Options: • 12 Bit • 16 Bit • 32 Bit
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Float	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraFamily	String	Returns the family of the camera.
CameraImagesWaiting	Integer	Number of images waiting to be transfered from camera. Placeholder feature – currently returns 1.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the

Feature	Type	Description
		camera will continue to acquire images indefinitely until the ‘AcquisitionStop’ command is issued. Options: • Fixed • Continuous
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • Rolling
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
EventSelector	Enumerated	Selects the acquisition events you wish to enable or disable using the EventEnable feature. Options: • **ExposureStartEvent** • **ExposureEndEvent** • **RowNExposureStartEvent** • **RowNExposureEndEvent** • **BufferOverflowEvent** • **EventsMissedEvent**
ExposureTime	Float	The requested exposure time in seconds.
ExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExternalTriggerDelay	Floating Point	Sets the delay time between the camera receiving an external trigger and the acquisition start.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low (default)** • **Medium** • **On**
FirmwareVersion	String	Returns the camera firmware version.
FrameGenFixedPixelValue	Integer	Configures the value returned by the Frame Generator when in fixed pixel mode. Otherwise, this feature is ignored. (See **FrameGenMode)**
FrameGenMode	Enumerated	Selects operation of simulated frame generation. Options: • Off – Normal image data returned. • ColumnCount – Pixel values contain an incrementing count for each column. • RowCount – Pixel values contain an incrementing count for each row in the image. • FrameCount – Pixel values contain an incrementing count for each frame in the image. • FixedPixel – All pixels have value of **FrameGenFixedPixelValue.**
FrameCount	Integer	Number of frames in a series when using fixed cycle mode.
FrameRate	Float	Configures the frame-rate in Hz when running in internal trigger.

Feature	Type	Description
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
GainMode	Enumerated	Returns the current gain mode Options: • Fastest frame rate (12-bit) • High dynamic range (16-bit)
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InterfaceType	String	Returns the camera interface type. Current types: • **USB3**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin to configure using the *IO Invert* Feature. Options: • **Spare Input** • **Aux Out 1** • **Aux Out 2** • **Fire N** • **Fire 1** • **Arm** • **External Trigger** • **External Exposure**
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not. Default state is set to true.
MultitrackCount	Integer	Configures the number of available multitracks. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: • **Mono12** • **Mono12Packed** • **Mono16** • **Mono32**
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.

Feature	Type	Description
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: • 25MHz
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
ReadoutTime	Floating Point	Returns the time to read out one frame.
RowNExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE of ROW N in Rolling Shutter.
RowNExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE of ROW N in Rolling Shutter.
RowReadTime	Floating Point	Returns the time to read out one row.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is enabled by default at power up for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
ShutterMode	Enumerated	Controls the behavior of the shutter. ShutterOutput can be triggered by setting AuxOutSourceTwo to ExternalShutterControl.
ShutterTransferTime	Floating Point	Sets the time at which the shutter will be opened before an exposure starts. Only available whenever the ShutterMode is set to ‘Auto’.
SimplePreAmpGainControl	Enumerated	Alias for GainMode.
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to trigger an exposure of a new frame on the camera whenever the trigger mode is set to *Software*.
SpuriousNoiseFilter	Boolean	Enables spurious noise filter.
TargetSensorTemperature	Floating Point	Configures the temperature to which the sensor will be cooled.
TemperatureControl	Enumerated	Allows the user to set the target temperature of the sensor based on a list of valid temperatures. Options: • 15.0 • -25.0 • -45.0
TemperatureStatus	Enumerated	Reports the current cooling state. [Read Only] Options: • **Cooler Off** – cooler has not been enabled. • **Cooling** – cooler is enabled but has not reached the set point for long enough. • **Stabilised** – sensor temperature has stabilized at the set point. • **Drift** – Cooler was previously stable but has drifted more the 2°C from set point. • **Not Stabilised** – Cooling has reached its target temperature but sensor temperature has not yet stabilised • **Fault** – Fault in cooling system. Check water supply or that air vents are not blocked. • **Sensor Over Temperature** – Temperature of sensor temperature has exceeded allowed limit. Check water supply or that air vents are not blocked.
TimestampClock	Integer	Reports the current value of the camera’s internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated

Feature	Type	Description
		by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera's internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera's internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Configures the camera trigger mode. Currently supported options: • **Internal** – Camera will run at rate controlled by the FrameRate feature. • **Software** – Each exposure is triggered by the SoftwareTrigger feature. • **External** – Each exposure is triggered by an external trigger pulse. • **External Start** – Each series is triggered by an external trigger pulse. • **External Exposure –** Each exposure is controlled by high level of external trigger pulse
TriggerSource	Enumerated	Configures the source of the external trigger signal. • **D-Type Connector** - connector on the camera.
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.6. MARANA

Feature	Type	Description
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AOIBinning	Enumerated	Configures symmetric binning of the sensor area of interest in pixels. Options: • 1x1 • 2x2 • 3x3 • 4x4 • 8x8
AOIHBin	Integer	Configures the horizontal binning of the sensor area of interest in pixels. See **Section 4.7 Area of Interest.**
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Configures the active readout mode. Options: • Image – a single AOI is read from the sensor. • Multitrack – multiple tracks of the AOIWidth can be read out.
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest.**
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**.
AuxiliaryOutSource	Enumerated	Configures which signal appears on the auxiliary output pin. Options: • FireRow1 – High when the sensor is exposing row 1. • FireRowN – High when the sensor is exposing the last row in the

Feature	Type	Description
		AOI. • FireAll – High when all rows are exposing at the same time. • FireAny – High when any row of the sensor is exposing.
AuxOutSourceTwo	Enumerated	Allows the user to select which option is set for Aux 1. Options: • ExternalShutterControl • FrameClock • RowClock • ExposedClock
Baseline	Integer	Returns the baseline count value of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. Options: • 12 Bit • 16 Bit • 32 Bit
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Float	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraFamily	String	Returns the family of the camera.
CameraImagesWaiting	Integer	Number of images waiting to be transfered from camera. Placeholder feature – currently returns 1.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire ‘FrameCount’ number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the ‘AcquisitionStop’ command is issued. Options: • Fixed • Continuous
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • Rolling
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
EventSelector	Enumerated	Selects the acquisition events you wish to enable or disable using the EventEnable feature. Options: • **ExposureStartEvent** • **ExposureEndEvent** • **RowNExposureStartEvent** • **RowNExposureEndEvent**

Feature	Type	Description
		• **BufferOverflowEvent** • **EventsMissedEvent**
ExposureTime	Float	The requested exposure time in seconds.
ExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE in Global Shutter and FIRE of Row 1 in Rolling Shutter.
ExternalTriggerDelay	Floating Point	Sets the delay time between the camera receiving an external trigger and the acquisition start.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low** • **Medium** • **On**
FirmwareVersion	String	Returns the camera firmware version.
FrameGenFixedPixelValue	Integer	Configures the value returned by the Frame Generator when in fixed pixel mode. Otherwise, this feature is ignored. (See **FrameGenMode)**
FrameGenMode	Enumerated	Selects operation of simulated frame generation. Options: • Off – Normal image data returned. • ColumnCount – Pixel values contain an incrementing count for each column. • RowCount – Pixel values contain an incrementing count for each row in the image. • FrameCount – Pixel values contain an incrementing count for each frame in the image. • FixedPixel – All pixels have value of **FrameGenFixedPixelValue.**
FrameCount	Integer	Number of frames in a series when using fixed cycle mode.
FrameRate	Float	Configures the frame-rate in Hz when running in internal trigger.
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
GainMode	Enumerated	Returns the current gain mode Options: • STD HG • HDR
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InterfaceType	String	Returns the camera interface type. Current types: • **USB3**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin to configure using the *IO Invert* Feature. Options: • **Spare Input** • **Aux Out 1** • **Aux Out 2** • **Fire N** • **Fire 1** • **Arm** • **External Trigger**

Feature	Type	Description
		- **External Exposure**
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not. Default state is set to true.
MultitrackCount	Integer	Configures the number of available multitracks. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: - **Mono12** - **Mono12Packed** - **Mono16** - **Mono32**
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: - 25MHz
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
ReadoutTime	Floating Point	Returns the time to read out one frame.
RowNExposureEndEvent	Integer	When enabled this will notify the user on the Negative edge of the FIRE of ROW N in Rolling Shutter.
RowNExposureStartEvent	Integer	When enabled this will notify the user on the Positive edge of the FIRE of ROW N in Rolling Shutter.
RowReadTime	Floating Point	Returns the time to read out one row.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is enabled by default at power up for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
ShutterMode	Enumerated	Controls the behavior of the shutter. ShutterOutput can be triggered by setting AuxOutSourceTwo to ExternalShutterControl.
ShutterTransferTime	Floating	Sets the time at which the shutter will be opened before an exposure starts.

Feature	Type	Description
	Point	Only available whenever the ShutterMode is set to 'Auto'.
SimplePreAmpGainControl	Enumerated	Alias for GainMode.
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to trigger an exposure of a new frame on the camera whenever the trigger mode is set to *Software*.
SpuriousNoiseFilter	Boolean	Enables spurious noise filter.
TargetSensorTemperature	Floating Point	Configures the temperature to which the sensor will be cooled.
TemperatureControl	Enumerated	Allows the user to set the target temperature of the sensor based on a list of valid temperatures. Options: • 15.0 • -25.0 • -45.0
TemperatureStatus	Enumerated	Reports the current cooling state. [Read Only] Options: • **Cooler Off** – cooler has not been enabled. • **Cooling** – cooler is enabled but has not reached the set point for long enough. • **Stabilised** – sensor temperature has stabilized at the set point. • **Drift** – Cooler was previously stable but has drifted more the 2°C from set point. • **Not Stabilised** – Cooling has reached its target temperature but sensor temperature has not yet stabilised • **Fault** – Fault in cooling system. Check water supply or that air vents are not blocked. • **Sensor Over Temperature** – Temperature of sensor temperature has exceeded allowed limit. Check water supply or that air vents are not blocked.
TimestampClock	Integer	Reports the current value of the camera's internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera's internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera's internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Configures the camera trigger mode. Currently supported options: • **Internal** – Camera will run at rate controlled by the FrameRate feature. • **Software** – Each exposure is triggered by the SoftwareTrigger feature. • **External** – Each exposure is triggered by an external trigger pulse. • **External Start** – Each series is triggered by an external trigger pulse. • **External Exposure –** Each exposure is controlled by high level of external trigger pulse
TriggerSource	Enumerated	Configures the source of the external trigger signal. • **D-Type Connector** - connector on the camera.
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.7. BALOR

Feature	Type	Description
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.

Feature	Type	Description
AOIBinning	Enumerated	Configures symmetric binning of the sensor area of interest in pixels. Options: • 1x1 • 2x2 • 3x3 • 4x4 • 8x8
AOIHBin	Integer	Configures the horizontal binning of the sensor area of interest in pixels. See **Section 4.7 Area of Interest.**
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILayout	Enumerated	Configures the active readout mode. Options: • Image – a single AOI is read from the sensor. • Multitrack – multiple tracks of the AOIWidth can be read out.
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIStride	Integer	The size of one row in the image in bytes. Extra padding bytes may be added to the end of each line after pixel data to comply with line size granularity restrictions imposed by the underlying hardware interface. See **Section 4.4 Image Format .**
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest.**
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest**.
AuxiliaryOutSource	Enumerated	Configures which signal appears on the auxiliary output pin. Options: • FireRow1 – High when the sensor is exposing row 1. • FireRowN – High when the sensor is exposing the last row in the AOI. • FireAll – High when all rows are exposing at the same time. • FireAny – High when any row of the sensor is exposing.
AuxOutSourceTwo	Enumerated	Allows the user to select which option is set for Aux 1. Options: • ExternalShutterControl • FrameClock • RowClock • ExposedClock
Baseline	Integer	Returns the baseline count value of the image with current settings
BitDepth	Enumerated	Returns the number bits used to store information about each pixel of the image. Options: • 12 Bit • 16 Bit • 32 Bit
BufferOverflowEvent	Integer	When enabled this will notify the user that the image buffer on the camera has been exceeded, causing the current acquisition to stop.
BytesPerPixel	Float	Returns the calculated bytes per pixel. This is read only.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.

Feature	Type	Description
CameraFamily	String	Returns the family of the camera.
CameraImagesWaiting	Integer	Number of images waiting to be transfered from camera. Placeholder feature – currently returns 1.
CameraModel	String	Returns the camera model.
CameraName	String	Returns the name of the camera.
CameraPresent	Boolean	Returns whether the camera is connected to the system.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the 'AcquisitionStop' command is issued. Options: • Fixed • Continuous
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • Global
EventEnable	Boolean	Enable or Disable the acquisition event selected via the EventSelector feature.
EventsMissedEvent	Integer	When enabled this will notify the user that an acquisition event, which the user registered a callback for, has been missed.
ExposureTime	Float	The requested exposure time in seconds.
ExternalTriggerDelay	Floating Point	Sets the delay time between the camera receiving an external trigger and the acquisition start.
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low** • **Medium** • **On**
FirmwareVersion	String	Returns the camera firmware version.
FrameGenFixedPixelValue	Integer	Configures the value returned by the Frame Generator when in fixed pixel mode. Otherwise, this feature is ignored. (See **FrameGenMode)**
FrameGenMode	Enumerated	Selects operation of simulated frame generation. Options: • Off – Normal image data returned. • ColumnCount – Pixel values contain an incrementing count for each column. • RowCount – Pixel values contain an incrementing count for each row in the image. • FrameCount – Pixel values contain an incrementing count for each frame in the image. • FixedPixel – All pixels have value of **FrameGenFixedPixelValue.**
FrameCount	Integer	Number of frames in a series when using fixed cycle mode.
FrameRate	Float	Configures the frame-rate in Hz when running in internal trigger.
FullAOIControl	Boolean	Indicates whether or not the camera supports arbitrary AOI selection. If this feature is false then the supported AOI's are limited to those listed in **Section 4.7 Area of Interest.**
GainMode	Enumerated	Returns the current gain mode Options: • Standard

Feature	Type	Description
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
InterfaceType	String	Returns the camera interface type. Current types: - **CXP**
IOInvert	Boolean	Indicates whether or not the operation of the IO Pin selected through the *IO Selector* Feature is inverted.
IOSelector	Enumerated	Selects the IO Pin to configure using the *IO Invert* Feature. Options: - **Spare Input** - **Aux Out 1** - **Aux Out 2** - **Fire N** - **Fire 1** - **Arm** - **External Trigger** - **External Exposure**
MaxInterfaceTransferRate	Float	Returns the maximum sustainable transfer rate in frames per second of the interface for the current shutter mode and AOI.
MetadataEnable	Boolean	Enable metadata. This is a global flag which will enable inclusion of metadata in the data stream as described in **Section 4.6, Metadata**. When this flag is enabled the data stream will always contain the MetadataFrame information. This will override the subsequent metadata settings when disabled. For example: If this feature is disabled and MetadataTimestamp is enabled, then metadata will not be included in the data stream. For example: If this feature is enabled and MetadataTimestamp is disabled, then metadata will be included in the data stream, but without timestamp information.
MetadataFrame	Boolean	Indicates whether the MetadataFrame information is included in the data stream. This is read only and is automatically sent if metadata is enabled.
MetadataTimestamp	Boolean	Enables inclusion of timestamp information in the metadata stream. The timestamp indicates the time at which the exposure for the frame started.
MultitrackBinned	Boolean	Configures whether the currently selected multitrack will be binned or not. Default state is set to true.
MultitrackCount	Integer	Configures the number of available multitracks. When set to 0, multitrack is disabled.
MultitrackEnd	Integer	Configures the row at which the currently selected multitrack ends.
MultitrackSelector	Integer	Selects multitrack index. It is in the range 0-255.
MultitrackStart	Integer	Configures the row at which the currently selected multitrack begins.
Overlap	Boolean	Enables overlap readout mode.
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: - **Mono12** - **Mono12Packed** - **Mono16** - **Mono32**
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: - 180MHz
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
ReadoutTime	Floating	Returns the time to read out one frame.

Feature	Type	Description
	Point	
RowReadTime	Floating Point	Returns the time to read out one row.
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is enabled by default at power up for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
ShutterMode	Enumerated	Controls the behavior of the shutter. ShutterOutput can be triggered by setting AuxOutSourceTwo to ExternalShutterControl.
ShutterTransferTime	Floating Point	Sets the time at which the shutter will be opened before an exposure starts. Only available whenever the ShutterMode is set to 'Auto'.
SimplePreAmpGainControl	Enumerated	Alias for GainMode.
SoftwareTrigger	Command	Generates a software trigger in the camera. Used to trigger an exposure of a new frame on the camera whenever the trigger mode is set to *Software*.
SpuriousNoiseFilter	Boolean	Enables spurious noise filter.
TargetSensorTemperature	Floating Point	Configures the temperature to which the sensor will be cooled.
TemperatureControl	Enumerated	Allows the user to set the target temperature of the sensor based on a list of valid temperatures. Options: • -30.0
TemperatureStatus	Enumerated	Reports the current cooling state. [Read Only] Options: • **Cooler Off** – cooler has not been enabled. • **Cooling** – cooler is enabled but has not reached the set point for long enough. • **Stabilised** – sensor temperature has stabilized at the set point. • **Drift** – Cooler was previously stable but has drifted more the 2°C from set point. • **Not Stabilised** – Cooling has reached its target temperature but sensor temperature has not yet stabilised • **Fault** – Fault in cooling system. Check water supply or that air vents are not blocked. • **Sensor Over Temperature** – Temperature of sensor temperature has exceeded allowed limit. Check water supply or that air vents are not blocked.
TimestampClock	Integer	Reports the current value of the camera's internal timestamp clock. This same clock is used to timestamp images as they are acquired when the MetadataTimestamp feature is enabled. The clock is reset to zero when the camera is powered on and then runs continuously at the frequency indicated by the TimestampClockFrequency feature. The clock is 64-bits wide.
TimestampClock Frequency	Integer	Reports the frequency of the camera's internal timestamp clock in Hz.
TimestampClock Reset	Command	Resets the camera's internal timestamp clock to zero. As soon as the reset is complete the clock will begin incrementing from zero at the rate given by the TimestampClockFrequency feature.
TriggerMode	Enumerated	Configures the camera trigger mode. Currently supported options: • **Internal** – Camera will run at rate controlled by the FrameRate feature. • **Software** – Each exposure is triggered by the SoftwareTrigger feature. • **External** – Each exposure is triggered by an external trigger pulse. • **External Start** – Each series is triggered by an external trigger pulse. • **External Exposure –** Each exposure is controlled by high level of external trigger pulse

Feature	Type	Description
TriggerSource	Enumerated	Configures the source of the external trigger signal. • **D-Type Connector** - connector on the camera. • **Frame Grabber** – connector on the frame-grabber card. (Only available when using CXP interface)
VerticallyCentreAOI	Boolean	Vertically centres the AOI in the frame. With this enabled, AOITop will be disabled.

4.3.8. SIMCAM

Feature	Type	Description
AcquisitionStart	Command	Starts an acquisition.
AcquisitionStop	Command	Stops an acquisition.
AOIHBin	Integer	Configures the Horizontal Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIHeight	Integer	Configures the Height of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest.**
AOILeft	Integer	Configures the left hand coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest**.
AOITop	Integer	Configures the top coordinate of the sensor area of interest in sensor pixels. See **Section 4.7 Area of Interest.**
AOIVBin	Integer	Configures the Vertical Binning of the sensor area of interest. See **Section 4.7 Area of Interest**.
AOIWidth	Integer	Configures the Width of the sensor area of interest in super-pixels. See **Section 4.7 Area of Interest**.
CameraAcquiring	Boolean	Returns whether or not an acquisition is currently acquiring.
CameraModel	String	Returns the camera model.
CycleMode	Enumerated	Configures whether the camera will acquire a fixed length sequence or a continuous sequence. In Fixed mode the camera will acquire 'FrameCount' number of images and then stop automatically. In Continuous mode the camera will continue to acquire images indefinitely until the 'AcquisitionStop' command is issued. Options: • **Fixed** • **Continuous**
DeviceCount	Integer	Returns the number of cameras detected.
ElectronicShutteringMode	Enumerated	Configures which on-sensor electronic shuttering mode is used. • For pulsed or fast moving images *Global* shuttering is recommended. • For the highest frame rates and best noise performance *Rolling* is recommended. Options: • **Rolling** • **Global**
ExposureTime	Floating Point	The requested exposure time in seconds. *Note: In some modes the exposure time can also be modified while the acquisition is running.*
FanSpeed	Enumerated	Configures the speed of the fan in the camera. Options: • **Off** • **Low** • **On**
FrameCount	Integer	Configures the number of images to acquire in the sequence while **CycleMode** is **Fixed**. The value of FrameCount must be any value which is a multiple of AccumulateCount. This ensures the accumulation contains the correct number of frames. When this feature is unavailable then the camera

Feature	Type	Description
		does not currently support fixed length series, therefore you must explicitly abort the acquisition once you have acquired the amount of frames required.
FrameRate	Floating Point	Configures the frame rate in Hz at which each image is acquired during any acquisition sequence. This is the rate at which frames are acquired by the camera which may be different from the rate at which frames are delivered to the user. For example when AccumulateCount has a value other than 1, the apparent frame rate will decrease proportionally.
ImageSizeBytes	Integer	Returns the buffer size in bytes required to store the data for one frame. This will be affected by the Area of Interest size, binning and whether metadata is appended to the data stream.
LogLevel	Enumerated	Sets the level of logging used. Options: • **NONE** • **FATAL** • **ERROR** • **WARN** • **INFO** *(default)* • **DEBUG** • **VERB** The log level is set independently for each device library using the system handle. See **Opening a Camera Handle**. Changing the LogLevel changes the verbosity of output. The log file can be found in the temporary folder of the operating system. The default level can also be changed by setting an environment variable named after the library, e.g. atdevregcamloglevel, to one of the options above.
PixelCorrection	Enumerated	Configures the pixel correction to be applied. Options: • **Raw**
PixelEncoding	Enumerated	Configures the format of data stream. See **Section 4.5, Pixel Encoding**. Options: • **Mono8** • **RGB8Packed** • **Mono12** • **Mono12Packed** • **Mono12Coded** • **Mono12CodedPacked** • **Mono16** • **Mono22Parallel** • **Mono22PackedParallel** • **Mono32** See **Section 4.8, PixelEncoding and PreAmpGainControl** for the dependency between this feature and the PreAmpGainControl feature.
PixelHeight	Floating Point	Returns the height of each pixel in micrometers.
PixelReadoutRate	Enumerated	Configures the rate of pixel readout from the sensor. Options: • **550 MHz**
PixelWidth	Floating Point	Returns the width of each pixel in micrometers.
PreAmpGain	Enumerated	Configures the gain applied to the gain channel selected through the *Pre Amp Gain Selector* Feature. Options: • **x1**

Feature	Type	Description
		- **x2** - **x10** - **x30**
PreAmpGainChannel	Enumerated	Configures which pre amp gain channel(s) will be used for reading out the sensor. Options: - **High** - **Low** - **Both**
PreAmpGainSelector	Enumerated	Selects the gain channel that you subsequently wish to configure using the *Pre Amp Gain* Feature. Options: - **High** - **Low**
SensorCooling	Boolean	Configures the state of the sensor cooling. Cooling is disabled by default at power up and must be enabled for the camera to achieve its target temperature. The actual target temperature can be set with the TemperatureControl feature where available for example on the Neo camera.
SensorHeight	Integer	Returns the height of the sensor in pixels.
SensorTemperature	Floating Point	Read the current temperature of the sensor.
SensorWidth	Integer	Returns the width of the sensor in pixels.
SerialNumber	String	Returns the camera serial number.
SoftwareVersion	String	Returns the version of the SDK.
Synchronous Triggering	Boolean	Configures whether external triggers are synchronous with the read out of a sensor row. Asynchronous triggering may result in data corruption in the row being digitised when the triggers occurs.
TargetSensor Temperature	Floating Point	Configures the temperature to which the sensor will be cooled. To be used for cameras with no correction data (-50->25). Otherwise *TemperatureControl* should be used.
TriggerMode	Enumerated	Allows the user to configure the camera trigger mode at a high level. If the trigger mode is set to *Advanced* then the *Trigger Selector* and *Trigger Source* feature must also be set. Options: - **Internal** - **Software** - **External** - **External Start** - **External Exposure** - **Advanced**

4.4. IMAGE FORMAT

Images are presented to the application in the general format shown in **Figure 1**. Pixels are returned row by row starting from the top row and with the leftmost pixel being sent first in each row. The number of pixels in each row can be obtained from the **AOIWidth** feature and the number of rows in the image can be obtained from the **AOIHeight** feature.

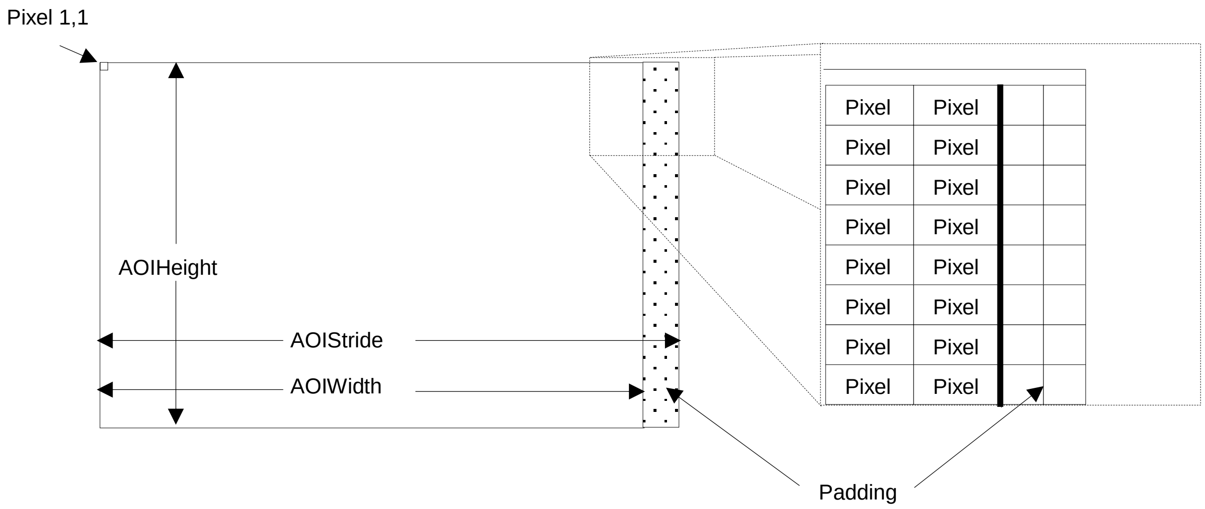
Pixel 1,1
AOIHeight
AOIStride
AOIWidth
Pixel
Padding

Figure 1: General Image Format

Stride

At the end of each row there may be additional padding bytes. This padding area does not contain any valid pixel data and should be skipped over when processing or displaying an image. This padding is necessary to ensure the image can be transferred over the interface between the camera and the PC and its size is dependent on the specific hardware interface being used as well as the current AOI settings. **Figure 2** shows what this padding looks like when viewing the raw data for an image in memory; in this example the pixels are 16-bit wide and the AOIWidth is 10.

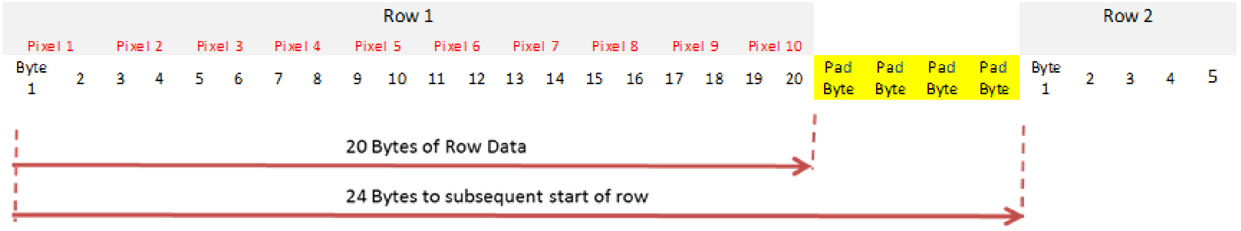
Row 1
Pixel 1
Pixel 2
Pixel 3
Pixel 4
Pixel 5
Pixel 6
Pixel 7
Pixel 8
Pixel 9
Pixel 10
Byte 1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Pad Byte
Pad Byte
Pad Byte
Pad Byte
Row 2
Byte 1
2
3
4
5
20 Bytes of Row Data
24 Bytes to subsequent start of row

Figure 2: Padding between image rows in memory

To ensure that your application will operate successfully with any hardware interface or AOI configuration, you should make use of the **AOIStride** feature to skip over the padding at the end of each row. The AOIStride feature represents the total number of bytes that each row of the image contains and includes the memory necessary for pixel data plus any padding at the end of the row. The stride should be used during processing of an image to obtain the memory address of each row relative to the previous row. See the example code below. Note that AOIStride is measured in bytes whereas AOIWidth is measured in pixels.

```
//Get the next image from the SDK
AT_WaitBuffer(&ImageBuffer, &ImageSize, AT_INFINITE);

//Retrieve the dimensions of the image
AT_GetInt(Hndl, L"AOIStride", &Stride);
AT_GetInt(Hndl, L"AOIWidth", &Width);
AT_GetInt(Hndl, L"AOIHeight", &Height);

for (AT_64 Row=0; Row < Height; Row++) {

  //Cast the raw image buffer to a 16-bit array.
  //...Assumes the PixelEncoding is 16-bit.
  unsigned short* ImagePixels = reinterpret_cast<unsigned short*>(ImageBuffer);

  //Process each pixel in a row as normal
  for (AT_64 Pixel=0; Pixel < Width; Pixel++) {
    SomeProcessing(ImagePixels[Pixel]);
  }

  //Use Stride to get the memory location of the next row.
  ImageBuffer += Stride;
}
```

ATUtility Library

The **ATUtility library** provided with the SDK contains functionality that can be used to strip the padding from an image. Once this is done the image can be processed without concern for padding. Stripping the padding from an image will however incur some processing overhead. See example code below and **Section 6.1, ATUTILITY**.

```
//Get the next image from the SDK
AT_WaitBuffer(&ImageBuffer, &ImageSize, AT_INFINITE);

//Retrieve the dimensions of the image
AT_GetInt(Hndl, L"AOIStride", &Stride);
AT_GetInt(Hndl, L"AOIWidth", &Width);
AT_GetInt(Hndl, L"AOIHeight", &Height);

unsigned short ImagePixels[Width*Height];

//Use an atutility function to strip padding from the image
AT_ConvertBuffer(ImageBuffer, ImagePixels, Width, Height,
                 Stride, L”Mono16”, L”Mono16”);

for (AT_64 Pixel=0; Pixel < Width*Height; Pixel++) {
  SomeProcessing(ImagePixels[Pixel]);
}
```

4.5. PIXEL ENCODING

There are several **Pixel Encoding** options available for the pixels in an image. Each of the formats is described below showing the pixel size, its layout in memory and sample C++ code for extracting pixel information out of the raw memory array.

In the descriptions below MSB refers to the most significant bits of the pixel, LSB refers to the least significant bits. ImageBuffer is the address of the start of the image in memory.

4.5.1. MONO12PACKED

12-bit Monochrome Data, stored by packing two adjacent pixels into three bytes.

Bits 11:4 (MSB)	***Bits 3:0 (LSB)***

ImageBuffer+0	**Pixel A (MSB)**	
ImageBuffer+1	**Pixel B (LSB)**	**Pixel A (LSB)**
ImageBuffer+2	**Pixel B (MSB)**	
ImageBuffer+3	**Pixel C (MSB)**	
ImageBuffer+4	**Pixel D (LSB)**	**Pixel C (LSB)**
ImageBuffer+5	**Pixel D (MSB)**	

```
PixelA = (ImageBuffer [0] << 4) + (ImageBuffer [1] & 0xF);
PixelB = (ImageBuffer [2] << 4) + (ImageBuffer [1] >> 4);
PixelC = (ImageBuffer [3] << 4) + (ImageBuffer [4] & 0xF);
PixelD = (ImageBuffer [5] << 4) + (ImageBuffer [4] >> 4);
```

4.5.2. MONO12

12-bit Monochrome Data, stored as 16-bit little-endian with zero padded upper bits.

Bits 11:8 (MSB)	***Bits 7:0 (LSB)***

ImageBuffer+0	**Pixel A (LSB)**	
ImageBuffer+1	**0**	**Pixel A (MSB)**
ImageBuffer+2	**Pixel B (LSB)**	
ImageBuffer+3	**0**	**Pixel B (MSB)**
ImageBuffer+4	**Pixel C (LSB)**	
ImageBuffer+5	**0**	**Pixel C (MSB)**

```
unsigned short* ImagePixels = reinterpret_cast<unsigned short*>(ImageBuffer);
PixelA = ImagePixels [0];
PixelB = ImagePixels [1];
PixelC = ImagePixels [2];
```

4.5.3. MONO16

16-bit Monochrome Data, stored as 16-bit little-endian.

Bits 15:8 (MSB)	***Bits 7:0 (LSB)***

ImageBuffer+0	**Pixel A (LSB)**
ImageBuffer+1	**Pixel A (MSB)**
ImageBuffer+2	**Pixel B (LSB)**
ImageBuffer+3	**Pixel B (MSB)**
ImageBuffer+4	**Pixel C (LSB)**

ImageBuffer+5	**Pixel C (MSB)**

```
unsigned short* ImagePixels = reinterpret_cast<unsigned short*>(ImageBuffer);
PixelA = ImagePixels [0];
PixelB = ImagePixels [1];
PixelC = ImagePixels [2];
```

4.5.4. MONO32

32-bit Monochrome Data, stored as 32-bit little-endian.

Bits 31:24 (MSB)	***Bits 23:16***	***Bits 15:8***	***Bits 7:0 (LSB)***

ImageBuffer+0	**Pixel A (LSB)**
ImageBuffer+1	**Pixel A**
ImageBuffer+2	**Pixel A**
ImageBuffer+3	**Pixel A (MSB)**
ImageBuffer+4	**Pixel B (LSB)**
ImageBuffer+5	**Pixel B**
ImageBuffer+6	**Pixel B**
ImageBuffer+7	**Pixel B (MSB)**

```
unsigned int* ImagePixels = reinterpret_cast<unsigned int*>(ImageBuffer);
PixelA = ImagePixels [0];
PixelB = ImagePixels [1];
```

4.5.5. MONO8 (LIMITED AVAILABILITY)

8-bit Monochrome Data

Bits 7:0

ImageBuffer+0	**Pixel A**
ImageBuffer+1	**Pixel B**
ImageBuffer+2	**Pixel C**
ImageBuffer+3	**Pixel D**

```
unsigned char* ImagePixels = reinterpret_cast<unsigned char*>(ImageBuffer);
PixelA = ImagePixels [0];
PixelB = ImagePixels [1];
PixelC = ImagePixels [2];
ImagePixels [3];
```

4.6. METADATA

Metadata can be enabled through the **MetadataEnable** Boolean feature. When metadata is enabled extra information will be appended onto each image by the camera. The **ImageSizeBytes** feature will update to include the extra memory required for the metadata. By default metadata is disabled.

The features used to configure Metadata are:

- MetadataEnable – Enable inclusion of metadata information in the data stream.
- MetadataFrameInfo – Enable inclusion of frame information in the data stream.
- MetadataTimestamp – Enable inclusion of timestamp information in the data stream.
- MetadataFrame – Enable inclusion of image data in the data stream, this will always be included if metadata is enabled.

With Metadata enabled the format of the image stream is shown below.

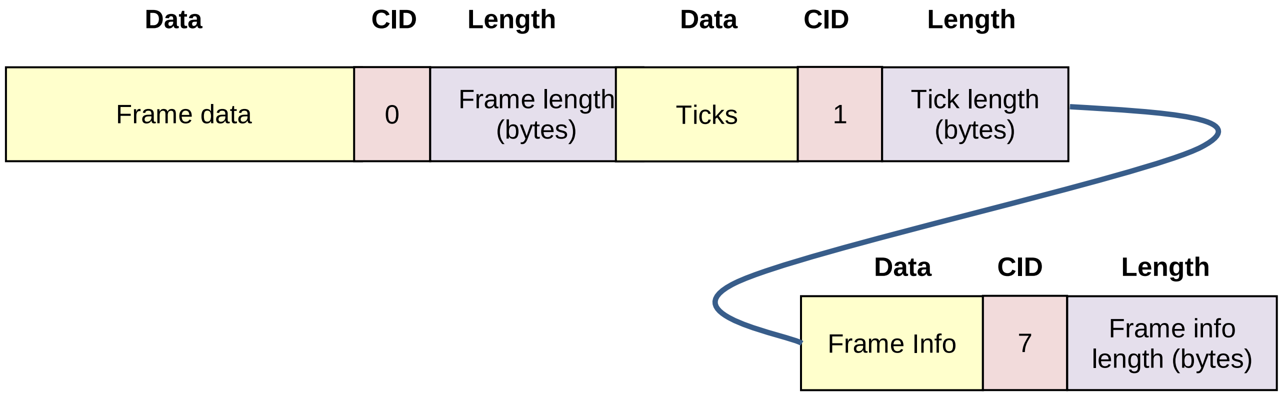

Figure 3: Format of the image stream with Metadata enabled.

Usage: Data, CID (Chunk ID), Length and then repeated as necessary.

Metadata is composed in blocks, with each block representing a particular type of metadata. Each block has an independent identifier called the **CID** (Chunk Identifier). As well as the CID, each block contains the actual metadata value and also a length field to facilitate parsing through the metadata blocks. The Length field in each block indicates the size of the metadata information plus the size of the CID. Note that the Length field is the last field in each metadata block. Parsing of metadata should be done in reverse, starting from the end of the data stream, and working back through the metadata until you reach the start of the data stream.

The three types of metadata block currently supported are **Frame (CID 0)**, **Timestamp (CID 1)** and **Frame Information (CID 7)**. The Frame metadata block simply contains the image data and is always enabled if metadata is enabled. The Timestamp metadata block contains a timestamp indicating the time at which the exposure for the frame was started. The Frame Information metadata block contains information on the structure of the image data.

All fields in the metadata blocks are stored little endian. i.e. least significant byte first.

The metadata format is described below:

Data
Data will vary depending on which CID it is. In this case it may be the actual image data or the timestamp information.

Chunk Identifier (CID)
A CID is used to label each block. Each CID is 4 bytes in length. The valid values for CID are shown below:

0 - Frame Data

- This represents the actual image.

1 – FPGA “Ticks”

- From camera power up, a 64 bit counter will count number of FPGA clocks or “Ticks”.
- The Ticks data will always be a 64 bit number – 8 bytes

7 – Frame Info

- Information about the frame including AOI, pixel encoding and stride.

Length

The length is a 4 byte number. This is where the length in bytes of the metadata block is stored. It includes the size of both the Data and the CID field.

Note

For the Frame Metadata block, Length is equal to the stride length x number of image rows, plus any padding at the end of the image plus the size of the CID field.

See the Tutorial section **Error! Reference source not found.** for example code showing how to configure and

Timestamp Frequency

The frequency of the timestamp clock can be retrieved through the TimestampClockFrequency feature.

Timestamp Clock

The current value of the timestamp clock can be read directly from the camera by accessing the TimestampClock feature. This can be used to synchronise the timestamp attached to each image with an absolute calendar time. To do this the program should first read the current time from the PC clock, then read the TimestampClock feature. This reference point can then be used to find out the absolute time at which any image was acquired. Alternatively the TimestampClockReset feature can be executed which will reset the timestamp clock to zero.

Frame Info

The frame info block takes the form:

Bits 63:48 (MSB)	***Bits 47:32***	***Bits 31:24***	***Bits 23:16***	***Bits 15:0 (LSB)***
AOI Height	***AOI Width***	***0***	***Pixel Encoding***	***Stride***

AOI Height, AOI Width and Stride are all 16 bit numbers. Pixel encoding takes the values:

- 0 for Mono16
- 1 for Mono12
- 2 for Mono12Packed.
- 3 for Mono32

4.7. AREA OF INTEREST

The **Area of Interest (AOI),** sometimes referred to as Region of Interest (ROI) is configured with the following features:

- AOIHBin, AOIVBin or AOIBinning
- AOIWidth
- AOILeft
- AOIHeight
- VerticallyCentreAOI
- AOITop

It is recommended that these features are configured in the order listed above as features towards the top of the list will override the values below them if the values are incompatible. Width depends on binning and left depends of width, therefore they should be set in the order: AOIHBin, AOIWidth, AOILeft. Similarly, height depends on binning and top depends on height, giving: AOIVBin, AOIHeight, AOITop. If the symmetric AOIBinning feature is used, it should be set before all others as both AOIWidth and AOIHeight depend on this feature.

Super-Pixels

A **super-pixel** is the result of combining multiple sensor pixels into a single data pixel by binning the values from each sensor pixel together. The amount of binning in each direction is configured either by setting the AOIHBin and AOIVBin features or by using the AOIBinning feature. The AOIWidth and AOIHeight features are set and retrieved in units of super-pixels Therefore, when binning is in use, the AOIWidth value will always indicate the number of data pixels that each row of the image data contains and not the number of pixels read off the sensor. The AOILeft and AOITop coordinates are specified in units of sensor pixels.

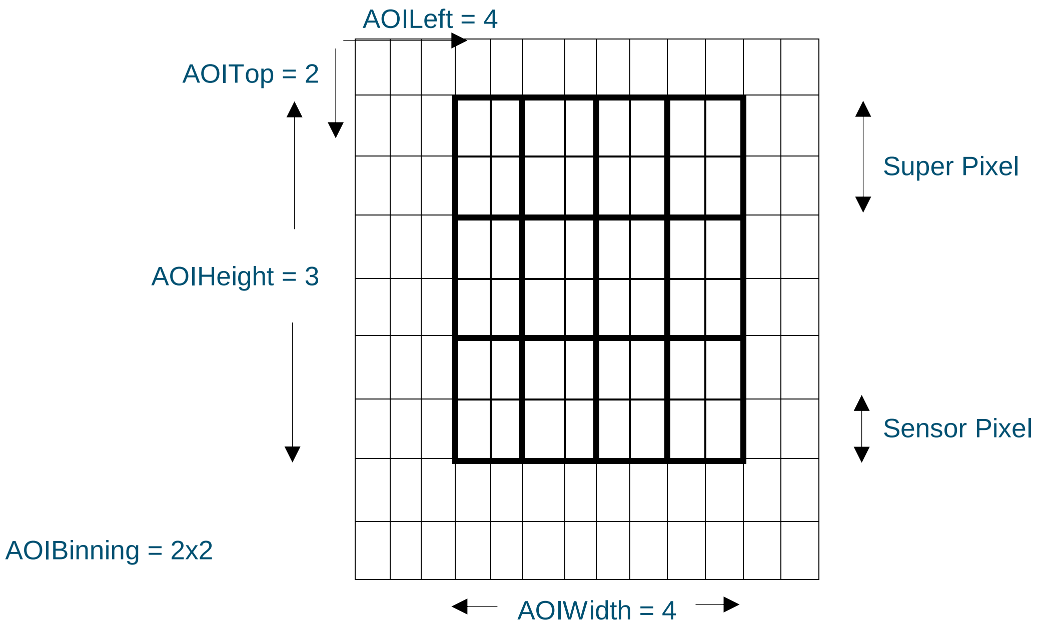

Figure 4: Configuring an AOI and presentation of Super-Pixels

Support for AOI Control

Some older versions of the Neo camera do not support full control over the AOI. In this case the FullAOIControl feature will return false.

If the FullAOIControl feature is not implemented or returns false for your camera then there are restrictions on the configurations of AOI's that can be used. The available AOI's are listed in the table below.

Width	Height	Top	Left(12bit)*	Left(16bit)**	X Centre (12 bit)	X Centre (16bit)
2592	2160	1	1	1	1297	1297
2544	2160	1	17	25	1289	1297
2064	2048	57	257	265	1289	1297
1776	1760	201	401	409	1289	1297
1920	1080	537	337	337	1297	1297
1392	1040	561	593	601	1289	1297
528	512	825	1025	1033	1289	1297
240	256	953	1169	1177	1289	1297
144	128	1017	1217	1225	1289	1297
2592	304	929	1	1	1297	1297

*12bit refers to mono12packed pixel encoding.
**16bit refers to mono12 or mono16 pixel encoding.

4.8. PIXELENCODING AND PREAMPGAINCONTROL

When changing PreAmpGainControl, the PixelEncoding feature will automatically adjust to a default setting for that PreAmpGainControl. For example, if the PreAmpGainControl feature is changed from '12-bit (low noise)' to '16-bit (low noise & high well capacity)' then the PixelEncoding feature will switch automatically to Mono16. This will only occur if the PixelEncoding is currently set to Mono12 or Mono12Packed. Similarly, PixelEncoding will switch automatically to either Mono12 or Mono12Packed when the PreAmpGainControl feature is changed from '16-bit (low noise & high well capacity)' to '12-bit (low noise)'. When PixelEncoding is set to Mono32, it does not change when PreAmpGainControl is changed.

4.9. SENSOR COOLING

It is important to cool the temperature of the sCMOS sensor to reduce the amount of noise in the images captured, see example below of the same image at different sensor temperatures. Sensor cooling can be set with the Boolean feature SensorCooling. The sensor temperature can then be set with the Enumerated feature, TemperatureControl and read using the Float feature SensorTemperature. To check the status of the cooling mechanism, read the TemperatureStatus feature. The possible status options are:

Cooler Off	Cooling has been disabled.
Stabilised	Temperature has stabilised at Target Temperature.
Cooling	Temperature is approaching Target Temperature.
Drift	Temperature has drifted outside Target Range having stabilised.
Not Stabilised	Temperature is within Target Range but has not yet stabilised.
Fault	Temperature has been outside Target Range for a long period of time.
Sensor Over Temperature	Temperature of sensor has exceed limit

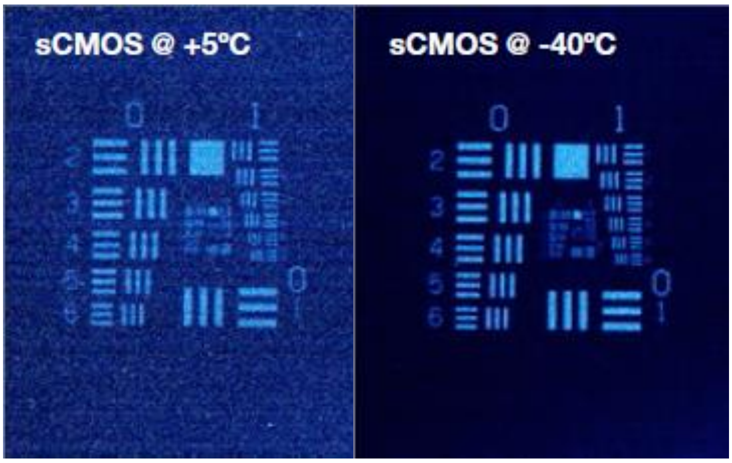

Figure 5: The effect of sensor cooling on noise: sCMOS sensor at +5°C and -40°C cooling.

```
double temperature = 0;
AT_SetBool(Hndl, L"SensorCooling", AT_TRUE);
AT_GetFloat(Hndl, L"SensorTemperature", &temperature);

cout << "Temperature: " << temperature << endl;

int temperatureCount = 0;
AT_GetEnumCount(Hndl, L"TemperatureControl", &temperatureCount);
AT_SetEnumIndex(Hndl, L"TemperatureControl", temperatureCount-1);

int temperatureStatusIndex = 0;
wchar_t* temperatureStatus[256];
AT_GetEnumIndex(Hndl, L"TemperatureStatus", &temperatureStatusIndex);
AT_GetEnumStringByIndex(Hndl, L"TemperatureStatus", temperatureStatusIndex,
temperatureStatus, 256);

while(wcscmp(L"Stabilised",temperatureStatus) != 0) {
  sleep(1);
  AT_GetEnumIndex(Hndl, L"TemperatureStatus", &temperatureStatusIndex);
  AT_GetEnumStringByIndex(Hndl, L"TemperatureStatus", temperatureStatusIndex,
temperatureStatus, 256);
  wcout << L"Temperature Status: " << temperatureStatus << endl;
}
```

```
cout << "Temperature Stabilised" << endl;
```

4.10. COMPARISON OF SDK2 AND SDK3

Action	SDK2	SDK3
Setting Exposure Time	`SetExposureTime(0.1);`	`AT_SetFloat(Hndl, L“ExposureTime”, 0.1);`
Using External Trigger	`SetTriggerMode(1);`	`AT_SetEnumString(Hndl, L“TriggerMode”, L“External”);`
Starting an Acquisition	`StartAcquisition();`	`AT_Command(Hndl, L“AcquisitionStart”);`

Determining if Pre Amp Gain is supported	`AndorCapabilities AndorCaps;`
	`GetCapabilties(&AndorCaps);`
	
	`if (AndorCaps.ulSetFunctions & AC_SETFUNCTION_PREAMPGAIN)`
	`  Implemented = 1;`
	`else`
	`  Implemented = 0;`
		`AT_IsImplemented(Hndl, L“PreAmpGain”, &Implemented);`

Getting Current Exposure Time	`GetAcquisitionTimings(&Exposure, &AccCycleTime, &KinCycleTime);`	AT_GetFloat(Hndl, L“ExposureTime”, &Exposure);
Setting Frame Rate (5fps)	`SetKineticCycleTime(0.2);`	`AT_SetFloat(Hndl, L“FrameRate”, 5.0);`
Getting the Serial Number	`GetCameraSerialNumber(&Serial);`	`AT_GetString(Hndl, L“SerialNumber”, Serial);`
Getting Current Trigger Mode	-- Not supported --	`AT_GetEnumIndex(Hndl, L“TriggerMode”, Mode);`

AOI Setup	`SetImage(1, 1, 256, 384, 512,768);`	`AT_SetInt(Hndl, L“AOIHBin”, 1);`
		`AT_SetInt(Hndl, L“AOIWidth”, 128);`
		`AT_SetInt(Hndl, L“AOILeft”, 256);`
		`AT_SetInt(Hndl, L“AOIVBin”, 1);`
		`AT_SetInt(Hndl, L“AOIHeight”, 256);`
		`AT_SetInt(Hndl, L“AOITop”, 512);`

Opening Second device	`GetCameraHandle(1, &SecondHandle)` `SetCurrentCamera(SecondHandle);`	`AT_Open(1, &SecondHandle);`
Getting Limits	`GetMinimumNumberInSeries(&Min)` -- Not supported --	`AT_GetIntMin(Hndl, L“FrameCount”, &Min);` `AT_GetIntMax(Hndl, L“FrameCount”, &Max);`

5. TUTORIAL

This section explains how to develop a program to use the API to communicate with a sCMOS camera. It points out the critical parts that the program must have and provides a foundation to build more complicated programs in the future.

The first thing that must be done is to add the appropriate library file to the project. During the SDK3 installation a 32-bit or 64-bit version of libatcore.so is copied to `<installation folder>/lib` depending on your OS type. You should also add `<installation folder>/inc` installation directory to your project. The atcore.h file from this directory will need to be included in the main project file.

The very first API call must be AT_InitialiseLibrary, and the very last call must be AT_FinaliseLibrary. These functions will prepare the API for use and free resources when no longer needed.

```
AT_InitialiseLibrary( );

AT_FinaliseLibrary( );
```

Every function will return an error code when called. It is recommended that a user check every return code before moving on to the next statement. If the AT_InitialiseLibrary function call fails, there is no point continuing on with the program as every proceeding function call will also fail.
Every function call will return an Integer and each return code that could possibly be returned is listed in the atcore.h file and documented in **Section 2.4 Error Codes**.

So now the program becomes:

```
int i_returnCode;
i_returnCode = AT_InitialiseLibrary( );
if (i_returnCode == AT_SUCCESS) {
  //continue with program
}
i_returnCode = AT_ FinaliseLibrary( );
if (i_returnCode != AT_SUCCESS) {
  //Error FinaliseLibrary
```

Error Checking

It is highly recommended that you check return codes from every function in case of error. For the purpose of this tutorial the error checking will be kept to a minimum to reduce the length of the program.

The next stage of the program is to get a handle to the camera that is to be controlled. This is done with the AT_Open function and there is a corresponding AT_Close function to release the camera handle. A camera index is passed into the AT_Open function specifying which camera you wish to open.

Now the program is:

```
int i_returnCode;
AT_H Hndl;
int i_cameraIndex = 0;
i_returnCode = AT_InitialiseLibrary( );
if (i_returnCode == AT_SUCCESS) {
  i_returnCode = AT_Open ( i_cameraIndex, &Hndl );
  if (i_returnCode == AT_SUCCESS) {
    //continue on with program
    //.........
    i_returnCode = AT_Close ( Hndl );
    if (i_returnCode != AT_SUCCESS) {
      // error closing handle
    }
  }
}
i_returnCode = AT_FinaliseLibrary( );
if (i_returnCode != AT_SUCCESS) {
  //Error FinaliseLibrary
}
```

From here on example code will assume that we have successfully got a handle to our camera, following on from the previous code. Now we will add code to the program to modify and view the “ExposureTime” setting of the camera.

It is recommended that any features that need to set or view that they should have their accessibility checked first. Every feature can be checked for being implemented (AT_IsImplemented), writable (AT_IsWritable), readable (AT_IsReadable) and read only (AT_IsReadOnly)

For this tutorial it is assumed that the ‘Exposure Time’ feature is writable and readable, to reduce code complexity.

```
double d_newExposure = 0.02;
i_returnCode = AT_SetFloat ( Hndl,  L"ExposureTime", d_newExposure);
if (i_returnCode == AT_SUCCESS) {
  //it has been set

}
```

NOTE
It is recommended that when using the AT_SetFloat functions that an AT_GetFloat is done afterwards to get the actual value that the camera will use as it may not be exactly what was set.

In order to ensure a low noise level in the images we must enable the sensor cooling mechanism. Cooling is off by default, to activate the mechanism use the Boolean feature SensorCooling. Then set the temperature with the TemperatureControl feature and check the status using TemperatureStatus.

```
double temperature = 0;
AT_SetBool(Hndl, L"SensorCooling", AT_TRUE);

int temperatureCount = 0;
AT_GetEnumCount(Hndl, L"TemperatureControl", &temperatureCount);
AT_SetEnumIndex(Hndl, L"TemperatureControl", temperatureCount-1);

int temperatureStatusIndex = 0;
wchar_t* temperatureStatus[256];
do {
      AT_GetEnumIndex(Hndl, L"TemperatureStatus", &temperatureStatusIndex);
      AT_GetEnumStringByIndex(Hndl, L"TemperatureStatus", temperatureStatusIndex,
          temperatureStatus, 256);

}
while(wcscmp(L"Stabilised",temperatureStatus) != 0);
```

This stage shows how to get data from a single acquisition. In preparation the program must allocate memory to store the acquired image in. To get the size of memory to be declared use the integer “ImageSizeBytes” feature.
No error checking will be shown in the following example to help with clarity- but it is recommended that in final code all return codes are checked.

Buffer Byte Alignment
The memory declared to hold the acquisition should be aligned on an 8 byte boundary, this helps with system performance and also prevent any alignment fault that could occur. The pucAlignedBuffer variable in the following code shows how to enforce 8 byte alignment.

```
AT_64 ImageSizeBytes;
AT_GetInt( Hndl, L"ImageSizeBytes", &ImageSizeBytes);
//cast so that the value can be used in the AT_QueueBuffer function
int i_imageSize = static_cast<int>(ImageSizeBytes);

unsigned char* uc_Buffer = NULL;
gblp_Buffer = new unsigned char[i_imageSize+8]; // Add 8 to allow data alignment

// Adjust pointer so that it falls on an 8-byte boundary
unsigned char* pucAlignedBuffer = reinterpret_cast<unsigned char*>(
(reinterpret_cast<unsigned long>( gblp_Buffer ) + 7 ) & ~0x7);
```

The next stage is to let SDK know what memory to use for the upcoming acquisition. This is done with the AT_QueueBuffer API function call. Multiple buffers can be queued to the SDK before an acquisition starts if you are acquiring a sequence of frames. For now we will assume that only one frame is required.

```
AT_QueueBuffer(Hndl,  pucAlignedBuffer, ImageSizeBytes);
```

Now start the acquisition with the Command “AcquisitionStart”. The command to stop the acquisition is "AcquisitionStop".

```
AT_Command(Hndl, L"AcquisitionStart");
//get the data
AT_Command(Hndl, L"AcquisitionStop");
```

When the acquisition has been started there is a function AT_WaitBuffer that can be used to put the calling thread to block until the current image has been captured and is ready for the program to use. A time out value in milliseconds is also specified to AT_WaitBuffer to force it to return if the acquisition has not occurred in that time frame.

```
unsigned char* pBuf;
int BufSize;
AT_WaitBuffer(Hndl, &pBuf, &BufSize, 10000);
```

It is vital to check the return code from the AT_WaitBuffer function before processing the returned buffer. The pBuf return from this function should be the same pointer to the one that was queued in the AT_QueueBuffer function. If all has been successful, then the data in the array pointed to by pBuf will contain the acquired image. The “Pixel Encoding” feature should then be checked to confirm the format of the datastream. See the **Features** Section for a more complete explanation.

After the acquisition is complete any buffers queued up with the AT_QueueBuffer command but not yet returned from AT_WaitBuffer, need to be released by the SDK or else the next acquisition will use them.-Failure to call the AT_Flush function after an acquisition will not only result in both queues not being cleared, but may lead to undefined behaviour.

```
AT_Flush(Hndl);
```

The final code for this tutorial can be seen in section “

5.1. FURTHER EXAMPLES

5.1.1. INITIALISE LIBRARY AND OPEN CAMERA

```
//InitialiseLibrary must be the first function call made by an application before
accessing other functions
AT_InitialiseLibrary();

//Declare an Andor Device Handle for referencing device later
AT_H Handle;

//Open the first device (Device 0)
AT_Open(0, &Handle);

//Close the device when finished using it
AT_Close(Handle);

//Call FinaliseLibrary when all access to API is complete
AT_FinaliseLibrary();
```

5.1.2. SIMPLE SINGLE FRAME ACQUISITION

```
AT_InitialiseLibrary();
AT_H Handle;
AT_Open(0, &Handle);

//Set the exposure time for this camera to 10 milliseconds

AT_SetFloat(Handle, L”ExposureTime”, 0.01);

//Get the number of bytes required to store one frame
AT_64 ImageSizeBytes;
AT_GetInt(Handle, L"ImageSizeBytes", &ImageSizeBytes);

int BufferSize = static_cast<int>(ImageSizeBytes);

//Allocate a memory buffer to store one frame
unsigned char* UserBuffer = new unsigned char[BufferSize];

//Pass this buffer to the SDK
AT_QueueBuffer(Handle, UserBuffer, BufferSize);

//Start the Acquisition running
AT_Command(Handle, L"AcquisitionStart");

//Sleep in this thread until data is ready, in this case set
//the timeout to infinite for simplicity
unsigned char* Buffer;

AT_WaitBuffer(Handle, &Buffer, &BufferSize, AT_INFINITE);

//Stop the acquisition
AT_Command(Handle, L"AcquisitionStop");
AT_Flush(Handle);


//Application specific data processing goes here..

//Free the allocated buffer
```

```
delete [] UserBuffer;

AT_Close(Handle);
AT_Close(Handle);
AT_FinaliseLibrary();
```

5.1.3. USING A FEATURE

```
AT_InitialiseLibrary();
AT_H Handle;
AT_Open(0, &Handle);

AT_BOOL Implemented;
//To determine if Exposure time is implemented by the camera
AT_IsImplemented(Handle, L"ExposureTime", &Implemented);

AT_BOOL ReadOnly;
//To determine if Exposure time a Read Only Feature
AT_IsReadOnly(Handle, L"ExposureTime", &ReadOnly);

if (Implemented==AT_TRUE) {
  //Get the Limits for Exposure Time
  double Min, Max;
  AT_GetFloatMin(Handle, L"ExposureTime", &Min);
  AT_GetFloatMax(Handle, L"ExposureTime", &Max);

  //Get the current accessibility
  AT_BOOL Writable, Readable;
  AT_IsWritable(Handle, L"ExposureTime", &Writable);
  AT_IsReadable(Handle, L"ExposureTime", &Readable);

  if (Readable==AT_TRUE) {
    //To get the current value of Exposure time in
    //microseconds
    double ExposureTime;
    AT_GetFloat(Handle, L"ExposureTime", &ExposureTime);
  }

  if (Writable==AT_TRUE) {
    //To set the value of Exposure Time to 10 seconds

    AT_SetFloat(Handle, L"ExposureTime", 0.00001);
  }

}

AT_Close(Handle);
AT_FinaliseLibrary();
```

5.1.4. CIRCULAR BUFFER

```
AT_InitialiseLibrary();
AT_H Handle;
AT_Open(0, &Handle);

AT_SetFloat(Handle, L"ExposureTime", 0.01);
```

```
AT_64 ImageSizeBytes;
AT_GetInt(Handle, L"ImageSizeBytes", &ImageSizeBytes);

int BufferSize = static_cast<int>(ImageSizeBytes);

//Declare the number of buffers and the number of frames interested in
int NumberOfBuffers = 10;
int NumberOfFrames = 100;

//Allocate a number of memory buffers to store frames
unsigned char** AcqBuffers = new unsigned char*[NumberOfBuffers];
unsigned char** AlignedBuffers = new unsigned char*[NumberOfBuffers];
for (int i=0; i < NumberOfBuffers; i++) {
  AcqBuffers[i] = new unsigned char[BufferSize + 7];
  AlignedBuffers[i] = reinterpret_cast<unsigned char*>((reinterpret_cast<unsigned
long>(AcqBuffers[i% NumberOfBuffers]) + 7) & ~7);
}

//Pass these buffers to the SDK
for(int i=0; i < NumberOfBuffers; i++)  {
  AT_QueueBuffer(Handle, AlignedBuffers[i], BufferSize);
}

//Set the camera to continuously acquires frames
AT_SetEnumString(Handle, L"CycleMode", L"Continuous");

//Start the Acquisition running
AT_Command(Handle, L"AcquisitionStart");

//Sleep in this thread until data is ready, in this case set
//the timeout to infinite for simplicity
unsigned char* pBuf;
int BufSize;
for (int i=0; i < NumberOfFrames; i++) {
  AT_WaitBuffer(Handle, &pBuf, &BufSize, AT_INFINITE);

  //Application specific data processing goes here..

  //Re-queue the buffers
  AT_QueueBuffer(Handle, AlignedBuffers[i%NumberOfBuffers], BufferSize);
}

//Stop the acquisition
AT_Command(Handle, L"AcquisitionStop");
AT_Flush(Handle);

//Application specific data processing goes here..

//Free the allocated buffer
for (int i=0; i < NumberOfBuffers; i++) {
  delete[] AcqBuffers[i];
}
delete[] AlignedBuffers;
delete[] AcqBuffers;

AT_Close(Handle);
AT_FinaliseLibrary();
```

5.1.5. PIXEL ENCODING

```
AT_InitialiseLibrary();
AT_H Handle;
AT_Open(0, &Handle);

//Set the pixel Encoding to the desired settings Mono16 Data

AT_SetEnumString(Handle, L"PixelEncoding", L"Mono16");

AT_64 ImageSizeBytes;
AT_GetInt(Handle, L"ImageSizeBytes", &ImageSizeBytes);

int BufferSize = static_cast<int>(ImageSizeBytes);
unsigned char* UserBuffer = new unsigned char[BufferSize];

AT_QueueBuffer(Handle, UserBuffer, BufferSize);
AT_Command(Handle, L"AcquisitionStart");

unsigned char* Buffer;
AT_WaitBuffer(Handle, &Buffer,  &BufferSize, AT_INFINITE);

//Stop the acquisition
AT_Command(Handle, L"AcquisitionStop");
AT_Flush(Handle);

delete [] UserBuffer;

AT_Close(Handle);
AT_FinaliseLibrary();
```

5.1.6. CALL-BACKS

```
//This example will demonstrate how to setup, register and unregister a call-back
//Tests of the correct call-back context and a count of the number of updates received
are provided

AT_H Handle;
int g_iCallbackCount = 0;
int g_iCallbackContext = 0;

int AT_EXP_CONV Callback(AT_H Hndl, const AT_WC* Feature, void* Context)
{
  //Application specific call-back handling should go here
  g_iCallbackCount++;
  g_iCallbackContext = *reinterpret_cast<int*>(Context);
  return AT_CALLBACK_SUCCESS;
}

int main(int argc, char* argv[])
{
  AT_InitialiseLibrary();
  AT_Open(0, &Handle);

  //Set the call-back context, context values can be defined on per application basis
  int i_callbackContext = 5;
  //Reset the call-back count
  //Only required for the purposes of this example to show the call-back has been
received
  g_iCallbackCount = 0;

  //Register a call-back for the given feature
  AT_RegisterFeatureCallback(Handle, L"PixelReadoutRate", Callback,
(void*)&i_callbackContext);

  //Set the feature in order to trigger the call-back
  AT_SetEnumIndex(Handle, L"PixelReadoutRate", 0);

  // Application specific code should go here
  //For this example we shall check that the call-back has been successful
  if (g_iCallbackCount==0 || g_iCallbackContext != i_callbackContext) {
    //Deal with failed call-back
  }

  //Unregister the call-back, no more updates will be received
  AT_UnregisterFeatureCallback(Handle, L"PixelReadoutRate", Callback,
(void*)&i_callbackContext);

  AT_Close(Handle);
  AT_FinaliseLibrary();
}
```

6. ADDITIONAL LIBRARIES

This section describes the additional libraries that are provided as part of the SDK3 installation. These additional libraries are not required to use the SDK; they provide additional ease of use functionality.

6.1. ATUTILITY

This library provides additional general utility functions and can be used with an Embarcadero or Microsoft compatible compiler. To use this library perform the following steps (assumes you have already setup your project as described in **Section Getting Started**.

1. Add "atutility.h" to the list of header files included in your application source file.
2. Add the appropriate library from the SDK3 installation directory to your project.
    - atutility.lib for the Embarcadero compiler
    - atutilitym.lib for the Microsoft compiler
3. Copy the “atutility.dll” from the SDK3 installation directory to the directory that the executable is going to run from.

6.2. AT_INITIALISEUTILITYLIBRARY

```
int AT_InitialiseUtilityLibrary ()
```

This function is used to initialize the utility library. It must be called before any utility functions can be called. There are no parameters expected.

The following is a brief explanation of the error codes that can be returned by the function:

Error Code	Description
AT_SUCCESS (0)	The library has been initialised successfully.

6.3. AT_FINALISEUTILITYLIBRARY

```
int AT_FinaliseUtilityLibrary ()
```

This function is used to close the utility library. It must be called before the user application completes to clean up internal resources. There are no parameters expected.

The following is a brief explanation of the error codes that can be returned by the function:

Error Code	Description
AT_SUCCESS (0)	The library has been closed successfully.

6.4. AT_CONVERTBUFFER

```
int AT_ConvertBuffer(AT_U8* inputBuffer, AT_U8* outputBuffer, AT_64 width, AT_64
height, AT_64 stride, const AT_WC* inputPixelEncoding, const AT_WC*
outputPixelEncoding)
```

This function is used to convert a buffer from one pixel encoding to another pixel encoding. For an explanation of the different encoding types see the pixel encoding feature description in **Section 4.3 Feature Reference**
. The converted image data will not contain any metadata therefore, if you require access to the metadata you will have to extract this from the input buffer as described in **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
outputBuffer	This is a pointer to the buffer that you want the converted data to be stored in. This should be large enough to hold the converted image i.e. width x height x bytes/pixel for the output pixel encoding (Mono16 = 2, Mono32 = 4).
width	This is the width of the image stored in the input buffer in pixels. Can be determined by using the SDK3 integer feature "AOI Width".
height	This is the height of the image stored in the input buffer in pixels. Can be determined by using the SDK3 integer feature "AOI Height".
stride	This is the number of bytes/line for the image stored in the input buffer. Can be determined by using the SDK3 integer feature "AOI Stride".
inputPixelEncoding	This is the pixel encoding that was used to create the image stored in the input buffer. The valid values that can be used are: • Mono12 • Mono12Packed • Mono16 • Mono32
outputPixelEncoding	This is the pixel encoding that will be used to store the image in the output buffer. The valid values that can be used are: • Mono16 • Mono32

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_NOTINITIALISED (1)`	The library has not been initialised.
`AT_ERR_INVALIDINPUTPIXELENCODING (1003)`	The input pixel encoding is not valid.
`AT_ERR_INVALIDOUTPUTPIXELENCODING (1002)`	The output pixel encoding is not valid.

6.5. CONVERT BUFFER EXAMPLE

The following simple console application shows how to use the AT_ConvertBuffer function to convert from “Mono12Packed” image data to “Mono16” image data:

```
#include "atcore.h"
#include "atutility.h"


int main(int argc, char* argv[])
{
  int i_retCode;
  i_retCode = AT_InitialiseLibrary();
  if (i_retCode == AT_SUCCESS) {
    i_retCode = AT_InitialiseUtilityLibrary ();
    if (i_retCode == AT_SUCCESS) {
      AT_64 iNumberDevices = 0;
      AT_GetInt(AT_HANDLE_SYSTEM, L"Device Count", &iNumberDevices);
      if (iNumberDevices > 0) {
        AT_H Hndl;
        i_retCode = AT_Open(0, &Hndl);
        if (i_retCode == AT_SUCCESS) {
          AT_SetEnumeratedString(Hndl, L"Pixel Encoding", L"Mono12Packed");
          AT_SetFloat(Hndl, L"Exposure Time", 0.01);

          double temperature = 0;
          AT_SetBool(Hndl, L"SensorCooling", AT_TRUE);

          int temperatureCount = 0;
          AT_GetEnumCount(Hndl, L"TemperatureControl", &temperatureCount);
          AT_SetEnumIndex(Hndl, L"TemperatureControl", temperatureCount-1);

          int temperatureStatusIndex = 0;
          wchar_t temperatureStatus[256];
          AT_GetEnumIndex(Hndl, L"TemperatureStatus", &temperatureStatusIndex);
          AT_GetEnumStringByIndex(Hndl, L"TemperatureStatus", temperatureStatusIndex,
temperatureStatus, 256);

          while(wcscmp(L"Stabilised",temperatureStatus) != 0) {
            AT_GetEnumIndex(Hndl, L"TemperatureStatus", &temperatureStatusIndex);
            AT_GetEnumStringByIndex(Hndl, L"TemperatureStatus", temperatureStatusIndex,
temperatureStatus, 256);
          }

          //Get the number of bytes required to store one frame
          AT_64 iImageSizeBytes;
          AT_GetInt(Hndl, L"Image Size Bytes", &iImageSizeBytes);
          int iBufferSize = static_cast<int>(iImageSizeBytes);

          //Allocate a memory buffer to store one frame
          unsigned char* UserBuffer = new unsigned char[iBufferSize];

          AT_QueueBuffer(Hndl, UserBuffer, iBufferSize);
          AT_Command(Hndl, L"Acquisition Start");

          unsigned char* Buffer;
          if (AT_WaitBuffer(Hndl, &Buffer,  &iBufferSize, 10000) == AT_SUCCESS){
            //Unpack the 12 bit packed data
            AT_64 ImageHeight;
            AT_GetInt(Hndl, L"AOI Height", &ImageHeight);
            AT_64 ImageWidth;
            AT_GetInt(Hndl, L"AOI Width", &ImageWidth);
            AT_64 ImageStride;
            AT_GetInt(Hndl, L"AOI Stride", &ImageStride);
            unsigned short* unpackedBuffer =
                             new unsigned short[ImageHeight*ImageWidth];
            AT_ConvertBuffer(Buffer,
                           reinterpret_cast<unsigned char*>(unpackedBuffer),
                           ImageWidth, ImageHeight,
               ImageStride, L"Mono12Packed", L"Mono16");

            //  process unpacked image data
```

```
                delete[] unpackedBuffer;
              }
              AT_Command(Hndl, L"Acquisition Stop");
              AT_Flush(Hndl);
              delete[] UserBuffer;
            }
            AT_Close(Hndl);
          }
        }
      }
      AT_FinaliseLibrary();
      AT_FinaliseUtilityLibrary();
      return 0;
    }
```

6.6. AT_CONVERTBUFFERUSINGMETADATA

```
int AT_ConvertBufferUsingMetaData(AT_U8* inputBuffer, AT_U8* outputBuffer, AT_64
imagesizebytes, const AT_WC* outputPixelEncoding)
```

This function is used to convert a buffer from the input pixel encoding to another pixel encoding. For an explanation of the different encoding types see the pixel encoding feature description in the **Section 4.3 Feature Reference** . The converted image data will not contain any metadata therefore, if you require access to the metadata you will have to extract this from the input buffer as described in **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
outputBuffer	This is a pointer to the buffer that you want the converted data to be stored in. This should be large enough to hold the converted image i.e. width x height x bytes/pixel for the output pixel encoding (Mono16 = 2, Mono32 = 4).
imagesizebytes	This is the size of the image stored in the input buffer in bytes. Can be determined by using the SDK3 integer feature "ImageSizeBytes".
outputPixelEncoding	This is the pixel encoding that will be used to store the image in the output buffer. The valid values that can be used are: • Mono16 • Mono32

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_NOTINITIALISED (1)`	The library has not been initialised.
`AT_ERR_INVALIDOUTPUTPIXELENCODING (1002)`	The output pixel encoding is not valid.
`AT_ERR_INVALIDMETADATAINFO (1004)`	The input buffer does not include metadata. This may be due to the system not supporting this option or it not being activated.

6.7. AT_GETWIDTHFROMMETADATA

```
int AT_GetWidthFromMetadata(AT_U8* inputBuffer, AT_64 imagesizebytes, AT_64& width)
```

This function is used to retrieve the width of the image stored in the input buffer from the metadata of the image. For more information on the metadata, see the **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_NOTINITIALISED (1)`	The library has not been initialised.
`AT_ERR_INVALIDOUTPUTPIXELENCODING (1002)`	The output pixel encoding is not valid.
`AT_ERR_INVALIDMETADATAINFO (1004)`	The input buffer does not include metadata. This may be due to the system not supporting this option or it not being activated.

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_METADATANOTFOUND (1006)`	The input buffer does not include the required metadata. This may be due to the system not supporting this option or it not being activated.

6.8. AT_GETHEIGHTFROMMETADATA

```
int AT_GetHeightFromMetadata(AT_U8* inputBuffer, AT_64 imagesizebytes, AT_64& height)
```

This function is used to retrieve the height of the image stored in the input buffer from the metadata of the image. For more information on the metadata, see the **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
imagesizebytes	This is the size of the image stored in the input buffer in bytes. Can be determined by using the SDK3 integer feature "ImageSizeBytes".
height	This is a reference to the variable that will contain the height of the image stored in the input buffer in pixels.

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_METADATANOTFOUND (1006)`	The input buffer does not include the required metadata. This may be due to the system not supporting this option or it not being activated.

6.9. AT_GETSTRIDEFROMMETADATA

```
int AT_GetStrideFromMetadata(AT_U8* inputBuffer, AT_64 imagesizebytes, AT_64& stride)
```

This function is used to retrieve the number of bytes/line for the image stored in the input buffer from the metadata of the image. For more information on the metadata, see the **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
imagesizebytes	This is the size of the image stored in the input buffer in bytes. Can be determined by using the SDK3 integer feature "ImageSizeBytes".
pixelEncoding	This is the pixel encoding that was used to create the image stored in the input buffer. The valid values that can be used are: • Mono12 • Mono12Packed • Mono16 • Mono32
pixelEncodingSize	This is the number of characters that the pixelEncoding variable can contain.

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_METADATANOTFOUND (1006)`	The input buffer does not include the required metadata. This may be due to the system not supporting this option or it not being activated.

6.10. AT_GETPIXELENCODINGFROMMETADATA

```
int AT_GetPixelEncodingFromMetadata(AT_U8* inputBuffer, AT_64 imagesizebytes, AT_WC*
pixelEncoding, AT_U8 pixelEncodingSize)
```

This function is used to retrieve the pixel encoding of the image stored in the input buffer from the metadata of the image. For an explanation of the different encoding types see the pixel encoding feature description in the **Section 4.3 Feature Reference.** For more information on the metadata, see the **Section 4.6 Metadata.** The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
imagesizebytes	This is the size of the image stored in the input buffer in bytes. Can be determined by using the SDK3 integer feature "ImageSizeBytes".
pixelEncoding	This is the pixel encoding that was used to create the image stored in the input buffer. The valid values that can be used are: • Mono12 • Mono12Packed • Mono16

	- Mono32
pixelEncodingSize	This is the number of characters that the pixelEncoding variable can contain.

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_METADATANOTFOUND (1006)`	The input buffer does not include the required metadata. This may be due to the system not supporting this option or it not being activated.

6.11. AT_GETTIMESTAMPFROMMETADATA

```
int AT_GetTimeStampFromMetadata(AT_U8* inputBuffer, AT_64 imagesizebytes, AT_64&
timeStamp)
```

This function is used to retrieve the timestamp of the image stored in the input buffer from the metadata of the image. For more information on the metadata, see the **Section 4.6 Metadata**. The following table provides a brief description of the parameters.

Parameter	Description
inputBuffer	This is a pointer to the input buffer that you want to convert.
imagesizebytes	This is the size of the image stored in the input buffer in bytes. Can be determined by using the SDK3 integer feature "ImageSizeBytes".
timeStamp	This is a reference to the variable that will contain the timestamp of the image stored in the input buffer in pixels.

The following table provides a brief description of the error codes that can be returned by the function:

Error Code	Description
`AT_SUCCESS (0)`	The function call has been successful.
`AT_ERR_METADATANOTFOUND (1006)`	The input buffer does not include the required metadata. This may be due to the system not supporting this option or it not being activated.

APPENDIX A sCMOS Feature Quick Reference

Feature	Type
AccumulateCount	Integer
AcquisitionStart	Command
AcquisitionStop	Command
AlternatingReadoutDirection	Boolean
AOIBinning	Enumerated
AOIHBin	Integer
AOIHeight	Integer
AOILayout	Enumerated
AOILeft	Integer
AOIStride	Integer
AOITop	Integer
AOIVBin	Integer
AOIWidth	Integer
AuxiliaryOutSource	Enumerated
AuxOutSourceTwo	Enumerated
Baseline	Integer
BitDepth	Enumerated
BufferOverflowEvent	Integer
BytesPerPixel	Floating Point
CameraAcquiring	Boolean
CameraModel	String
CameraName	String
CameraPresent	Boolean
ControllerID	String
FrameCount	Integer
CycleMode	Enumerated
DeviceCount	Integer
ElectronicShutteringMode	Enumerated
EventEnable	Boolean
EventsMissedEvent	Integer
EventSelector	Enumerated
ExposedPixelHeight	Integer
ExposureTime	Floating Point
ExposureEndEvent	Integer
ExposureStartEvent	Integer
External Trigger Delay	Floating Point
FanSpeed	Enumerated
FastAOIFrameRateEnable	Boolean
FirmwareVersion	String
FrameRate	Floating Point
FullAOIControl	Boolean
ImageSizeBytes	Integer
InterfaceType	String
IOInvert	Boolean
IOSelector	Enumerated
LineScanSpeed	Floating Point
LUTIndex	Integer
LUTValue	Integer
MaxInterfaceTransferRate	Floating Point
MetadataEnable	Boolean
MetadataTimestamp	Boolean
MetadataFrame	Boolean
MultitrackBinned	Boolean
MultitrackCount	Integer
MultitrackEnd	Integer
MultitrackSelector	Integer
MultitrackStart	Integer
Overlap	Boolean
PixelEncoding	Enumerated

Feature	Type
PixelHeight	Floating Point
PixelReadoutRate	Enumerated
PixelWidth	Floating Point
PreAmpGainControl	Enumerated
ReadoutTime	Floating Point
RollingShutterGlobalClear	Boolean
RowNExposureEndEvent	Integer
RowNExposureStartEvent	Integer
RowReadTime	Floating Point
ScanSpeedControlEnable	Boolean
SensorCooling	Boolean
SensorHeight	Integer
SensorReadoutMode	Enumerated
SensorTemperature	Floating Point
SensorWidth	Integer
SerialNumber	String
ShutterOutputMode	Enumerated
SimplePreAmpGainControl	Enumerated
Shutter Transfer Time	Floating Point
SoftwareTrigger	Command
StaticBlemishCorrection	Boolean
SpuriousNoiseFilter	Boolean
TargetSensorTemperature	Floating Point
TemperatureControl	Enumerated
TemperatureStatus	Enumerated
TimestampClock	Integer
TimestampClockFrequency	Integer
TimestampClockReset	Command
TriggerMode	Enumerated
VerticallyCentreAOI	Boolean

APPENDIX B Apogee Feature Quick Reference

Feature	Type	Available Options
AcquiredCount	Integer	Na
AOIHBin	Integer	Na
AOIHeight	Integer	Na
AOILayout	Enumerated	Image, Kinetics, TDI
AOILeft	Integer	Na
AOITop	Integer	Na
AOIVBin	Integer	Na
AOIWidth	Integer	Na
BackoffTemperatureOffset	Floating Point	Na
BitDepth	Enumerated	12 Bit(Not AltaF/Aspen/Ascent), 16 Bit For AltaU/E use PixelReadoutRate “Normal” - > “16 bit”, “Fast” -> “12-bit”
CameraName	String	Na
CameraFamily	String	Na
CameraMemory	Integer	Na
ColourFilter	Enumerated	None, Blue, TrueSense
CoolerPower	Double	Na
DDR2Type	String	Na
DisableShutter	Boolean	Na
DriverVersion	String	Na
ExternalIOReadout	Boolean	Na
FanSpeed	Enumerated	Off, Low, Medium, High
FirmwareVersion	String	Na
ForceShutterOpen	String	Na
FrameCount	Integer	Na
FrameRate	Floating Point	Na
FrameInterval	Boolean	Na
FrameIntervalTiming	Boolean	Na
HeatSinkTemperature	Floating Point	Na
InputVoltage	Floating Point	Na
InterfaceType	String	Na
IOControl	Enumerated	Default, User
IODirection	Boolean/Enumerated	Input/0, Output/1
IOState	Boolean	Na
IRPreFlashEnable	Boolean	Na
KeepCleanEnable	Boolean	Na
KeepCleanPostExposureEnable	Boolean	Na
MicrocodeVersion	String	Na
Overlap	Boolean	Na
PixelHeight	Floating Point	Na
PixelReadoutRate	Enumerated	Normal, Fast (Not Available AltaE)
PortSelector	Integer	Na
PreAmpGainValue	Integer	Na
PreAmpOffsetValue	Integer	Na
SensorCooling	Boolean	Na
SensorHeight	Integer	Na
SensorModel	String	Na
SensorTemperature	Floating Point	Na
SensorType	Enumerated	CCD, CMOS
SensorWidth	Integer	Na
ShutterMode	Enumerated	Open, Closed, Auto
ShutterStrobePeriod	Double	Na
ShutterStrobePosition	Double	Na
ShutterAmpControl	Boolean	Na
ShutterState	Boolean	Na
TemperatureStatus	Enumerated	Backoff
TransmitFrames	Boolean	Na
TriggerMode	Enumerated	Internal, External, External Start, External Exposure
UsbProductId	Integer	Na

Feature	Type	Available Options
UsbDeviceId	Integer	Na

APPENDIX C Function Quick Reference

```
int AT_InitialiseLibrary();
int AT_FinaliseLibrary();

int AT_Open(int DeviceIndex, AT_H* Handle);
int AT_OpenDevice(AT_WC* Device, AT_H* Handle);
int AT_Close(AT_H Hndl);

typedef int (*FeatureCallback)(AT_H Hndl, AT_WC* Feature, void* Context);
int AT_RegisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback,
void* Context);
int AT_UnregisterFeatureCallback(AT_H Hndl, AT_WC* Feature, FeatureCallback EvCallback,
void* Context);

int AT_IsImplemented(AT_H Hndl, AT_WC* Feature, AT_BOOL* Implemented);
int AT_IsReadOnly(AT_H Hndl, AT_WC* Feature, AT_BOOL* ReadOnly);
int AT_IsReadable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Readable);
int AT_IsWritable(AT_H Hndl, AT_WC* Feature, AT_BOOL* Writable);

int AT_SetInt(AT_H Hndl, AT_WC* Feature, AT_64 Value);
int AT_GetInt(AT_H Hndl, AT_WC* Feature, AT_64 * Value);
int AT_GetIntMax(AT_H Hndl, AT_WC* Feature, AT_64 * MaxValue);
int AT_GetIntMin(AT_H Hndl, AT_WC* Feature, AT_64 * MinValue);

int AT_SetFloat(AT_H Hndl, AT_WC* Feature, double Value);
int AT_GetFloat(AT_H Hndl, AT_WC* Feature, double * Value);
int AT_GetFloatMax(AT_H Hndl, AT_WC* Feature, double * MaxValue);
int AT_GetFloatMin(AT_H Hndl, AT_WC* Feature, double * MinValue);

int AT_SetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL Value);
int AT_GetBool(AT_H Hndl, AT_WC* Feature, AT_BOOL* Value);

int AT_SetEnumIndex(AT_H Hndl, AT_WC* Feature, int Value);
int AT_SetEnumString(AT_H Hndl, AT_WC* Feature, AT_WC* String);
int AT_GetEnumIndex(AT_H Hndl, AT_WC* Feature, int* Value);
int AT_GetEnumCount(AT_H Hndl, AT_WC* Feature, int* Count);
int AT_IsEnumIndexAvailable(AT_H Hndl, AT_WC* Feature, int Index, AT_BOOL* Available);
int AT_IsEnumIndexImplemented(AT_H Hndl, AT_WC* Feature, int Index, AT_BOOL*
Implemented);
int AT_GetEnumStringByIndex(AT_H Hndl, AT_WC* Feature, int Index, AT_WC* String, int
StringLength);

int AT_Command(AT_H Hndl, AT_WC* Feature);

int AT_SetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value);
int AT_GetString(AT_H Hndl, AT_WC* Feature, AT_WC* Value, int StringLength);
int AT_GetStringMaxLength(AT_H Hndl, AT_WC* Feature, int* MaxStringLength);

int AT_QueueBuffer(AT_H Hndl, AT_U8 * Ptr, int PtrSize);
int AT_WaitBuffer(AT_H Hndl, AT_U8 ** Ptr, int* PtrSize, unsigned int Timeout);
int AT_Flush(AT_H Hndl);
```

APPENDIX D Code Listing for Tutorial

```
#include "atcore.h"

int main(int argc, char* argv[])
{
  int i_returnCode;
  AT_H Hndl;
  int i_cameraIndex = 2;
  i_returnCode = AT_InitialiseLibrary( );

  if (i_returnCode == AT_SUCCESS) {
    i_returnCode = AT_Open ( i_cameraIndex, &Hndl );
    AT_WC ExpFeatureName[] = L"Exposure Time";
    double d_newExposure = 0.02;
    i_returnCode = AT_SetFloat ( Hndl,  ExpFeatureName, d_newExposure);
    if (i_returnCode == AT_SUCCESS) {
      //it has been set
      double d_actualExposure;
      i_returnCode = AT_GetFloat ( Hndl,  ExpFeatureName, &d_actualExposure);
      if (i_returnCode == AT_SUCCESS) {
        //the actual exposure being used is d_actualExposure

        AT_64 ImageSizeBytes;
        AT_GetInt( Hndl, L"Image Size Bytes", &ImageSizeBytes);
        //cast to prevent warnings
        int i_imageSize = static_cast<int>(ImageSizeBytes);

        unsigned char* gblp_Buffer = new unsigned char[i_imageSize+8];

        unsigned char* pucAlignedBuffer = reinterpret_cast<unsigned char*>(
(reinterpret_cast<unsigned long>( gblp_Buffer ) + 7 ) & ~0x7);

        i_returnCode = AT_QueueBuffer(Hndl,  pucAlignedBuffer, i_imageSize);
        if (i_returnCode == AT_SUCCESS) {
          i_returnCode = AT_Command(Hndl, L"Acquisition Start");
          if (i_returnCode == AT_SUCCESS) {
            unsigned char* pBuf;
            int BufSize;
            i_returnCode = AT_WaitBuffer(Hndl, &pBuf, &BufSize, 10000);
            if ( i_returnCode  == AT_SUCCESS) {
              //successfully got image
              if (pBuf == pucAlignedBuffer) {
                //check pixel encoding to confirm format of data stream and process
              }
              else {
                //Error buffer pointer incorrect from AT_WaitBuffer
              }

            }
            else {
              //error with AT_WaitBuffer, analyse i_returnCode
            }
          }

          AT_Command(Hndl, L"Acquisition Stop");
          AT_Flush(Hndl );
        }
      }
      i_returnCode = AT_Close ( Hndl );
      if (i_returnCode != AT_SUCCESS) {
        // error closing handle
      }
    }
  }

  i_returnCode = AT_FinaliseLibrary( );
  if (i_returnCode != AT_SUCCESS) {
    //Error FinaliseLibrary
  }
  return 0;
}
```

APPENDIX E Conversion between char* and AT_WC

The following code shows an example of how to convert a char* null terminated string to the equivalent wide character string.

```
#include “stdlib.h”
char szStr[512];
AT_WC wcszStr[512];
mbstowcs(wcszStr, szStr, 512);
```

and from wide character string to char*

```
#include “stdlib.h”
char szStr[512];
AT_WC wcszStr[512];
wcstombs(szStr, wcszStr, 512);
```