USER'S GUIDE TO:

www.andor.com

Software Version 2.90

PAGE

SECTION 1 - INTRODUCTION

The Andor **S**oftware **D**evelopment **K**it (**SDK**) gives the programmer access to the Andor range of CCD and Intensified CCD cameras. The key part of the SDK is the Dynamic Link Library (DLL) which can be used with a wide variety of programming environments, including, C, C++, C#, Visual Basic and LabVIEW. The library is compatible with Windows 2000, XP, Vista and Windows 7. A Linux version of the SDK is also available. Currently, Andor provides both 32-bit and 64-bit versions of the SDK, for Windows and Linux.

The SDK provides a suite of functions that allow you to configure the data acquisition process in a number of different ways. There are also functions to control the CCD temperature and shutter operations. The driver will automatically handle its own internal memory requirements.

To use the SDK effectively, the user must develop a software package to configure the acquisition, provide memory management, process the data captured, and create the user interface.

The manual is broken into several sections, and it is recommended that the user read **Sections 1 - 10** before starting to use the SDK. These sections describe the installation process, camera initialization/configuration and data capture.

Section 11 is a complete function reference detailing the function syntax, parameters passed and error codes returned.

To further aid the user there is a comprehensive list of examples included with the SDK. The examples illustrate the use of C, Visual Basic and LabVIEW.

TECHNICAL SUPPORT

Contact details for your nearest representative can be found on our website.

SOFTWARE IMPROVEMENTS AND ADDITIONAL FEATURES

Version 2.90.30004.0

New features:

- USB iStar now supported
- Added function GetNumberPhotonCountingDivisions
- Added function GetPreAmpGainText
- Added 64-bit C# wrapper
- Added Shamrock C# wrapper
- Added 64-bit VB.NET header
- Added 64-bit LabVIEW support
- Added support for 50kHz and 1MHz on iKonM-PV inspector system

Bug fixes:

- GetKeepCleanTime not implemented for DV885
- The maximum binning should be limited by the size of the AD pipeline
- Recursive filter was not being reset between acquisitions.
- Frame Averaging filter was not working in frame transfer mode.
- Fixed crash on shudown with iKon-L
- Fixed crash if GetAcquisitionTimings is called for random tracks before tracks are set up.
- Removed some memory leaks
- Incorrect timings from GetAcquisitionTimings on Clara.
- Fixed saving random tracks to Fits.
- Luca S did not support temperature control.
- Minimum image length for a DU860 increased to 6 to avoid problems with isolated crop mode.
- Fixed SetPreAmpGain and IsPreAmpGainAvailable functions to check that the preamp gain index parameter is within range.
- Fixed data glitch on DV885 in frame transfer, external exposure mode (requires firmware upgrade)

Version 2.88.30002.0

New features:

- Added SDK function IsCountConvertModeAvailable to limit acquisition settings available for count convert.
- Added support for new iKon-L systems.
- Added support for new iKon-M systems.
- Added OptAcquire support for DV885 systems.

Bug fixes:

- Fixed race condition in WaitForAcquisitionTimeout.
- Image in crop mode on DU860 was shifting by 4 pixels for heights of less than 4.
- Fixed SR303 hardware issue where the step position of the wavelength drive will move when powered on.
- SetPCIMode should return DRV_NOT_SUPPORTED when not using the CCI-23/CCI-24 card.
- All Shamrock LabVIEW function names prepended with shamrock_ to avoid conflicts.

Version 2.88.30000.0

New features:

- Added OptAcquire feature to simplify configuration of iXon systems
- Added Count Convert feature to return data as photons or electrons
- Added Data Averaging feature for real time and post processing
- Added Spurious noise Filters for both real time and post processing
- Added Photon Counting post processing option
- Andor LabVIEW library updated to use version 8.0
- Added Dual Exposure Mode for iKon-L
- Updated SIFIO to enable the retrieval of calibration data
- Updated Shamrock SDK to include a calibration for Zolix spectrographs
- Added SDK function and capability for GetBaselineClamp
- Changed keep clean in FVB mode for iXon to prevent temperature drift

Bug fixes:

- Updated capability options for C#
- Updated Andor LabVIEW library
- Shutter open/close times fixed for Auto mode
- Fixed EM gain control when using multiple systems from the same executable
- Fixed isolated crop mode when data is being accumulated
- Fixed issues with control of multiple systems with multiple threads
- Fixed exposure time in software trigger mode when using large cycle time
- Fixed memory leak in GetAvailableCameras function
- Fixed random tracks stopping in video mode

Version 2.87.30000.0

New features:

- Clara E now supported
- Newton DU970/71P cameras now supported
- Cycle time reduced for imaging on Newton and iVac systems
- Number of accumulations can now be set in a kinetic series in overlap mode
- FVB cycle time reduced in crop mode provided only the height of the sensor has been cropped

Bug fixes:

- Clara near infra red mode not operating correctly when using FVB read mode
- Minimum exposure time increased to 1 millisecond for Clara near infra red mode
- Change to remove odd/even pixel noise after a number of accumulations in iDus
- Change to resolve image wrap around on Newton sensors
- Image was being shifted between frames when photon counting was being used on a Clara
- Fast kinetics now working in FVB mode
- First pulse missed in ring of exposures on Clara
- Updated bitmap header data to allow avi's to play in Windows 7
- Multiple systems was not supported for 64-bit Windows
- TimeStamp from Clara meta data was incorrect for a kinetic series of accumulations
- Video mode was eventually freezing in iCam PCI systems
- Fix for Spooling to fits issue in Windows 7
- Fix for image shift seen in DU940P newton cameras

Version 2.86.30000.0

New features:

- Clara meta data now stored in sif file format
- Vertical and horizontal flip tags added to the FITS header
- Newton now supports multiple images per USB interrupt to reduce CPU load
- Support added for new revision of Newton DU920P
- Control of gate mode added to iStar floating toolbar

Bug fixes:

- Fixed bug where SetPhotonCountingThreshold was always returning DRV_NOT_SUPPORTED
- Fixed reported acquisition timings for external trigger non frame transfer mode
- Fixed the SDK flipper mirror issue (problem with the port numbers being used) and updated shipped examples
- GetFIFOUsage is now thread safe
- USB driver for SR500 and SR750 updated to avoid conflicts with servo controllers
- Fixed External trigger, frame transfer, video mode operation

Version 2.85.30000.0

New features:

- Andor Clara image quality improved
- Option to run external exposure in a kinetic series for all cameras which support iCam
- Photon Counting check added to GetCapabilities
- Added kinetic cycle time tag to spooled tiff files
- PrepareAcquisition now returns an error if insufficient memory available

Bug fixes:

- SetSpool now returns DRV_NOT_AVAILABLE under Linux when trying to spool to FITS
- Fixed crash on initialize when no Andor cameras were connected
- Fixed problem with reinitializing Shamrock models SR500 and SR750
- Fixed problem where calling IsCoolerOn during an acquisition could stop the acquisition
- Fixed issue where events from a previous acquisition were not getting cleared
- Additional pixel shift removed from overlap mode on Clara

Version 2.84.30000.0

New features:

- Andor Clara now supported
  - SetDACOutput
  - SetDACOutputScale
  - GetNumberIO
  - SetIODirection
  - SetIOLevel
  - GetIOLevel
  - GetIODirection
  - SetTriggerInvert
  - IsAmplifierAvailable
  - SetOverlapMode

SetMetaData
GetMetaDataInfo

Bug fixes:

- Spooled files beyond 4GB could not be opened
- Data was being lost when spooled files of small images went beyond 4GB
- Spooled FITS file had cycle time saved as 0
- IsPreAmpAvailable should use channel passed rather than current one
- Random tracks data corrupted for consecutive tracks for cameras other than iXon+
- GetImages16 LabVIEW wrapper was calling wrong SDK function

Version 2.83.30001.0

New features:

- Added SetImageFlip and SetImageRotate functions to LabView wrapper

Bug fixes:

- Added ShamrockGetCalibration function to the Shamrock SDK help

Version 2.83.30000.0

New features:

- iVac systems now fully supported
- Shamrock spectrographs SR500 and SR750 now fully supported
- Fast kinetics now available for Luca-R
- Added High Capacity Mode support for DW936 cameras

Bug fixes:

- Fixes to Delphi header
- Fixed discrepancies between cycle times for multi-track and random track
- Fixed problem in fast kinetics when there was an odd number of super pixels
- Removed corrupted fire pulse in fast kinetics, external trigger
- Fix to resolve oscillations in data for certain Newton systems
- Fix for potential fail of auto cooling on Luca systems
- Fixed maximum number in series in fast kinetics for frame transfer systems
- Fixed exposure time reported in fast kinetics

Version 2.82.30000.0

New features:

- Added option for horizontal binning in random track mode
- Added capabilities for Horizontal Binning, MultiTrackHRange, and No Gaps in Random Tracks
- New capability added to test for overlapped external exposure mode

- Deprecated SetGain for SetMCPGain which is a more accurate naming convention
- Added Dud column support to SDK – allows SDK to be configured to interpolate bad columns

Bug fixes:

- Fixed minimum exposure for Luca-R
- Updated documentation – error code correction for get data functions.
- Fixed missing cases of GetTemperature in LabVIEW wrapper.
- Updated documentation – Corrected contact information.
- StartAcquisition now returns an error if horizontal binning does not divide evenly into range for multi-tracks
- Fixed crash when StartAcquisition is called in random track mode before random tracks are setup
- Fixed default EM gain – Set to off when system initialized
- SetRandomTracks no longer returns an error if not in random track mode
- Image mode Linux example will now work with an InGaAs
- SetRandomTracks was not returning an error for certain incorrect track combinations
- Fixed SetBaselineClamp and SetBaselineOffset – The test for availability was not complete
- Fixed GetRingExposureRange - Now uses same limit as SetRingExposureTimes
- Fixed SetRandomTracks - Was failing for some valid tracks
- Fixed SetGain error code - Now returns DRV_NOT_SUPPORTED if not an ICCD
- Fixed bug in SetRandomTracks to prevent negative numbers for number of tracks with correct return code
- GetAmpMaxSpeed now tests for NULL array parameter
- SetCustomTrackHBin returns DRV_NOT_SUPPORTED if not available for a system
- Fixed GetAmpDesc – Tests negative value for 3rd parameter – could cause crash
- Fixed GetAmpDesc – could return unterminated string
- Luca R cooler control was never supported but SDK returned DRV_SUCCESS - SDK functions now return proper error codes
- Fixed bug in Initialisation/Shutdown cycling – could cause crash
- Extra fire pulse when using kinetic series external exposure on DU885
- Fixed incorrect data when using kinetic series external exposure on Luca-R
- Fixed external exposure trigger mode for Luca-S

Version 2.81.30004.0

New features:

- Improved noise performance on DZ936 cameras at 3 and 5MHz horizontal readout speeds

Bug fixes:

- None

Version 2.81.30003.1

New features:

- None

Bug fixes:

- Fixed some documentation errors in LabVIEW context help
- Fixed Shamrock close and re-initialisation in C interface of Shamrock SDK
- Fixed Shamrock close operation in LabVIEW

Version 2.81.30002.0

New features:

- None

Bug fixes:

- Fixed cooling issue on Fibre Optic systems
- Fixed hot column issue on iXon DU888 cameras.
- Fixed crash in External Trigger on Newton
- Fixed DLL error on Windows Install program.

Version 2.81

New features:

- Improved shutdown in Linux during abnormal termination (Ctrl+C etc.) – signal handlers added
- CCI-24 support added to Linux SDK

Bug fixes:

- Removed Linux Device Driver compilation warnings for Kernel 2.6.23 and above.
- Crash could occur if GetAcquiredData was called before PrepareAcquisition or StartAcquisition.
- SetDriverEvent causes crash when called when system not initialized.
- GetImagesPerDMA did not return correct value unless PrepareAcquisition has been called.
- Timings incorrect for Frame Transfer in iCam mode.
- GetMostRecentImage[16] now returns correct data when used in Accumulate acquisition mode
- Fixed crash that would occur if GetNumberAvailableImages called before acquisition started
- Acquisitions now complete correctly if camera is reinitialised after being previously shutdown
- 64-bit SDK will now initialise USB cameras without the necessity of having libusb0_x64.dll in same directory as executable.
- Calibration values returned from Shamrock SDK were offset by 2 pixels from correct locations
- Fixed memory leak in SaveAsSif
- Fixed Luca re-initialisation issue - temperature reporting incorrect
- Documentation updates and corrections

Version 2.80

New features:

- iKon-L support added

- Added SetAccumulationCycleTime to LabVIEW library
- Random and multi tracks now available in frame transfer mode for iXon+
- SetNumberPrescans function added
- New timing functions added
  - GetKeepCleanTime
  - GetReadOutTime

Bug fixes:

- SetEMAdvanced was not working on Luca-R
- Random tracks external start was broken on a DU888
- InGaAs was not working in last release
- Multiple USB cameras could not be controlled
- Fast Kinetics external trigger was not working on early DV885 cameras
- Kinetic cycle time calculated wrongly when accumulating

Version 2.79

New features:

- Memory allocation improved to allow larger kinetic series to be acquired without spooling
- Luca-R range supported
- iKon-L supported
- GetImageFlip and GetImageRotation added
- Multi tracks available in frame transfer mode for iXon+
- Random tracks available in frame transfer mode for iXon+
- Capability added to test for multi and random tracks in frame transfer
- SetMultiTrackHRange added
- Random tracks can now be configured with no gaps in between for iXon+

Bug fixes:

- Temperature drifting is now handled for all cameras
- GetTemperatureStatus did not return result for iXon
- Pixel values for last column on DU885 incorrect
- Fast kinetics, external trigger not operating correctly on a DU885
- Crash when initiailising multiple usb cameras
- Kinetic cycle time could not be set reliably
- Shutter timings not correct at 35MHz on a DU885
- SetShutter function not functioning correctly for iXon+
- Grams files created not compatible with certain software packages
- EM gain could not be turned off completely
- Glitches were found in fire pulse for FVB mode on iXon+
- Image flipping and rotation properties incorrect in sif file
- CCI-20 controller card not initialising (ERROR_ACK)

Version 2.78.5

New features:

- SetIsolatedCropMode added to LabVIEW library
- SaveAsTiffEx function added to provide choice of whether data is scaled

Bug fixes:

- SaveAsTiff function for a kinetic series saved the same image for every frame
- SaveAsTiff now checks for available memory to avoid crash
- GetHeadModel function was returning model in lowercase
- iXon FPGA version not being read properly in Initialize function
- SetIsolatedCropMode function repaired
- GetAvailableCameras did not update with USB devices plugged in & out
- Using GetCameraEventStatus on fast acquisitions caused acquisition to fail
- Long kinetic series of FITS was not working

Version 2.78

New features:

- Support for Luca 285 added.
- Data transfer from USB cameras improved.
- SetIsolatedCropMode function added to provide crop mode option (added for iXon+):
- Improved support for integrate on chip: Added
  GetDDGIOCFrequency
  GetDDGIOCNumber
  SetDDGIOCNumber
- Option to export to raw data:
  SaveAsRaw

Bug fixes:

- SaveAsSif in SDK not storing readout speed correctly.
- Shutter now works correctly for Classic cameras when the software is run for the first time after rebooting PC.
- Data was wrapping at 65K if taking a kinetic series of accumulations.

Version 2.77

New features:

- Supports 32 and 64-bit Windows XP and Vista
- Moved to new USB device driver libUSB
- iCam: New Run Till Abort functionality for latest iXon (with CCI-23 controller card), and Luca Cameras:
  SendSoftwareTrigger
  SetRingExposureTimes
  GetAdjustedRingExposureTimes
  GetNumberRingExposureTimes
  GetRingExposureRange
  IsTriggerModeAvailable
- New image manipulation functions:
  SetImageFlip
  SetImageRotate
- Save as GRAMS SPC file format – SaveAsSPC
- Calculate the red and blue relative to green factors to white balance a colour image - WhiteBalance

Version 2.76

New features:

- Additional capabilities added to GetCapabilities function
- GetAmpDesc function added
- Timeout added for WaitForAcquisition function

Bug fixes:

- Error returned if an invalid EM gain mode is selected
- Fixed issues with Fast Kinetics on an iXon
- Sometimes a camera was not ready to acquire when an acquisition event was sent
- Fixed initialization problem when a '.' was in the path send to Initialize() function

Version 2.75

New features:

- Spooling to FITS, SIF and TIFF now available.
- SetBaselineOffset function added
- SetShutterEx added to control both an internal and external shutter through a DV8285
- SaveAsSif now handles spooled files

Bug Fixes:

- GetNumberHSSpeeds now includes error checking for classics
- GetCapabilities returns correct bit depth for an iDus
- IsPreAmpGainAvailable now indicates yes for classic cameras as long as the gain index is zero and other parameters are valid
- EMGain Capability now returned correctly for iDus, Newton, USB iStar
- SaveAsBmp was not working in latest version

Version 2.74

New features:

- Support for new Luca range of Cameras
- Control of linear EM gain:
  GetEMCCDGain
  GetEMGainRange
  SetEMGainMode
- Option to save to FITS file format : SaveAsFITS
- Crop mode available with Newton: SetCropMode

Version 2.73

New features:

- Support for Newton and SurCam range of Cameras

Bug fixes:

- GetMostRecentImage does not now prevent access to images previous to the one obtained
- Controller type can be tested.

SECTION 2 - SOFTWARE INSTALLATIONS

PC requirements

Please consult the Specification Sheet for your camera for the minimum and the recommended PC requirements.

SDK WINDOWS INSTALLATION

The installation of the Andor SDK software is a straightforward process, the steps for which are outlined below. Before proceeding with the installation, it is recommended that you read the remainder of this section first.

1. Insert the CD supplied with the SDK, and execute the "**SETUP.EXE**" program. This will take you through the complete installation process. You will be prompted to select the type of camera you have purchased as the installation needs to configure, were required, the "**Detector.ini**" file appropriately. You will also be requested to select a destination directory; this should be a directory that all users planning to use the SDK have full read/write privileges to. The directory will be created if it does not already exist. It is recommended that if you are performing an upgrade or reinstall that you do it to a clean directory.

Example programs will be copied into sub-directories of the installation directory specified above.

2. If not already installed, proceed with installing camera hardware. Consult your User guide for details. You may have to restart the PC to complete the installation
3. Navigate to the directory `'<destination directory>\Examples\C'` directory. Go into any sub directory and run the `'.exe'` file that you see there. If this runs successfully then your installation has completed. If it does not run with a successful message please consult the troubleshooting guide later in this section.

The installation process will copy the following files into the specified base directory:

ATMCD32D.DLL (32-bit Dynamic Link Library)

ATMCD64D.DLL (64-bit Dynamic Link Library)

DETECTOR.INI (Classic CCD, ICCD and iStar cameras only)

ATMCD32D.H (C, C++ only)

ATMCD32D.LIB (Borland compatible library, C, C++ only)

ATMCD32M.LIB (Microsoft compatible library, C, C++ only)

ATMCD32D.BAS (Visual Basic only)

ATMCD32D.PAS (Pascal only)

ATMCD32CS.DLL (C# only)

ATMCD32D.VB (VB.net)

NOTE: The files are also copied into each example directory. This is to allow each example to be run as a stand-alone program.

A device driver required to support the camera will also be installed. The actual driver installed will depend on the camera type and operating system version, i.e.:

- For PCI systems the driver file is **atmcdwdm.sys** for 32-bit operating systems, or **atmcdwdm64.sys** for 64-bit operating systems.
- For USB cameras the driver file is **libusb0.sys** for 32-bit operating systems, or **libusb0_x64.sys** for 64-bit operating systems.

NOTE: Do not have more than one example or other SDK software (e.g. Andor Solis™, iQ™) running at the same time.

Windows Troubleshooting

Installing on **Windows 7**

- Some users have experienced difficulty installing the SDK on Windows 7, if so please see the Window 7 Driver Installation Guide.

If you are running a **PCI camera**

- Check that the Andor Technology PCI driver appears in the Ensure that an Andor section in exists in the Device Manager and that an Andor Technology PCI driver appears in it. To access the Device Manager, go to the Control Panel and click on the "System" control. From here, select the Hardware tab and then click on the Device Manager button.
- Shut down the PC and ensure that the PCI card is seated correctly
- For 32-bit OS, ensure that the file atmcdwdm.sys file appears in the `C:\WINDOWS\system32\drivers` directory. The latest version is 4.29.0.0
- For 64-bit OS, ensure that the file atmcdwdm64.sys file appears in the `C:\WINDOWS\system32\drivers` directory. The latest version is 4.29.0.0
- If the Windows NT driver atmcd.sys is in the "Drivers" directory delete it and restart the PC.

If you are experiencing communication problems with the Andor **USB cameras** carry out the following actions:

- Confirm that the PC being used is **USB 2.0 compatible** and that a USB 2.0 port is being used for the camera
- Check the power to the iDus camera.
- Check the USB cable from the PC to the iDus camera.
- Ensure that a **LibUSB-Win32 Devices** section exists in the Device Manager and tab and that your camera is listed. To access the Device Manager, go to the Control Panel and click on the "System" control. From here, select the Hardware tab and then click on the Device Manager button. If the entry does not exist or there is an exclamation mark beside it carry out the following actions

1. Power the camera off and on and after the new hardware is detected, follow the instructions to install a driver for the new device. When asked for a location, point to the directory where the software was installed**.**
2. If there is a **USB device** with an **exclamation mark** beside it and you cannot account for this device then it is probably the Andor camera and the driver is not installed. Install the driver as described previously or right click on the entry and update driver.
3. Close down any Andor software, remove the USB cable from either the camera or the PC and reconnect it again. Run the software to see if the camera is now detected.
4. If still not connected then , remove the USB cable from either the PC or the camera, power the camera off and on the camera and reconnect the USB cable again.
5. Run the software to see if the camera is now detected.

NOTE: If the camera is still not detected after step 6, please contact the appropriate technical support person

SDK LINUX INSTALLATION

The first step is to unpack the archive that you have received. With the following steps replace <version> with the version number of the archive you have. E.g. 2.15

1. Open a terminal
2. Change the directory to where the `andor-<version>.tar.gz` file is located
3. Type `'tar -zxvf andor-<version>.tar.gz'`

A new directory named 'andor' is created.

To install the SDK run the script ‘install_andor’ from the ‘andor’ directory. See the ‘INSTALL’ file located in the same directory, for further information.

LABVIEW INSTALLATION

When you install the SDK onto a machine with LabVIEW installed, the SDK DLL and LabVIEW files are automatically copied into the LabVIEW install directory.

All Andor SDK function wrappers are present in a LabVIEW library file, "atmcd32d.llb", installed in your “*user.lib*” directory in you LabVIEW install folder.

The library can be added to any of your palette views. Instructions for adding the SDK to your palette view are described below.

Note: Depending on the version of LabVIEW you are using, the menu structure may be different. Please consult your LabVIEW manual for general help on adding LLBs if you have any issues.

1) Select the menu item "**Tools -> Advanced -> Edit Palette Views...**"
2) Right Click on the Functions tool bar & select "**Insert -> Submenu...**"
3) In the dialog select "**Link to LLB library...**"
4) Navigate to the user.lib directory and select "*atmcd32d.llb*" - The submenu with all SDK functions has been added
5) Right click on the new palette view and select "**Rename Submenu...**"
6) Change the name to "Andor SDK"
7) Repeat steps 2-6 for the Controls tool bar.

Linux Troubleshooting

If you are having trouble running your camera under the Linux operating system please check the following before contacting Technical Support

For PCI,

- Check that the device driver is loaded. Type '`/sbin/lsmod`' – andordrvlx should be listed.

For USB,

- Check that libUSB is available, '`whereis libusb`'
- Check that the Andor device is listed in the `/proc/bus/usb/devices` file.
- Check that the relevant device under `/proc/bus/usb/00X/00Y` has write access for all users.

SECTION 3 - READOUT MODES

INTRODUCTION

Andor systems are based on a detector known as a **C**harged **C**oupled **D**evice (**CCD**). The detector is divided up as a 2-dimensional array of pixels, each capable of detecting light. For example, systems based on an EEV 30-11 CCD chip have 1024 X 256 pixels, where each pixel is 26μm^2 (all examples given in this manual assume an EEV 30-11 based system). This 2-dimensional nature allows the device to be operated using a number of different binning patterns. We refer to these binning patterns as **Readout Modes**.

Andor has several different readout modes as follows:

- **F**ull **V**ertical **B**inning (**FVB**)
- Single-Track
- Multi-Track
- Random-Track
- Image
- Cropped

Figure 1 shows the binning patterns :

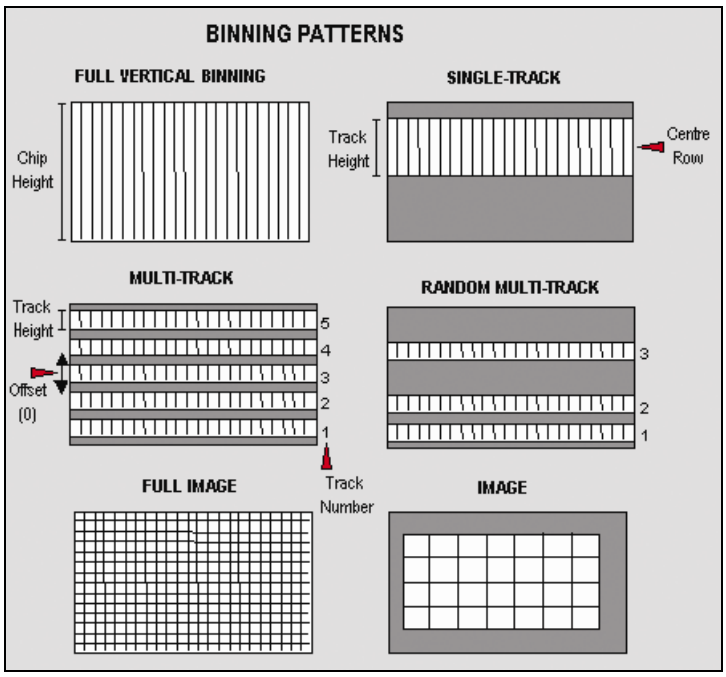


Figure 1: Binning patterns

We will now look at each of these modes in more detail.

NOTE: All of the patterns described can be simulated by the user in software but by carrying out the pattern in the camera greatly increases speed and improves Signal to Noise ratio.

Full Vertical Binning

Full **V**ertical **B**inning **(FVB)** is the simplest mode of operation. It allows you to use the CCD chip as a Linear Image Sensor (similar to a photo diode array). The charge from each column of pixels is vertically binned into the shift register. This results in a net single charge per column. Therefore, for a 30-11 CCD an acquisition using **FVB** will result in 1024 data points.

To set-up a Full Vertical Binning acquisition call:

SetReadMode**(0)**

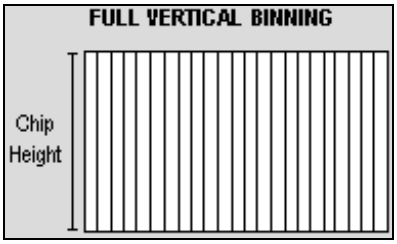


Figure 2: Full Vertical Binning

Single-Track

Single-Track mode is similar to the Full Vertical Binning mode discussed previously in that upon completion of an acquisition you will have a single spectrum. However, that is where the similarities end.

With Single-Track you can specify not only the height (in pixels) of the area to be acquired but also its vertical position on the CCD. To ensure the best possible Signal to Noise ratio all the rows within the specified area are binned together into the shift register of the CCD and then digitized.

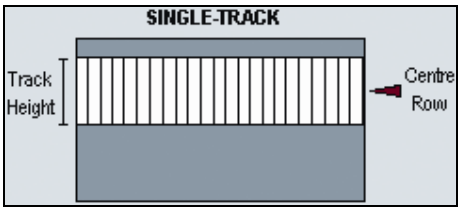


Figure 3: Single-track

Single-Track mode is useful because you are able to precisely define only the area of the CCD sensor that is illuminated by light. This is particularly important in low light level applications as it allows you to minimize the contribution of dark current in the measured signal. Also, if you are using an imaging spectrograph, such as the Shamrock, with a multiple core fiber, this mode allows you to select a single fiber for examination.

To set-up a Single-Track acquisition you need to call the following functions:

SetReadMode(3);

SetSingleTrack(128,20);

NOTE: If a non frame-transfer camera is used, a shutter may be required to prevent light (which would otherwise fall on the CCD-chip outside the specified track) from corrupting the data during binning. Please refer to SECTION 8 - SHUTTER CONTROL for further information.

Multi-Track

Multi-Track mode allows you to create one or more tracks (each of which behaves like the **Single-Track** above). With Multi-Track you specify the number of tracks and the track height. The driver internally sets the actual position of each track so that the tracks are evenly spaced across the CCD. The tracks can be vertically shifted, en masse, by specifying a positive or negative offset about a central position. For greater control over the positioning of the tracks use **Random-Track** mode.

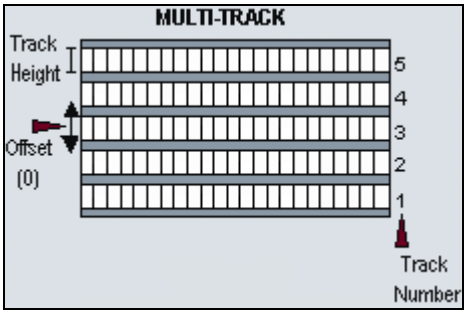


Figure 4: Multi-Track

Multi-Tracks will allow you to simultaneously acquire a number of spectra, delivered typically via a fiber bundle. If you are using a non-frame transfer camera and a continuous source, you will need to use a shutter to avoid streaking the spectra during the binning process. Please refer to SECTION 8 - SHUTTER CONTROL for further information.

To set-up a Multi-Track acquisition you need to call the following functions:

SetReadMode(1);
SetMultiTrack(5,20,0,bottom, gap);

The SetMultiTrack function also returns the position of the first pixel row of the first track "**bottom**", together with the gap between tracks, "**gap**". This allows the user to calculate the actual position of each track.

NOTE:

1. **Before using Multi-Track mode with fiber bundles it is often useful to acquire a full resolution image of the output. Having observed the vertical position and spacing of the individual spectra, you can vary track height and offset accordingly.**
2. **Imaging spectrographs vertically invert input light (i.e. light from the top fiber will fall on the bottom track on the CCD-chip.)**

Random-Track

In Random-Track mode the position and height of each track is specified by the user, unlike Multi-Track mode were the driver sets the position of each track automatically.

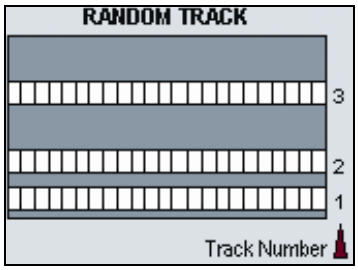


Figure 5: Random-Track

Random-Track will allow you to simultaneously acquire a number of spectra, delivered typically via a fiber bundle. Unless you are acquiring data from a pulsed source you will need to use a shutter to avoid streaking the spectra during the binning process. To set-up a Random-Track acquisition you need to call the following functions**:**

```
SetReadMode(2);
int position[6];
position[0] = 20;
position[1] = 30;  //end of track 1, 11 rows height
position[2] = 40;  //start of track 2
position[3] = 40;  //end of track 2, 1 row height
position[4] = 100; //start of track 3
position[5] = 150; //end of track 3, 51 rows height
SetRandomTracks(3,position);
```

The SetRandomTracks function validates all the entries and then makes a local copy of the tracks positions. For the array of tracks to be valid the track positions MUST be in ascending order.

NOTES:

1. **A track of 1 row in height will have the same start and end positions.**
2. **Before using Random-Track mode with fiber bundles it is often useful to acquire a Full Resolution Image of the output.**
3. **Having observed the vertical positions of the individual spectra set the Random-Track mode accordingly.**
4. **Imaging spectrographs vertically invert input light (i.e. light from the top fiber will fall on the bottom track on the CCD-chip.)**

Image

In **Image** mode the CCD is operated much like a camera. In this mode you get a measured value for each pixel on the CCD, in effect allowing you to 'take a picture' of the light pattern falling on the pixel matrix of the CCD. To prevent smearing the image, light must be prevented from falling onto the CCD during the readout process. Please refer to SECTION 8 - SHUTTER CONTROL for further information.

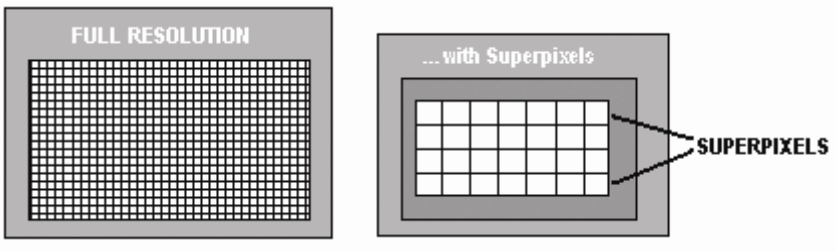


Figure 6: Image mode

To reduce the file size and increase the speed of readout it is possible to specify a sub-area of the CCD to be read out. It is also possible to bin pixels together horizontally and vertically to create super pixels.

To set up a "Full Resolution Image" acquisition you need to call the following functions:

SetReadMode(4);
SetImage(1,1,1,1024,1,256);

To acquire a sub-area with lower left co-ordinates of (19, 10), with binning of 4 in both the horizontal and vertical directions, and 100x16 pixels in the acquired image you would call the SetImage function with the following parameters:

SetImage(4,4,19,118,10,25);

By a process of binning charge vertically into the shift register from several rows at a time (e.g. 4) and then binning charge horizontally from several columns of the shift register at a time (e.g. 4) the ANDOR SDK system is effectively reading out charge from a matrix of super pixels which each measure 4 x 4 real pixels. The result is a more coarsely defined image, but faster processing speed, lower storage requirements, and a better signal to noise ratio (since for each element or super pixel in the resultant image, the combined charge from several pixels is being binned and read out, rather than the possibly weak charge from an individual pixel).

Cropped

In **Cropped** mode, we can "fool" the sensor into thinking it is smaller than it actually is, and readout continuously at a much faster frame rate. The spectral time resolution is dictated by the time taken to readout the smaller defined section of the sensor.

If your experiment dictates that you need fast time resolution but cannot be constrained by the storage size of the sensor, then it is possible to readout the EMCCD in a "cropped sensor" mode, as illustrated below.

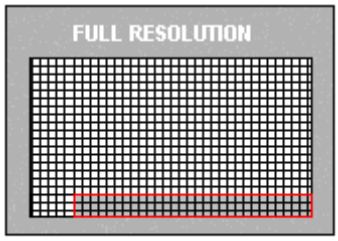


Figure 7: Cropped mode

To set up the CCD with a cropped image, as in figure 7, see SetIsolatedCropMode.

NOTE: It is important to ensure that no light falls on the excluded region otherwise the acquired data will be corrupted.

SECTION 4 - ACQUISITION MODES

ACQUISITION MODE TYPES

In the previous section the different ReadOut Modes (binning patterns) supported by the Andor SDK were discussed. In addition the Andor SDK allows you to control the number and the timing details of acquisitions made using the various binning patterns. To simplify the process of controlling these acquisitions the Andor SDK has divided the acquisition process into several different **Acquisition Modes:**

- **Single Scan**
- **Accumulate**
- **Kinetic Series**
- **Run Till Abort**
- **Fast Kinetics**

Single Scan is the simplest form of acquisition where a single scan is captured.

Accumulate mode takes a sequence of single scans and adds them together.

Kinetic Series mode captures a sequence of single scans, or possibly, depending on the camera, a sequence of accumulated scans.

Run Till Abort continually performs scans of the CCD until aborted.

If your system is a Frame Transfer CCD, the acquisition modes can be enhanced by setting the chip operational mode to Frame Transfer.

In the remainder of this section we will discuss in detail what each of these modes actually are and what needs to be specified to fully define an acquisition.

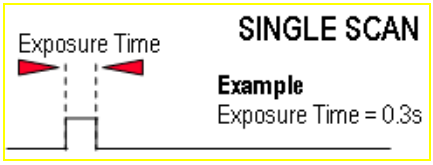
The table below summarizes the information that is needed for each acquisition mode:

MODE	EXPOSURE TIME	ACCUMULATE CYCLE TIME	NO. OF ACCUMULATIONS	KINETIC CYCLE TIME	NO. IN KINETIC SERIES
SINGLE SCAN	X				
ACCUMULATE	X	X	X		
KINETIC SERIES	X	X	X	X	X
RUN TILL ABORT	X			X	
FAST KINETICS	X	X			X

NOTE: For the purpose of this document an acquisition is taken to mean the complete data capture process. By contrast, a scan is a single readout of data from the CCD-Chip, i.e. a complete data acquisition comprises the capture of one or more scans.

Single Scan

Single Scan is the simplest acquisition mode available with the Andor system. In this mode Andor SDK performs one scan (or readout) of the CCD and stores the acquired data in the memory of the PC.



To set the acquisition mode to Single Scan call:

SetAcquisitionMode(1)

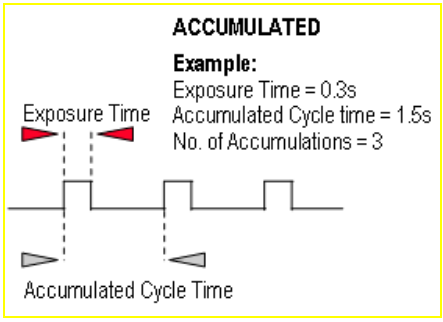
SetExposureTime(0.3)

Here the exposure time is the time during which the CCD sensor is sensitive to light. The exposure time is set via the SetExposureTime function.

NOTE: Due to the time needed to shift charge into the shift register, digitize it and operate shutters, where necessary, the exposure time cannot be set to just any value. For example, the minimum exposure time depends on many factors including the readout mode, trigger mode and the digitizing rate. To help the user determine what the actual exposure time will be the driver automatically calculates the nearest allowed value, not less than the user's choice. The actual calculated exposure time used by Andor SDK may be obtained via the GetAcquisitionTimings function (this function should be called after the acquisition details have been fully defined i.e. readout mode, trigger mode etc. have been set).

Accumulate

Accumulate mode adds together (in computer memory) the data from a number of scans to form a single 'accumulated scan'. This mode is equivalent to taking a series of Single Scans and "manually" adding them together. However, by using the built-in Accumulate mode you gain the ability to specify the time delay (or period) between two consecutive scans and also the total number of scans to be added.



To set the acquisition mode to Accumulate call:

SetAcquisitionMode (2)

To fully define an *Accumulate* acquisition you will need to supply the follow information:

Exposure Time. This is the time in seconds during which the CCD sensor collects light prior to readout. Set via the SetExposureTime function.

Number of Accumulations. This is the number of scans to be acquired and accumulated in the memory of the PC. Set via the SetNumberAccumulations function.

Accumulate Cycle Time. This is the period in seconds between the start of each scan.

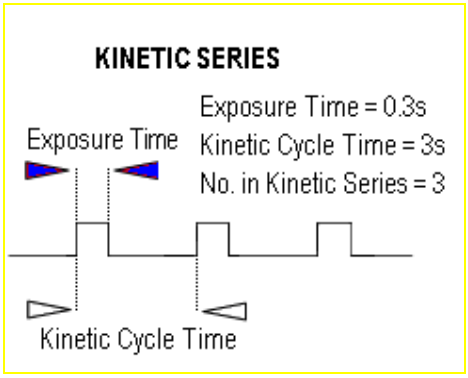
Set via the SetAccumulationCycleTime function. (This parameter is only applicable if you have selected **Internal** trigger – **Please refer to SECTION 6 – TRIGGERING for further information**.

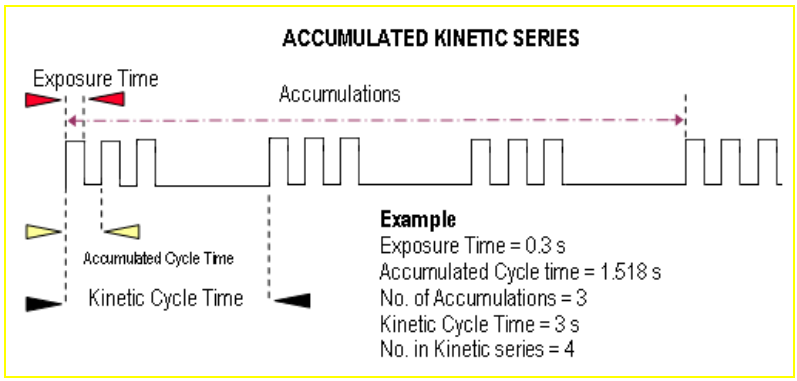
NOTES:

1. **If the exposure time or the cycle time are set too low or are not permissible values, the driver will automatically calculate the nearest appropriate value.**
2. **The actual values used can be obtained via the GetAcquisitionTimings function (this function should be called after the acquisition has been fully defined (i.e. readout mode, trigger mode etc. have been set).**
3. **In External Trigger mode the delay between each scan making up the acquisition is not under the control of the Andor system but is synchronized to an externally generated trigger pulse.**

Kinetic Series

Kinetic Series mode captures a sequence of single scans, or a sequence of accumulated scans, into memory. This mode is equivalent to manually taking a series of single scans (or accumulated scans). However, by using the built-in Kinetic Series mode you gain the ability to specify the time delay (or period) between two consecutive scans and also the total number of scans to be acquired.





NOTE: In External Trigger mode the delay between each scan making up the acquisition is not under the control of the Andor SDK, but is synchronized to an externally generated trigger pulse.

To set the acquisition mode to **Kinetic Series** call:

SetAcquisitionMode(3)

To fully define a Kinetic Series acquisition you will need to supply the following information:

Exposure Time. This is the time in seconds during which the CCD collects light prior to readout.

Set via the SetExposureTime function.

Number of Accumulations. This is the number of scans you want to add together to create each member of your kinetic series. The default value of **1** means that each member of the kinetic series will consist of a single scan.

Set via the SetNumberAccumulations function.

Accumulate Cycle Time. This is the period in seconds between the start of individual scans (see **Number of Accumulations** above) that are accumulated in computer memory to create each member of your kinetic series - each member of the series is an 'accumulated scan'.

Set via the SetAccumulationCycleTime function.

(This parameter is only applicable if you have selected the Internal trigger and the Number of Accumulations is greater than 1- **Please refer to SECTION 6 – TRIGGERING for further information**.)

Number in Kinetic Series. This is the number of scans (or 'accumulated scans') you specify to be in your series.

Set via the SetNumberKinetics function.

Kinetic Cycle Time. This is the period in seconds between the start of each scan (or set of accumulated scans, if you have set the **Number of Accumulations** to more than **1**) in the series.

Set via the SetKineticCycleTime function.

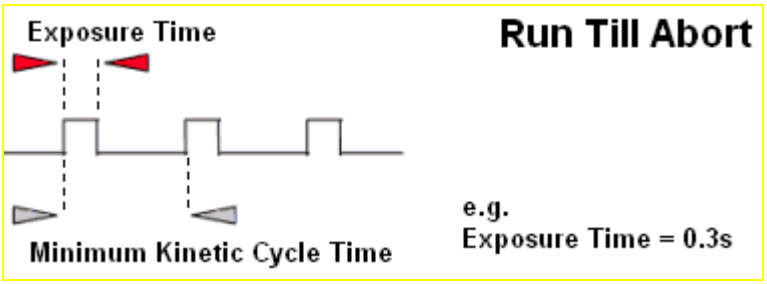
(This parameter is only applicable if you have selected the **Internal** trigger - see Trigger Modes.)

NOTE:

1. **If the exposure time or the cycle time are set too low or are not permissible values, the driver will automatically calculate the nearest appropriate value.**
2. **The actual values used can be obtained via the GetAcquisitionTimings function. This function should be called after the acquisition has been fully defined i.e. readout mode, trigger mode etc. have been set). If you are using a shutter, please refer to SECTION 8 – SHUTTER CONTROL for further information**

Run Till Abort

Run Till Abort mode continually performs scans of the CCD at the rate set by the user, until the acquisition is stopped by calling the AbortAcquisition function. The minimum possible delay between each scan will be the minimum Kinetic Cycle Time.



To set the acquisition mode to **Run Till Abort** call:

SetAcquistionMode(5)
SetExposureTime(0.3)
SetKineticCycleTime(0)

Here the exposure time is the time during which the CCD sensor is sensitive to light.

NOTES:

1. **The total number of images acquired during the acquisition can be obtained at any time by calling the GetTotalNumberImagesAcquired function. The data acquired during the acquisition will be stored in the circular buffer until it is overwritten by new scans. The capacity of the circular buffer can be obtained by calling the GetSizeOfCircularBuffer function. To retrieve all valid data from the circular buffer before it is overwritten by new data the GetNumberNewImages and GetImages functions should be used. Alternatively, to retrieve only the most recent image the GetMostRecentImage function can be used. Finally, to retrieve the oldest image the GetOldestImage function can be used.**
2. **Due to the time needed to shift charge into the shift register, digitize it and operate shutters, where necessary, the exposure time cannot be set to just any value. For example, the minimum exposure time depends on many factors including the readout mode, trigger mode and the digitizing rate. To help the user determine what the actual exposure time will be, the driver automatically calculates the nearest allowed value that is not less than the user's choice. Thus, the actual calculated exposure time used by Andor SDK may be obtained via GetAcquisitionTimings (this function should be called after the acquisition details have been fully defined i.e. readout mode, trigger mode etc. have been set).**

When in this mode of operation (Run Till Abort) some systems have an enhanced trigger mode and enhanced exposure time capability. To check if these enhanced features are available with your system, use the function GetCapabilities and check the ulTriggerModes variable for bit 3 (AC_TRIGGERMODE_CONTINUOUS) being set.

The enhanced features include:·

1. Ring of exposures
2. Software Trigger or External trigger
3. Ability to change exposure times during acquisition without aborting the run.
4. External Level Exposure (Bulb) Trigger

These enhanced features are particular useful in situations where you need to acquire data at a fast rate but not at some predefined rate or when you need to change the exposure time between successive scans. A good example would be calcium imaging where you need to take 2 images at different wavelengths with possibly different light levels. With this new mode of operation you would set the experiment up as follows:

1. Configure the camera to acquire an image

   SetReadMode, SetImage, SetFrameTransferMode
2. Select Run-till-abort mode SetAcquisitionMode
3. Select Software trigger SetTriggerMode(10)

   Confirm with IsTriggerModeAvailable(10)
4. Set exposure time. SetExposureTime or SetRingExposureTimes
5. Move filter to first position
6. Start acquisition. StartAcquisition
7. Send software Trigger. SendSoftwareTrigger
8. Wait for an acquisition event. See SetDriverEvent
9. Move Filter to next position.
10. Change exposure time. See SetExposureTime
11. Retrieve data see GetAcquiredData
12. Go to step 7

In the procedure outlined above we manually changed the exposure during the sequence. However, we could have used the new “Ring of exposures” feature to set up the two exposure times in advance and let the camera automatically switch between them as necessary. see SetRingExposureTimes

There is also the ability to detect the end of the exposure and start reconfiguring the experiment for the next

acquisition while the readout of the first scan is still in progress. See SetAcqStatusEvent.

NOTE: This will also work in External trigger mode SetTriggerMode, with an external trigger source determining the start of an exposure instead of the SendSoftwareTrigger command. In external trigger care must be taken to ensure that the external trigger occurs when the camera is ready for it i.e. the frequency of the external trigger source has to be within the capabilities of the camera with the current settings.

With External Exposure trigger mode the width of the trigger pulse source will determine the exposure time and the Ring of Exposures will not be applicable.

See also Acquisition Modes. GetAdjustedRingExposureTimes GetNumberRingExposureTimes GetRingExposureRange IsTriggerModeAvailable SendSoftwareTrigger SetRingExposureTimes SetTriggerMode

Fast Kinetics

Fast Kinetics is a special readout mode that uses the actual sensor as a temporary storage medium and allows an extremely fast sequence of images to be captured. The capture sequence is described with the following steps:

Step 1: both the Image and Storage areas of the sensor are fully cleaned out (the Keep Clean Cycle)

Step 2: the Keep Clean Cycle stops and the acquisition begins. The image builds up on the illuminated section of the sensor which is typically a *small* number of rows at the top of the sensor

Step 3: the sensor remains in this state until the exposure time has elapsed, at which point the complete sensor is clocked vertically by the number of rows specified by the user.

Steps 4 & 5: the process is continued until the number of images stored equals the series length set by the user.

Step 6: at this point the sequence moves into the readout phase by first vertically shifting the first image to the bottom row of the sensor. The sensor is then read out in the standard method.

Points to consider for Fast Kinetics Mode:

• Light MUST only be allowed to fall on the specified sub-area. Light falling anywhere else will contaminate the data.

• The maximum number of images in the sequence is set by the position of the sub-area, the height of the sub-area and the number of rows in the CCD (Image and Storage area)

• There are no Keep Clean cycles during the acquisition sequence.

• The industry fastest vertical shift speeds of the iXon^{EM}+ enables fastest time resolution with minimal vertical smearing.

• A range of internal trigger and external trigger options are available for Fast Kinetics Readout.

Frame Transfer

Frame transfer is a mode of operation of the chip that is only available if your system contains a Frame Transfer CCD (**FT CCD**). It can be switched on for any acquisition mode.

A FT CCD differs from a standard CCD in 2 ways:

- Firstly, a FT CCD contains 2 areas, of approximately equal size (see figure 7 below).
    1. The first area is the **Image area**, this area is at the top and farthest from the readout register. It is in this area that the CCD is sensitive to light.
    2. The second area is the **Storage area** and sits between the Image area and the readout register. This area is covered by an opaque mask, usually a metal film, and hence is not sensitive to light.
- The second way in which a FT CCD differs from a standard CCD is that the Image and the Storage areas can be shifted independently of each other.

These differences allow a FT CCD to be operated in a unique mode where one image can be read out while the next image is being acquired. It also allows a FT CCD to be used in imaging mode without a shutter.

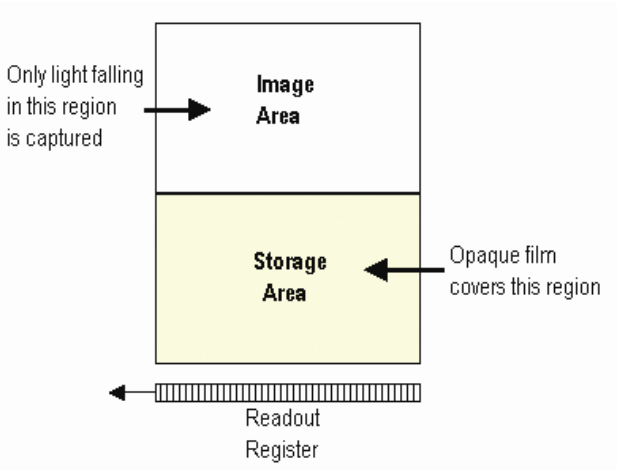


Figure 7: Frame Transfer CCD

Figure 8 takes you through the capture sequence for an FT CCD:

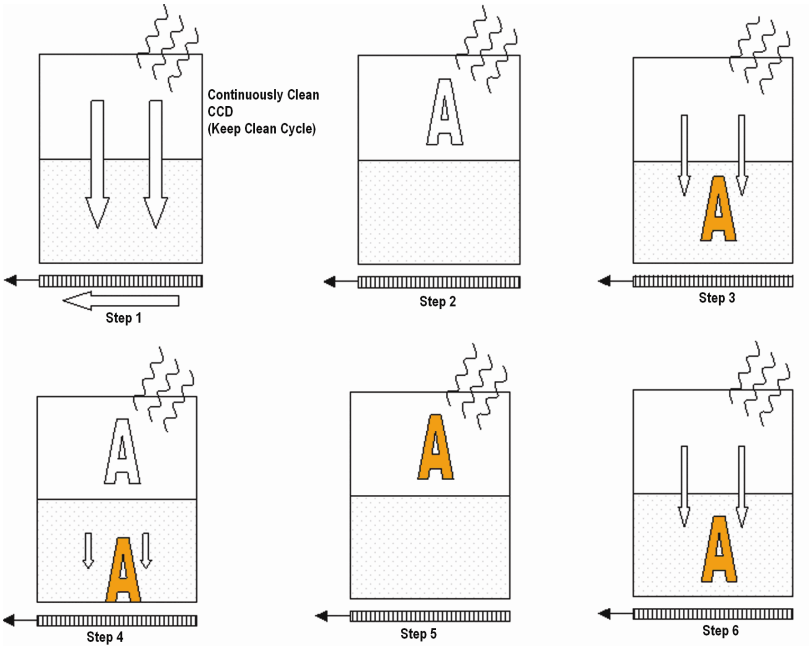


Figure 8: Capture sequence for a Frame Transfer CCD

Step 1: Both Image and Storage areas of the CCD are fully cleaned out. This is known as a **Keep Clean Cycle**. Keep Clean Cycles occur continuously to ensure that the camera is always ready to start an acquisition when required.

Step 2: On receipt of a start acquisition command the CCD stops the Keep Clean Cycle. This allows the image, photoelectric charge, to build up in the Image area of the CCD. The CCD remains in this state until the exposure time has elapsed, at which point the readout process starts.

Step 3: In this step the charge, built up in the Image *area,* is quickly shifted into the Storage area. The time required to move the charge into the storage area is calculated as follows:

No. of Rows in the Image Area x Vertical Shift Rate.

Once the Image area has been shifted into the storage area the Image area stops vertically shifting and begins to accumulate charge again, i.e. the next exposure starts.

Step 4: While the Image area is accumulating charge the Storage area is being read out. This readout phase can take tens of milliseconds to seconds depending on the image size, readout pattern and readout speed.

Step 5 & 6: On completion of the readout, the system will wait until the exposure time has elapsed before starting the next readout (**Step 6)**.

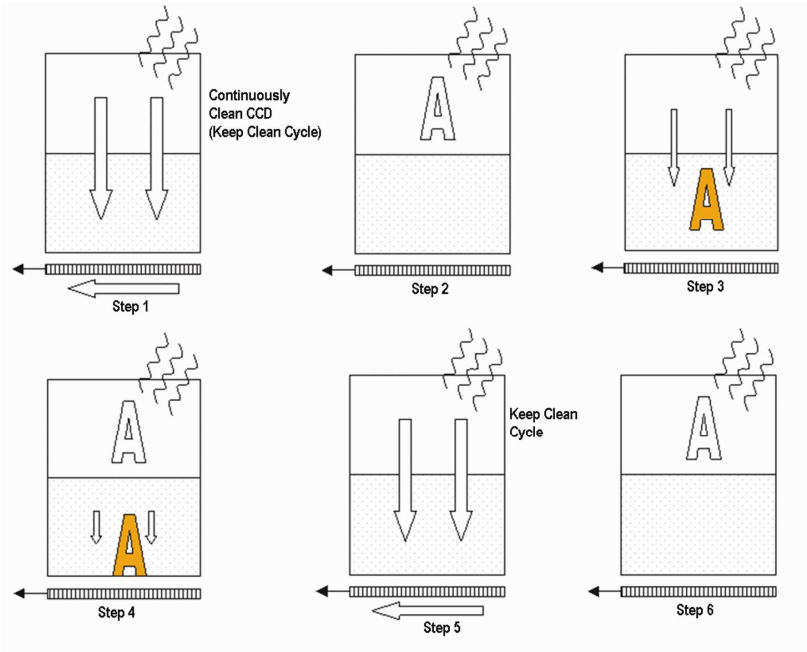
As the captured image is quickly shifted into the Storage area, a Frame Transfer CCD system can be used **without** a mechanical shutter.

NOTES:

- **When using Frame Transfer mode, the minimum exposure time for a FT CCD operated in frame transfer mode is the time taken to readout the image from the storage area.**

- **The Accumulation Cycle Time and the Kinetic Cycle Time are fully dependent on the exposure time and hence cannot be set via the software.**
- **For our Classic CCD range of cameras with frame transfer type sensors the camera can be operated in External Trigger mode. In this mode there are no keep cleans and the external trigger starts the "Readout" phase. The exposure time is the time between external triggers and hence the user cannot set the exposure or cycle times.**
- **For our iXon range of cameras the external trigger mode is more flexible. With these cameras the user can define the amount of time between the external trigger event occurring and the readout starting. This can be useful in those situations where the TTL trigger occurs before the light event you are trying to capture. As in the Classic Camera case, no keep cleans are running and the true exposure time is the time between triggers. However, the exposure window has moved in time by the exposure time.**
- **There is no need for a mechanical shutter. As the exposure time is long compared to the time required to shift the image into the storage area and therefore, image streaking will be insignificant.**

It is also possible to operate a FT CCD in a non-frame transfer mode. In this standard mode of operation, an FT CCD acts much like a standard CCD. The capture sequence for this standard mode is illustrated here:



- **Step 1:** Both Image and Storage areas of the CCD are fully cleared out (the Keep Clean Cycle).
- **Step 2:** When an acquisition begins, the CCD stops the Keep Clean Cycle. The image builds up in the Image area of the CCD. The CCD remains in this state until the exposure time has elapsed, at which point the readout process starts.
- **Step 3:** The charge built up in the Image area is quickly shifted, into the Storage area. The time required to move the charge into the Storage area is the same as in the Frame Transfer mode.
- **Step 4:** With the image now in the Storage area the captured image is read out. The time taken to read out the image is again the same as in the Frame Transfer mode.
- **Step 5:** On completion of the readout, the CCD is again completely cleared, ready to acquire the next image. The CCD remains in the Keep Clean Cycle until the end of the accumulation or kinetic cycle time, depending on the acquisition mode, i.e. back to **Step 1**. As at least one Keep Clean Cycle is performed between each exposure, the minimum exposure time is no longer set by the time to read out the image.

As the captured image is quickly shifted into the Storage area, even in **non-Frame Transfer mode**, the system may still be used without a mechanical shutter.

NOTES:

- **When using an FT CCD as a standard CCD, the Exposure Time, Accumulation Cycle Time and Kinetic Cycle Time can be set independently.**
- **The minimum exposure time is not related to the time taken to read out the image.**
- **External trigger operates as if the CCD was a Non-FT CCD.**
- **As the captured image is quickly shifted into the storage area, even in non-frame transfer mode, the system may still be used without a mechanical shutter.**
- **For short exposure times the image may appear streaked as the time taken to shift the image area into the storage area may be of similar magnitude.**
- **Light falling on the Image area while the Storage area is being read out may contaminate the image in the Storage area due to charge spilling vertically along a column from the Image area. The slower the readout rate or the shorter the exposure time the greater the possibility of corruption. To see why this is the case, consider the following situation:**

*"During a 100us exposure enough light has fallen on a pixel to register 10000 counts, or 100,000 electrons assuming 10e/count. The image is then shifted into the Storage area. To read out the image, assuming 1000x1000 pixels, it would take approximately 100ms at 10MHx readout rate. This means that during the reading out of the image 10 million counts (10000 * 1000) will have been acquired into the pixel described above. As a pixel saturates at approximately 160,000 electrons this means that the pixel will over saturated by 60 times. All the excess charge has to go somewhere, and spreads vertically along the CCD column. As the clocks in the Image area are not actively shifting the charge, the mobility of the charge will be low and you may not see any effect. However, when you consider that more than one pixel in any given column could be exposed to 10000 counts per 100us, the chance of corrupting data is correspondingly increased. Changing the readout rate to 1 microsecond per pixel will greatly decrease the possibility of data corruption due to the reduced time to read out the image. Reducing the amount of light falling on the CCD and increasing the exposure time accordingly will also reduce the possibility of data corruption."*

By default the system is set to non-Frame Transfer mode. To set the chip operation mode to Frame Transfer call:

SetFrameTransferMode**(1)**

To switch back to non-frame transfer mode call

SetFrameTransferMode**(0)**

To fully define a Frame Transfer acquisition you will need to supply the following information:

- **Exposure Time:** Time in seconds during which the CCD collects light prior to readout. Set via the SetExposureTime function.
- **Number of Accumulations:** Number of scans you want to add together to create each member of your kinetic series. The default value of **1** means that each member of the kinetic series will consist of a single scan. Set via the SetNumberAccumulations function.
- **Number in Kinetic Series:** Number of scans (or **accumulated scans**) you specify to be in your series. Set via the SetNumberKinetics function.

SECTION 5 - TRIGGERING

TRIGGER MODES

To assist the user in synchronizing data capture with external events the Andor system supports several modes of triggering, including

Internal

External

External Start

External Exposure (Bulb)

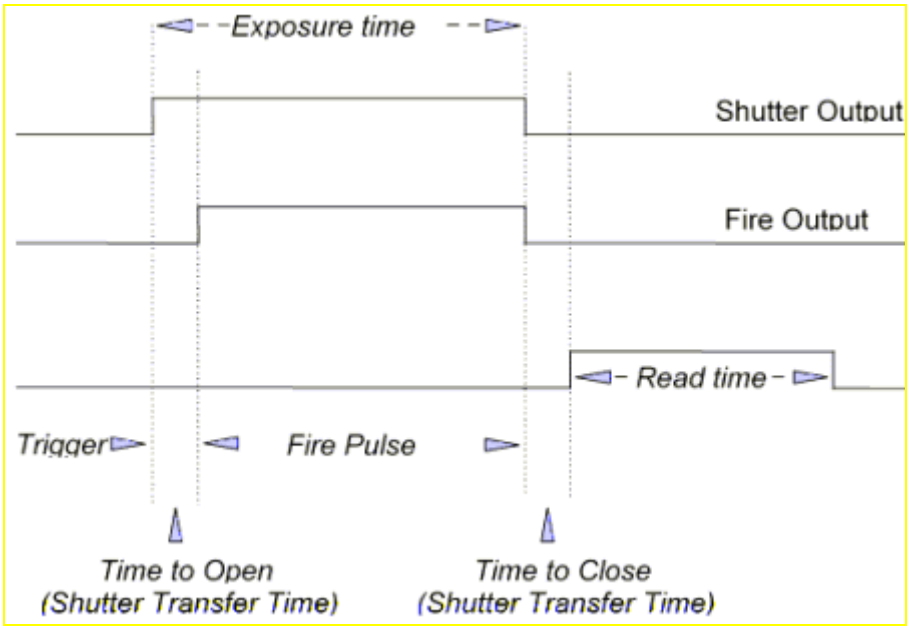
External FVB EM (only valid for EM Newton models in FVB mode) (needs added)

Software

The trigger mode is set via the SetTriggerMode function. In the remainder of this section we will examine the modes in detail and give some indication on the appropriate application of each trigger mode.

Internal

In Internal Trigger Mode once an acquisition has been started via the StartAcquisition function the Andor system determines when data is actually acquired. Before the camera starts the data capture process it ensures that the CCD is in the appropriate state. This ensures that all acquisitions are identical no matter how long a time has elapsed since data was last acquired (in fact the camera continually reads out the CCD to help prevent it from being saturated by light falling on it whilst it is not acquiring data). The camera also generates all the necessary pulses for shuttering and firing external sources. These pulses are accessed directly on the camera or via the Auxiliary Connector depending on the model. The Fire Output defines the position in time during which it is safe to allow a pulsed source to fire. The figure below illustrates the timing sequence for this mode of operation.



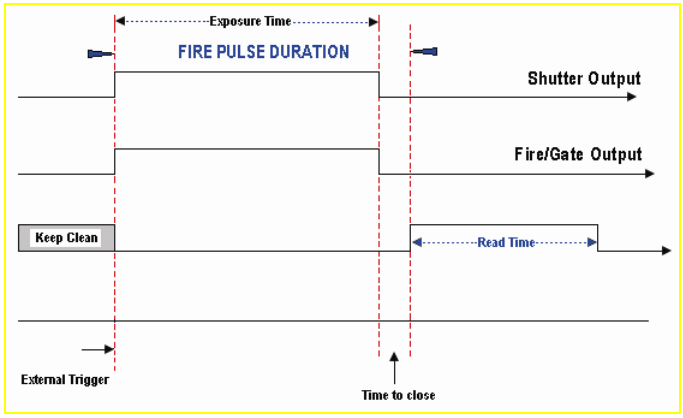
Internal Trigger Mode is ideal for situations where you are using 'continuous wave' (CW) light sources (an ordinary room light for instance) and incoming data, for the purposes of your observation, are steady and unbroken: thus you can begin acquisitions 'at will'.

You may use Internal Trigger Mode when you are able to send a trigger signal or 'Fire Pulse' to a short-duration, pulsed source (a laser, for example): in this case, initiating the data acquisition process can also signal the pulsed source to fire.

External

In **External Trigger Mode** once an acquisition has been started via the StartAcquisition function the camera is placed into a special dumping version of the 'Keep Clean' mode, which ensures that the CCD is not saturated before the external trigger occurs. Once the External Trigger is received the Keep Clean sequence is stopped and the acquisition is initiated.

The figure below illustrates the timing sequence for this mode of operation:



The external trigger can be fed in a number of ways:

- EXT TRIG socket of the **I/O Box** (available separately, model #IO160)
- **Pin 13** of the **Auxiliary Connector** on the Andor PCI Card
- The head in the case of iDus / iXon.

External Trigger mode is suited to data acquisitions involving a 'pulsed source' (e.g. a laser) where the source does NOT allow a trigger pulse to be sent to it but can generate one. It is possible to increase the frame rate when in external trigger mode by enabling the Fast External Trigger option, see SetFastExtTrigger.

When this option is enabled the system will not wait for a Keep Clean cycle to be completed before allowing an external trigger to initiate an acquisition. This may cause the background to change from one scan to another.

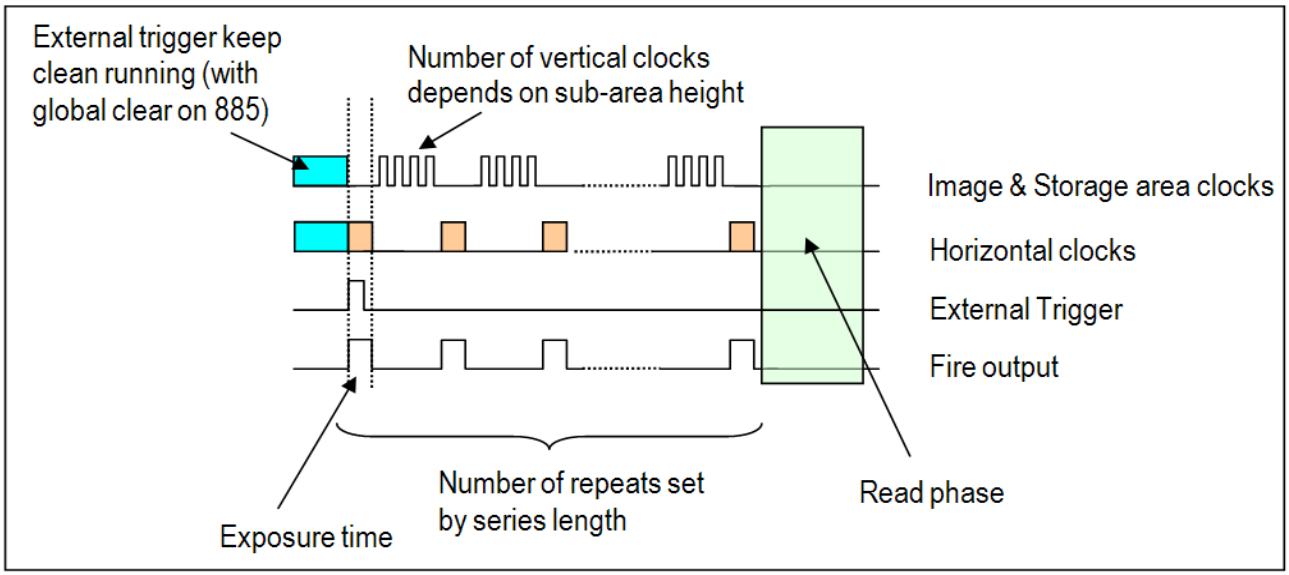
NOTES:

1. **If you have a shutter connected, and are using an external trigger, you must ensure that the shutter is open before the optical signal you want to measure occurs. When a camera is operated in frame transfer mode the external trigger sequence is different. Please refer to the camera user manual for a full description.**
2. **Some cameras may support the iCam technology. If they do, it will be fully operational in external trigger mode. It is very similar to the Software trigger functionality except that instead of a Software command instigating the acquisition, an external source does so. All the benefits described in the Software Trigger section can also be applied to the external trigger mode. It is set up in the same way with the same modes except that the trigger mode is set to External.**

Frame transfer is also fully functional in iCam External Trigger mode. When Frame Transfer is on it means that the Arm signal from the camera will be enabled during the current readout at a point to ensure the next exposure will end after the current readout is finished. This will give the fastest frame rate and also ensure that the next exposure cannot end until the previous one has been readout.

External Start

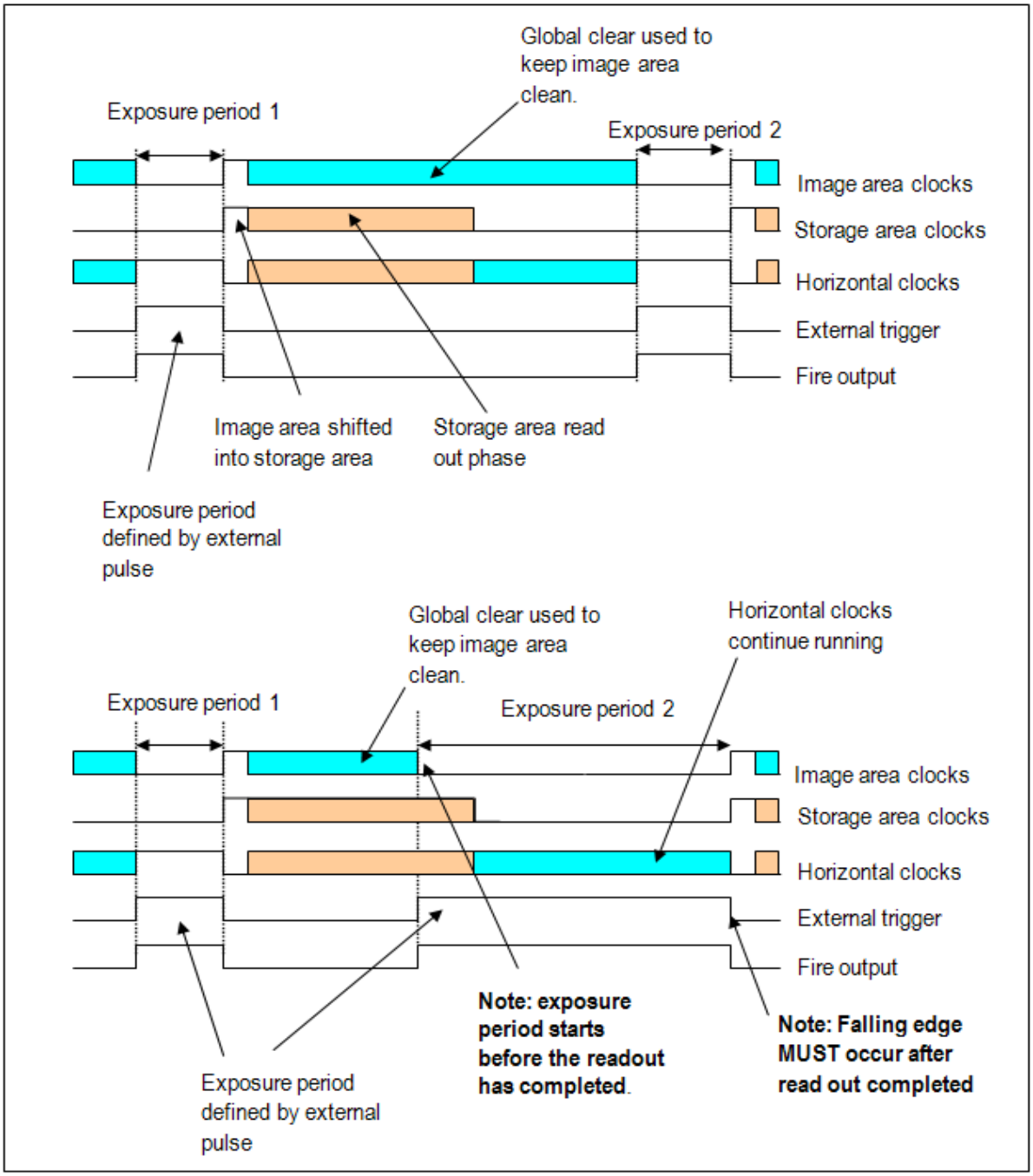
In **External Start Trigger Mode,** once an acquisition has been started via the StartAcquisition function, the camera system is placed into an external keep clean mode, which ensures that the CCD is not saturated before the external trigger occurs. Once the External Trigger is received, the Keep Clean sequence is stopped and the acquisition is initiated. After the initial acquisition the system will then continue to operate as in internal trigger mode. The figure below illustrates the timing sequence for this mode of operation.



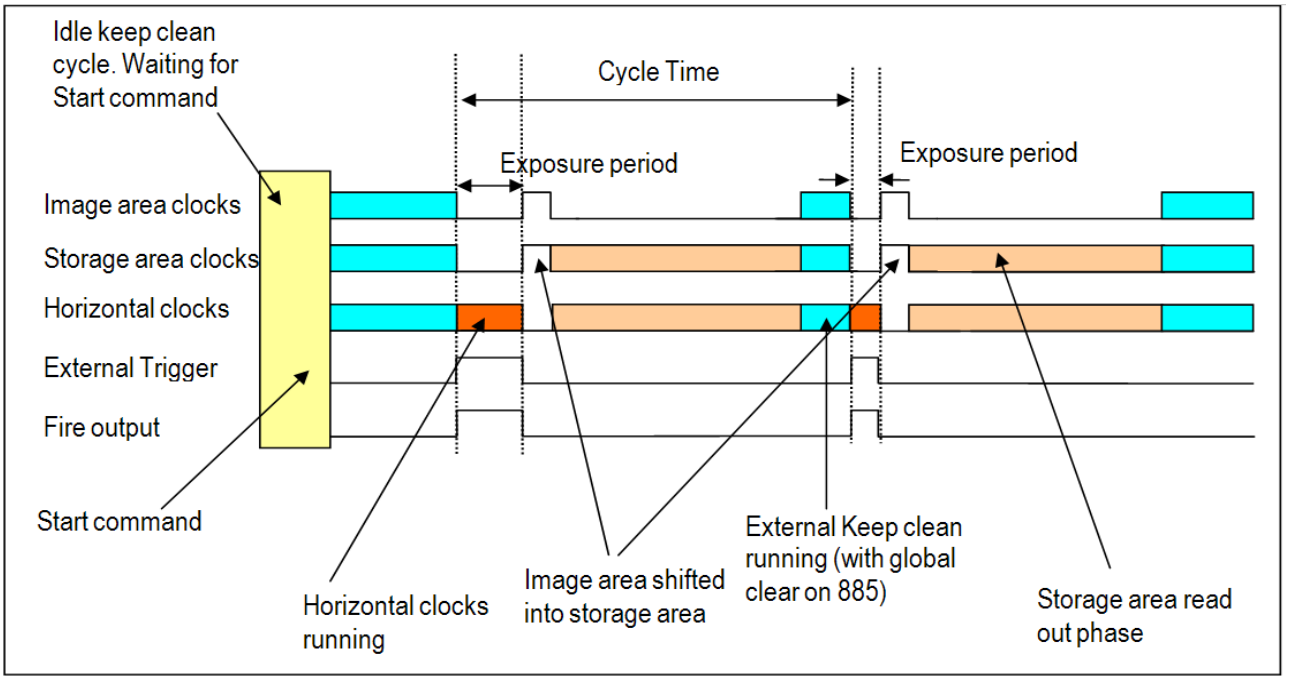
External Start trigger in Fast Kinetics mode

External Exposure

The **External Exposure** trigger is a mode of operation where the exposure time is fully controlled by the external trigger input. While the trigger input is high the CCD is accumulating charge in the Image area. When the external trigger goes low, the accumulated charge is quickly shifted into the Storage area and then read out in the normal manner. The figures below illustrate the timing sequences for this mode of operation.



External Exposure Trigger in Frame Transfer mode (885 model only)



External Exposure Trigger in Non-Frame Transfer mode

Note that not all systems support External Exposure mode. To check if this feature is available with your system, use the function GetCapabilities and check the ulTriggerModes variable for bit 5 (AC_TRIGGERMODE_EXTERNALEXPOSURE) being set. If this bit is set, please use the function GetCapabilities again and check the ulFeatures variable for bit 12 (AC_FEATURES_FTEXTERNALEXPOSURE) being set when Frame Transfer mode is used, and bit 13 (AC_FEATURES_KINETICEXTERNALEXPOSURE) being set when Kinetc and Frame Transfer modes are used together.

External FVB EM

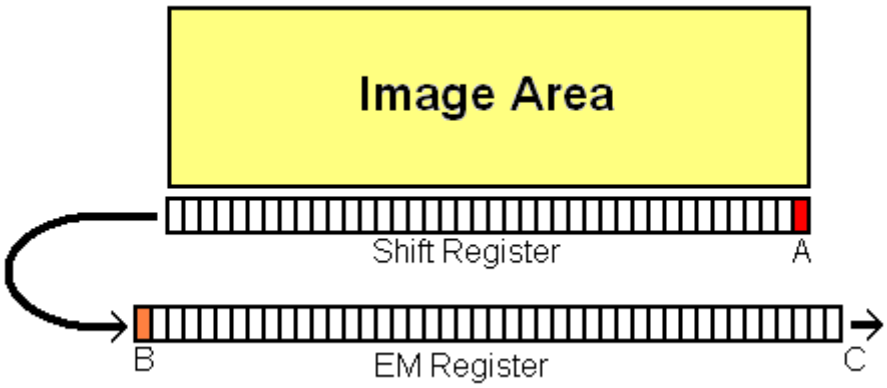
I**External FVB EM Trigger Mode** is much like operating an acquisition in FVB read mode with EM gain applied using external trigger with Keep cleans turned off. The difference surrounds the readout of the collected data and therefore the associated readout time:

When using EM gain a second (EM) register is used to apply the gain to the acquired data, The diagram below gives a quick overview of the readout process used in both processes.

Imagine a 'pixel' at position A.

Normally for the readout cycle to complete, this pixel will have to shift along the shift register and then along the entire length of the EM Register to C before the next acquisition can begin.

When using FVB EM Trigger Mode however, the EM Register Is used as a temporary storage area and so the pixel at A no longer needs to travel all the way to C but can stop at position B as this leaves sufficient space in the shift register for the next acquisition; the data is in effect 'pipelined'.



Note that not all systems support External FVB EM Trigger mode. To check if this enhanced feature is available with your system, use the function GetCapabilities and check the ulFeature variable for bit 10 (AC_FEATURES_KEEPCLEANCONTROL) being set.

Software

In **Software Trigger Mode,** once an acquisition has been started via the StartAcquisition function, the user software determines when data is actually acquired via the SendSoftwareTrigger command. This will give full control to the user software to ensure that it only requests an acquisition when it is ready. It permits the highly efficient upload of new exposure times between acquisitions and even allows a pre-load of up to 16 exposures to the camera which will be cycled through with each acquisition. It also permits the user software to perform certain actions before requesting the next acquisition, such as moving an external stage or even to change the exposure time.

Note that not all systems support Software Trigger mode. To check if these enhanced features are available with your system, use the function GetCapabilities and check the ulTriggerModes variable for bit 3 (AC_TRIGGERMODE_CONTINUOUS) being set. If this bit is set and the system is configured with the following modes:

- Read mode set to image
- Acquisition mode set to Run till abort
- Trigger mode set to 10

Then the SendSoftwareTrigger command will cause the acquisition to be taken.

It is recommended that you call IsTriggerModeAvailable(10) to check if your system is set up to use the SendSoftwareTrigger function.

If a SendSoftwareTrigger command is issued when the camera is not ready for it, it will be ignored and an appropriate return code returned.

The extra functionality of pre-loading exposures (up to a maximum of 16) to the camera is configured with the SetRingExposureTimes command. When the first acquisition is requested (SendSoftwareTrigger) the camera will take an acquisition with the first exposure in its list. When the second acquisition is requested the next exposure in the list will be used and so on. When the camera uses the final exposure in its list it will loop to the beginning again.

Notes on Frame Transfer

- On Frame Transfer systems, the Frame Transfer mode can be activated or deactivated. Currently, not all cameras can take advantage of the frame transfer operation in Software Trigger mode. By the nature of frame transfer, an exposure can be occurring when the previous acquisition is being read out. **Currently, no PCI connected cameras can be sent a software trigger when the camera is reading out**.
- USB cameras that support Software trigger can be sent a software trigger command during readout.
- Frame transfer is fully supported in external trigger mode.

SECTION 6 - SHIFT SPEEDS

The Andor system allows you to set the speed at which charge is shifted horizontally and vertically on the CCD.

The horizontal and vertical shift speeds are set via the SetHSSpeed and SetVSSpeed functions respectively.

The vertical shift speed is the speed at which each row on the CCD is shifted vertically into the Shift Register. The number of vertical shift speeds and their actual values are determined via the GetNumberVSSpeeds and GetVSSpeed functions.

The horizontal shift speed is the speed at which the charge in the shift register is shifted horizontally. It is also the speed at which the signal is digitized via the on board A/D converters. The number of horizontal shift speeds and their actual values are determined via the GetNumberHSSpeeds and GetHSSpeed functions. The horizontal shift speed is dependant on the CCD type and the model of plug-in card in the system. The shift speeds are always returned fastest first.

The following example retrieves the number of horizontal speeds allowed and their actual values in microseconds. Finally, it selects the fastest speed as follows:

```
GetNumberHSSpeeds(0, 0, &a); //first A-D, request data speeds for (I = 0; I < a;I++)
GetHSSpeed(0, 0, I, &speed[I]);
SetHSSpeed(0, 0);  /* Fastest speed */
```

SECTION 7 - SHUTTER CONTROL

SHUTTER MODES

In the sections on Acquisition modes and Readout modes the use of a shutter was highlighted to prevent the smearing of data. Smearing occurs if light is allowed to fall on to the CCD while the pixel charges are being binned into the shift register prior to readout. The Andor system has a dedicated shutter control line that ensures that the shutter is correctly operated at all times.

The SetShutter and SetShutterEx functions provide you with a selection of options that determine when and how a shutter should be used.

Fully Auto

Fully Auto is the simplest shutter mode because it leaves all shuttering decisions to the Andor system. The shutter opens and closes automatically in accordance with any acquisition parameters you have set.
This option will automatically provide suitable shuttering for the majority of data acquisitions.

Hold Open

If the shutter mode is set to **Hold Open** the shutter will be open before, during and after any data acquisition. Choose this option if you wish to take a series of acquisitions with the shutter opened at all times (e.g. if you are taking a series of acquisitions with a pulsed source with little or no background illumination).

Hold Closed

If the shutter mode is set to **Hold Close** the shutter remains closed before, during and after any data acquisition. Choose this option if you wish to take an acquisition in darkness (e.g. if you are acquiring a background scan).

SHUTTER TYPE

The shutter control line is a TTL compatible pulse, which can be either active high or active low to allow the control of an external shutter.

NOTE: If the camera has an internal shutter (the function IsInternalMechanicalShutter can be used to test this) but cannot control the internal and external shutter independently (check the capability AC_FEATURES_SHUTTEREX) then the TTL pulse will always be active high.

- If you set the shutter type to **TTL High** with SetShutter or SetShutterEx, the Andor SDK will cause the output voltage to go 'high' to open the shutter.
- If you set the shutter type to **TTL Low** with SetShutter or SetShutterEx, the Andor SDK will cause the output voltage to go 'low' to open the shutter.

For Classic systems this pulse will be sent through the Andor PCI card. For other systems this pulse will be sent through the shutter SMB connector on the camera.

The documentation supplied by the shutter manufacturer will advise the user whether your shutter opens at a high or a low TTL level.

NOTE: With Full Vertical Binning there is no shutter pulse. The shutter will always be in the Open position. See Shutter Mode on the previous page and Shutter Transfer Time on the next page.

The **I/O Box** also contains a 30V shutter jack socket, which produces the same signal as the TTL output but is always high to open (see User Guide for further details). **NOTE: Only applicable to classic systems.**

For iXon+ cameras that have independent shutter control (capability AC_FEATURES_SHUTTEREX) we can control the TTL type and mode of the internal (if available) and external shutter independently using function SetShutterEx, The external shutter signal will be output through the Shutter SMB port on the rear of the camera. The internal and external shutters will have the same opening and closing times.

SHUTTER TRANSFER TIME

Mechanical shutters take a finite time to open or close. This is sometimes called the **Shutter Transfer Time** and can be of the order of tens to hundreds of milliseconds. The Transfer Time is important for many reasons.

Firstly, if your shutter takes 40ms to open and you specify an exposure time of 20ms then the shutter will simply not get the time to open fully. Similarly, if you are triggering a pulse light source via the Fire pulse then you will want to ensure that the Fire pulse goes high only when the shutter is opened. Also, if you are acquiring data in an imaging mode (Multi-Track, Random-Track, Single-Track or Image), with either a continuous light source or a large high background illumination with a pulsed source, the shutter must be fully closed before readout begins. Otherwise, a smeared image will result.

The SetShutter and SetShutterEx functions allow you to specify a Transfer Time for both opening and closing the shutter.

The time you specify for the shutter **opening time** will affect the minimum exposure time you can set via the SetExposureTime function. For example, if you set the opening time to 0ms then the minimum exposure time will be set to the amount of time needed to clean the shift register on the CCD. However, if the **opening time** is set to a larger value than is needed to clean the shift register, say 50ms, then the minimum exposure time will be 51ms i.e. 1ms more than the time needed to open the shutter.

The SetExposureTime is in effect setting the length of time the shutter output will be in the '**open**' state. The rising edge of the **Fire** output signal follows the start of the shutter open state after a delay, equal to the value you set for the opening time via the SetShutter functions.

Andor SDK also automatically adds the Transfer Time for the closing of the shutter to the end of the acquisition sequence, introducing an appropriate delay between the start of the shutter 'closed' state and the commencement of the data being read out. This value is set via the **closing time** parameter in the SetShutter and SetShutterEx functions.

Figures 10 & 11 on the next page show the timing sequence for both Internal and External triggering modes.

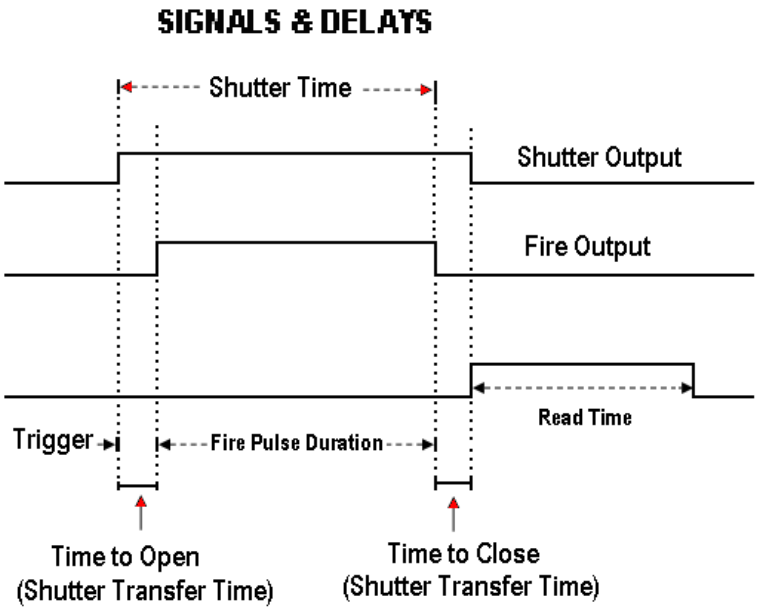


Figure 9: Timing diagram for shutter and fire pulses in internal trigger mode

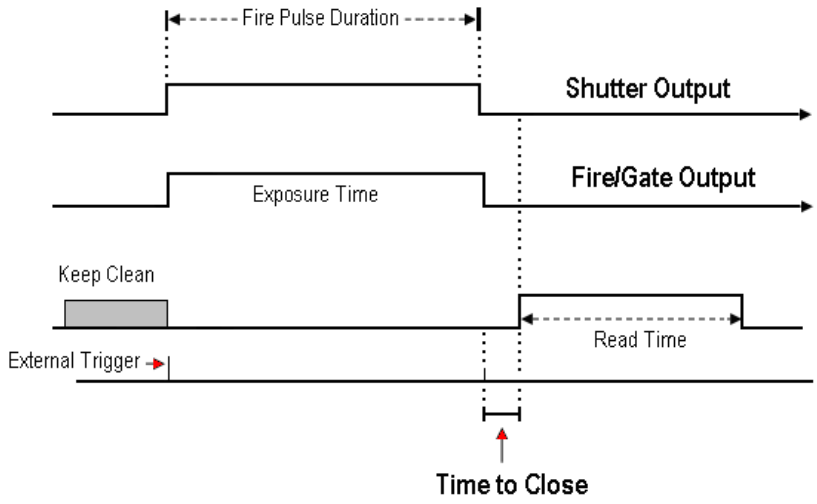


Figure 10: Timing diagram for shutter and fire pulses in external trigger mode

NOTES:

1. **In the case of external triggering, the external trigger pulse, the shutter pulse and the fire pulse are all coincident. If you are using a shutter and externally triggering the Andor system then the external trigger must be pulsed early enough to ensure that the shutter is fully opened before the light pulse arrives. Please consult the documentation supplied by the shutter manufacturer to get an indication of the transfer time you can expect from your particular shutter.**
2. **If you do not have a shutter connected, set the Closing Time and Opening Time parameters to 0. Setting these parameters to any other value will insert extra delays into cycle time calculations.**

SECTION 8 - TEMPERATURE CONTROL

The Andor camera incorporates a CCD, which is fabricated using a process known as Multi-Pin Phasing (MPP). As a result the dark current is reduced by a factor of approximately 100 compared to standard devices at the same temperature. To reduce the dark current even further Andor SDK allows you to cool and monitor the CCD temperature through a number of functions. The desired temperature is set via the SetTemperature function whilst the actual cooling mechanism is switched On and Off via the CoolerON and CoolerOFF functions.

The table below shows a typical example of temperatures attainable with the various systems available, with and without the assistance of water-cooling. Please refer to the specification supplied with your particular model for full details. The possible temperature range available to the SetTemperature function can be obtained using the GetTemperatureRange function.

Moderate Cooling		High Cooling		Ultra-High Cooling	
Air	Water	Air	Water	Air	Water
-5°C	-25°C	-30°C	-55°C	-75°C	-90°C

NOTES:

1. **Because rapid cooling and heating can cause thermal stresses in the CCD the rate of cooling and heating is regulated to be <10°C per minute on some systems.**
2. **While the system is cooling, or heating, you can acquire data but the 'Background Level' WILL change with temperature. The current temperature can be read using the GetTemperature function. This function also returns the status of any cooling process including whether the cooler is ON or OFF.**
3. **If the GetTemperature function returns the DRV_TEMP_STABILIZED status flag then the temperature is within 3°C of the set temperature and the microprocessor is no LONGER regulating the cooling rate. At this point the temperature regulation is controlled via analog electronics.**

SECTION 9 - SPECIAL GUIDES

CONTROLLING MULTIPLE CAMERAS

Using the SDK It is possible to control multiple Andor cameras. The following SDK functions permit the selection and use of one Andor camera at a time.

- GetAvailableCameras
- GetCameraHandle
- SetCurrentCamera
- GetCurrentCamera
- Initialize

***NOTE: If only one camera is available it is not necessary to use any of these functions since that camera will be selected by default.**

A maximum of eight cameras can be controlled by the SDK. This can be a combination of USB and PCI cameras but the maximum number of PCI cameras that can be supported is two.

While using more than one camera the other SDK functions are used in the normal way. When a function is called it only affects the currently selected camera and is not sent to all cameras. This allows each camera to be programmed individually but it also means that each camera has to be individually initialized and shut down.

Another aspect of this control method is that cameras cannot be simultaneously triggered using the software - if simultaneous triggering is required then external triggers should be used.

USING MULTIPLE CAMERA FUNCTIONS

The GetAvailableCameras function is used to return the number of Andor cameras available. A handle for each camera is obtained using the GetCameraHandle function (this handle should be stored for the lifetime of the application).

Any of the available cameras can then be selected by calling the SetCurrentCamera function and passing in the camera handle. Once a camera has been selected any other SDK function can be called as normal but it will only apply to the selected camera. Initialize must be called once for each camera that you wish to use. At any stage the GetCurrentCamera function can be called and it will return the handle of the currently selected camera.

NOTE:

1. **It is not possible to unplug any cameras or plug in new ones during the lifetime of the application.**
2. **It is not possible to trigger cameras simultaneously using software. To simultaneously trigger more than one camera external triggers can be used or alternatively one camera can be triggered by software and the fire pulse from this camera used to trigger the others.**
3. **Currently, if only one camera is installed there is no need to obtain the camera handle or select it since this camera will be used by default.**

This example pseudo code demonstrates how to use the functions relating to the operation of multiple cameras:

```
//
// Multiple Camera Pseudo Code Example
// Note: This code does not compile
//
// This example demonstrates how to: -
//  1. Determine the number of cameras available
//  2. Obtain a handle for each camera
//  3. Initialize each camera
//  4. Perform a single scan acquisition with each camera
//  5. Check which camera is currently selected
//  6. Shut down each camera
//

// Start of program

// Determine the number of cameras available
GetAvailableCameras(NumberOfCameras)

// Allocate memory for NumberOfCameras handles
long CameraHandles[NumberOfCameras]

// Obtain a handle for each camera and initialize
for (index = 0 to NumberOfCameras-1)
{
  GetCameraHandle(index, CameraHandles[index])
  SetCurrentCamera(CameraHandles[index])
  Initialize(...)
}

// Set an exposure time for each camera and start the acquisition
for (index = 0 to NumberOfCameras-1)
{
  SetCurrentCamera(CameraHandles[index])
  SetAcquisitionMode(1)
  SetExposureTime(...)

  // Any other camera settings

  StartAcquisition()

  // Wait until acquisition has finished
  ...
  ...
}

// Check which camera is currently selected
long UnknownCameraHandle
GetCurrentCamera(UnknownCameraHandle)

// Shut down each camera
for (index = 0 to NumberOfCameras-1)
{
  SetCurrentCamera(CameraHandles[index])
  ShutDown()
}

// End of program
```

Figure 11: Example of Multiple Camera Pseudo Code

DATA RETRIEVAL METHODS

How to determine when new data is available

There are a wide of range of functions available for retrieving data from the camera. Deciding which functions should be used depends on whether the data will be retrieved during an acquisition or once the acquisition is complete. See Retrieving Image Data

For certain cases it may be useful to know what stage an acquisition is at. The GetStatus function can be used to get the current status of the acquisition. It will return information such as, the acquisition is in progress or it is finished. See GetStatus for full list of return information.

Another way to know if an acquisition is finished is with the WaitForAcquisition function. When an acquisition is started, the WaitForAcquisition function can be called, it does not return from this function until the acquisition is finished. The function can be cancelled by calling the CancelWait function although this will require the user application to be multi-threaded.

```
//
// WaitForAcquisition Pseudo Code Example
// Note: This code does not compile
//

// Start of program

// Initialize camera
Initialize(...)

// Start the acquisition
StartAcquisition()

// Wait for the acquisition to complete
WaitForAcquisition()

// Retrieve data
...

// Shut down camera
ShutDown()

// End of program
```

Figure 12: Example of WaitForAcquisition Pseudo Code

The SetDriverEvent function can be used in conjunction with event handles. If an event is created using the WIN32 CreateEvent function and passed to the SDK using the SetDriverEvent function an event handle now exists which the SDK can use to inform the application that something has occurred.

To ensure that the event has been set by a new image arriving and not something else (e.g. temperature change) the GetTotalNumberImagesAcquired function can be used. This function will return the total number of images acquired and transferred to the Andor SDK, and which are now available to be retrieved by the user.(see section Retrieving Image Data). Comparing the new value to a previously stored one is an effective way of checking that there are new images available.

```
//
// SetDriverEvent Pseudo Code Example
// Note: This code does not compile
//

// Start of program

// Initialize camera
Initialize(...)

// Create an event handle
HANDLE hEvent = CreateEvent()

// Set the driver event
SetDriverEvent(hEvent)

// Start the acquisition
StartAcquisition()

// Wait for the acquisition to complete
WaitForSingleObject()

// Retrieve data
...

// Shut down camera
ShutDown()

// End of program
```

Figure 13: Example of SetDriverEvent Pseudo Code

Retrieving Image Data

Depending on the image settings there may be more than one image available after each notification. It is important to ensure that all of the new images are retrieved if they are required. The recommended functions for retrieving image data are as follows:

- **GetOldestImage**
- **GetMostRecentImage**
- **GetImages**
- **GetAcquiredData**

GetOldestImage, GetMostRecentImage, and GetImages are used to retrieve data from an internal 48MB circular buffer that is written to by all acquisition modes. They are particularly useful for retrieving data while an acquisition is taking place especially during run till abort mode but can also be used when the acquisition is complete. For all acquisition modes (except Run Till Abort) the GetAcquiredData function can be used to retrieve all the acquired data once the acquisition is complete.

NOTE: All functions mentioned here refer to retrieving 32-bit data but there are also 16-bit versions of these functions available.

GetOldestImage will retrieve the oldest available image from the circular buffer. Once the oldest image has been retrieved it is no longer available and calling GetOldestImage again will retrieve the next image. This is a useful function for retrieving a number of images. For example if there are 5 new images available, calling GetOldestImage 5 times will retrieve them all. GetMostRecentImage will retrieve the most recent image from the circular buffer. This provides a method for displaying the most recent image on screen while the acquisition is in progress (should be used in conjunction with the GetNumberNewImages function).

The GetNumberNewImages function returns the start and end index of the images that are available in the circular buffer. These indexes should be used along with the GetImages function to retrieve all of the available data. This provides an effective way of retrieving a number of new images in one function call.

GetAcquiredData should be used once the acquisition is complete to retrieve all the data from the series. This could be a single scan or an entire kinetic series.

DETERMINING CAMERA CAPABILITIES

Retrieving capabilities from the camera

It is important to be able to determine the capabilities of the camera. This allows the user to take the full benefit of all the features available.

There are a number of functions available which can be used to obtain this information and these can be found in the following areas of this section.

- **Horizontal Pixel Shift Capabilities**
- **Vertical Pixel Shift Capabilities**
- **Other Capabilities**

Horizontal Pixel Shift Capabilities

Depending on the camera type and model there will be variations in the number of A/D channels, the number of Output Amplifiers, the number & range of Horizontal Shift Speeds and the number & range of Pre-Amp Gains. The first step in this process is to determine the following:

- Number of A/D channels using the **GetNumberADChannels** function
- Number of output amplifiers using the **GetNumberAmp** function
- Maximum number of pre-amp gains using the **GetNumberPreAmpGains** function

NOTE: Not all PRE-AMP gains are available for each horizontal shift speed. The IsPreAmpGainAvailable **function is used to determine which are valid for a particular horizontal shift speed and this will be explained later.**

The bit depth of each A/D channel can be found using the GetBitDepth function.

Once this information has been obtained the next step is to find the number of available horizontal shift speeds for each output amplifier on each A/D channel using the GetNumberHSSpeeds function. Following this the value of each horizontal shift speed can be found using the GetHSSpeed function.

Each horizontal shift speed has an associated number of valid pre-amp gains. The next step is to obtain the value of each pre-amp gain using the GetPreAmpGain function. Not all pre-amp gains are available for each horizontal shift speed so using the IsPreAmpGainAvailable function it is possible to check which pre-amp gains are valid. Once the information has been retrieved the relevant selections can be made using the functions that follow:

- **SetADChannel**
- **SetOutputAmplifier**
- **SetHSSpeed**
- **SetPreAmpGain**

An example of the pseudo code for this capability is shown here:

```
//
// Horizontal Pixel Shift Pseudo Code Example
// Note: This code does not compile
//

// Start of program

// Initialize camera
Initialize(...)

long NumChannels, NumAmp, NumPreAmpGains
long BitDepth, NumHSpeeds, IsPreAmpAvailable
float HSSpeed

GetNumberADChannels(NumChannels)
GetNumberAmp(NumAmp)
GetNumberPreAmpGains(NumPreAmpGains)

for (i = 0 to NumChannels-1)
{
  GetBitDepth(i, BitDepth)
  for (j = 0 to NumAmp-1)
  {
    GetNumberHSSpeeds(i, j, NumHSpeeds)
    for (k = 0 to NumHSpeeds)
    {
      GetHSSpeed(i, j, k, HSSpeed)
      for (m = 0 to NumPreAmpGains-1)
      {
        GetPreAmpGain(m, PreAmpGain)
        IsPreAmpGainAvailable(i, j, k, m, IsPreAmpAvailable)
      }
    }
  }
}

// Shut down camera
ShutDown()

// End of program
```

Figure 14: Example of Horizontal Pixel Shift Pseudo Code

Vertical Pixel Shift Capabilities

Depending on the camera type and model there will be variations in the number of Vertical Shift Speeds available.

The first step in this process is to determine the number of vertical shift speeds using the GetNumberVSSpeeds function. Following this the value of each vertical shift speed can be found using the GetVSSpeed function.

Since the camera may be capable of operating at more than one vertical shift speed the GetFastestRecommendedVSSpeed function will return the index and the value of the fastest recommended speed available. The very high vertical shift speeds may require an increase in the amplitude of the vertical clock voltage using the SetVSAmplitude function.

The GetFastestRecommendedVSSpeed function returns the fastest speed which does not require the vertical clock voltage to be adjusted. If the fastest recommended speed is selected the vertical clock voltage should be set as normal.

NOTE: Exercise caution when increasing the amplitude of the Vertical Clock voltage, since higher clocking voltages may result in increased Clock-Induced Charge in your signal. In general, only the very highest speeds are likely to benefit from increased vertical clock voltage amplitude.

Once the information has been retrieved the relevant selections can be made using these functions:

- **SetVSSpeed**
- **SetVSAmplitude**

An example of the pseudo code for this capability is shown in figure 15:

```
//
// Vertical Pixel Shift Pseudo Code Example
// Note: This code does not compile
//

// Start of program

// Initialize camera
Initialize(...)

long NumVSSpeeds, RecommendedVSSpeedIndex
float VSSpeed

GetNumberVSSpeeds(NumVSSpeeds)
GetFastestRecommendedVSSpeed(RecommendedVSSpeedIndex, VSSpeed)

for (i = 0 to NumChannels-1)
{
  GetVSSpeed(i, VSSpeed)
}

// Shut down camera
ShutDown()

// End of program
```

Figure 15: Example of vertical pixel shift pseudo code

Other Capabilities

Other information about the camera can be obtained using the following functions:

- **GetCapabilities**
- **IsInternalMechanicalShutter**

The GetCapabilities function populates an **AndorCapabilities** structure with information associated with the camera. Afterwards this structure can be used to determine details about the camera e.g. supported acquisition modes, supported trigger types.

The IsInternalMechanicalShutter function is used to determine if the camera has an internal mechanical shutter.

Depending on the camera type and model there will be variations Output Amplifiers that can be applied to your acquisition:

- **EMCCD Gain**
- **Extended NIR**
- **High Capacity**

By using the GetCapabilities function you can determine which amplifiers are available to your camera, the ulSetFunctions field will return the revevent information.

EMCCD Gain

EMCCD is a quantitative digital camera technology that is capable of detecting single photon events whilst maintaining high Quantum Efficiency, achievable by way of a unique electron multiplying structure built into the sensor. If ulSetFunctions bit 5 returns 1 then EM Gain can be set by either the SetOutputAmplifier or the SetHSSpeed functions. (figure 16).

```
//
//EMCCD Gain Pseudo Code Example
//Note: This code does not compile
//

//Start of program

//Initialize camera
Initalize(...)

//Using SetHSSpeed()
SetHSSpeed(0,0);

//
//    OR
//

//Using SetOutputAmplifier()
SetOutputAmplifier(0);

//Shut down camera
Shutdown()

//End of Program
```

Figure 16: Example of EMCCD Gain Pseudo Code

Extended NIR

When using Extended is increased. This in turn increases the response of the sensor to these wavelengths with a QE increase from 40% Near Infra-Red Mode the increased thickness of the silicon on which the CCD is formed and by manipulating the voltages applied to the silicon substrate, the depth of the region where red and NIR light can convert to photoelectrons to 60% at 650 nm. If ulSetFunctions bit 7 returns 1 then Extended NIR mode can be activated by using either the SetOutputAmplifier or the SetHSSpeed functions. (figure 17).

```
//
//Extended NIR Pseudo Code Example
//Note: This code does not compile
//

//Start of program

//Initialize camera
Initalize(...)

//Using SetHSSpeed()
SetHSSpeed(1,0);

//
//    OR
//

//Using SetOutputAmplifier()
SetOutputAmplifier(1);

//Shut down camera
Shutdown()

//End of Program
```

Figure 17: Example of Extended NIR Pseudo Code

High Capacity

With High capacity enabled the responsivity of the sensor is reduced thus allowing the reading of larger charge packets during binning operations. If ulSetFunctions bit 7 returns 1 then High Capacity mode can be activated bu using the SetHighCapacity function.
(figure 19).

```
//
//High Capacity Pseudo Code Example
//Note: This code does not compile
//

//Start of program

//Initialize camera
Initalize(...)

//Enable High Capacity
SetHighCapacity(1);

//Shut down camera
Shutdown()

//End of Program
```

Figure 18: Example of High Capacity mode Pseudo Code

iCam

iCam technology is a combined firmware and software innovation that has been incorporated into Andor's EMCCD imaging cameras. iCam offers enhanced performance for acquisitions whether software triggered or hardware (externally) triggered, with absolute minimal overheads. It allows for faster frame rates in software by dedicated timing patterns that eliminate unnecessary overhead times. This, alongside the bi-directional communication between camera and PC, facilitates unparalleled synchronization with other peripheral equipment. A ring mode offers the capacity to use up to 16 different timing patterns uploaded into the camera head, thus trigger events can yield virtually instantaneous switching between exposure channels.

This new functionality has been added to the Run Till Abort acquisition mode and currently will only operate with Image readout mode. Cameras must contain a suitable firmware and if a PCI card is being used it must be a CCI-23 card and have a suitable firmware loaded. If you are unsure if your current Hardware is iCam compatible please download the 'iCam compatibility checker' from andor.com.

It will operate in Software and External trigger mode, with both Frame Transfer and Non Frame Transfer mode.

The idea behind this is that the SDK puts the camera into a 'heightened state of readiness' and when a trigger comes (either software or hardware) the acquisition can be taken immediately.

If you hardware is compatible and needs to be upgraded please contact productsupport@andor.com for a further application which will upgrade your system.

iCam technology is a combined firmware and software innovation that has been incorporated into Andor's EMCCD imaging cameras. iCam offers enhanced performance for acquisitions whether software triggered or hardware (externally) triggered, with absolute minimal overheads. It allows for faster frame rates in software by dedicated timing patterns that eliminate unnecessary overhead times. This, alongside the bi-directional communication between camera and PC, facilitates unparalleled synchronization with other peripheral equipment. A ring mode offers the capacity to use up to 16 different timing patterns uploaded into the camera head, thus trigger events can yield virtually instantaneous switching between exposure channels.

This new functionality has been added to the Run Till Abort acquisition mode and currently will only operate with Image readout mode. Cameras must contain a suitable firmware and if a PCI card is being used it must be a CCI-23 card and have a suitable firmware loaded. If you are unsure if your current Hardware is iCam compatible please download the 'iCam compatibility checker' from andor.com.

It will operate in Software and External trigger mode, with both Frame Transfer and Non Frame Transfer mode.

The idea behind this is that the SDK puts the camera into a 'heightened state of readiness' and when a trigger comes (either software or hardware) the acquisition can be taken immediately.

If you hardware is compatible and needs to be upgraded please contact productsupport@andor.com for a further application which will upgrade your system.

This is a unique interface whereby a user can choose from a pre-determined list of camera set-up configurations. The user need only choose how they would like their camera to be optimized, e.g. for 'Sensitivity and Speed', 'Dynamic Range and Speed', 'Time Lapse'. Parameters such as EM gain value, vertical shift speed, vertical clock amplitude, pre-amp sensitivity and horizontal readout speed will then be optimized accordingly, 'behind the scenes'. Furthermore, the option exists to create additional user-defined configurations. Pre-defined OptAcquire modes include:

1. Sensitivity and Speed (EM Amplifier)
Optimized for capturing weak signal at fast frame rates, with single photon sensitivity. Suited to the majority of EMCCD applications.
2. Dynamic Range and Speed (EM Amplifier)
Configured to deliver optimal dynamic range at fast frame rates. Moderate EM gain applied.
3. Fastest Frame Rate (EM Amplifier)
For when it's all about speed! Optimized for absolute fastest frame rates of the camera. Especially effective when combined with sub-array/binning selections.
4. Time Lapse (EM Amplifier)
Configured to capture low light images with time intervals between exposures. Overlap ('frame transfer') readout is deactivated.
5. Time Lapse and Short Exposures (EM Amplifier)
Configured to minimize vertical smear when using exposure times less than 3ms.
6. EMCCD Highest Dynamic Range (EM Amplifier)
Combines EMCCD low light detection with the absolute highest dynamic range that the camera can deliver. Since this requires slower readout, frame rate is sacrificed.
7. CCD Highest Dynamic Range (Conventional Amplifier)
Optimized for slow scan CCD detection with highest available dynamic range. Recommended for brighter signals *OR* when it is possible to apply long exposures to overcome noise floor.
8. Photon Counting EM
Configuration recommended for photon counting with individual exposures < 10sec.
9. Photon Counting with Long Exposures (> 1sec)
Configuration recommended for photon counting with individual exposures > 1sec.

The following list details the valid acquisition parameters and input values for use with OptAcquire functions.

Parameter: output_amplifier

Type: String

Valid Values: "Conventional or "Electron Multiplying".

Parameter: frame_transfer

Type: String

Valid Values: "ON" or "OFF".

Parameter: readout_rate

Type: Float

Valid Values: A valid and supported value which can be retrieved by subsequent calls to GetNumberHSSpeeds() and GetHSSpeed().

Parameter: shift_speed

Type: Float

Valid Values: A valid and supported value which can be retrieved by subsequent calls to GetNumberVSSpeeds() and GetVSSpeed().

Parameter: electron_multiplying_gain

Type: Integer

Valid Values: A valid and supported value which can be retrieved from a call to GetEMGainRange().

Parameter: vertical_clock_amplitude

Type: Integer

Valid Values: A valid and supported integer value in the range 0 – 4.

Parameter: preamplifier_gain

Type: Integer

Valid Values: A valid and supported value which can be retrieved from subsequent calls to GetNumberPreAmpGains() and GetPreAmpGain().

An example of the pseudo code for using OptAcquire is shown in figure 16 and figure 17:

```
//
// Example use of OptAcquire Functions
// Start OptAcquire using the Preset Modes
//

// Initialise, specifying a user xml file to use.  The file does not need
// to exist at this stage however a file name must be provided.
// This must be the first call before using any other OptAcquire function
OA_Initialize("MyFile.xml", FileNameLength);

// Get Number of Preset Modes
OA_GetNumberOfPreSetModes(NumberOfModes);

// Get all the Mode Names available, allocate enough space to retrieve all the mode names
// i.e. buffer to retrieve list of modes must be capable of storing the maximum number of
// characters allowed for a mode name (255) * number of modes
OA_GetPreSetModeNames(ListOfPresetModeNames);

// Get the number of acquistion parameters associated with a particular mode
OA_GetNumberOfAcqParams(ModeName, NumberOfParams);

// Get the name of all the acquisition parameters for a particular mode
// i.e. buffer to retrieve list of modes must be capable of storing the maximum number of
// characters allowed for a parameter (255) * number of params
OA_GetModeAcqParams(ModeName, ListOfParams);

// Get the details of each of the acquisition parameters for the mode
OA_GetInt (ModeName, "electron_multiplying_gain", IntValue);
OA_GetFloat (ModeName, "readout_rate", FloatValue);
OA_GetString (ModeName, "output_amplifier", StringValue, StringLen);
....

// Enable a Mode
OA_EnableMode(ModeName);
```

Figure 16: Example of OptAcquire using Preset Modes

```
// Adding a New Mode to a User File

// Initialise, specifying a user xml file to use.  The file does not need
// to exist at this stage however a file name must be provided.
// This must be the first call before using any other OptAcquire function
OA_Initialize("MyFile.xml", FileNameLength);

// Add a new mode, maximum length for mode name is 255 characters and also
// for mode description is 255 characters
OA_AddMode(ModeName, ModeNameLength, ModeDescription, ModeDescriptionLength);

// Define the Acquisition Parameters which make up the new mode.  Acquistion
// parameters must be valid parameters supported by OptAcquire
OA_SetInt (ModeName, "electron_multiplying_gain", IntValue);
OA_SetFloat (ModeName, "readout_rate", FloatValue);
OA_SetString (ModeName, "output_amplifier", StringValue, StringLen);
....

// Write the new mode to file
// Note: Modes CANNOT be written to the Preset file
OA_WriteToFile("MyFile.xml", FileNameLen);
```

Figure 17: Example of OptAcquire Adding a New Mode

SECTION 10 - EXAMPLES

INTRODUCTION

We present here a number of examples of controlling Andor SDK to acquire data. Source code for each example can be found on the disk. Each example is presented in three different languages, Visual Basic, LabVIEW and C.

The examples were devised to demonstrate the wide versatility and range of the data acquisition mechanisms available with Andor SDK. The examples are all based on variations of the flowchart described on the following pages.

The flowchart is a basic demonstration of how to set up and control the Andor system to acquire data with the appropriate Andor SDK commands located just to the right of the flowchart.

The flowchart is divided into three sections, the first deals with the initialization of the system and controlling the sensor temperature. The second section deals with the data acquisition process while the third illustrates the proper shutdown procedure.

NOTE: Do not have more than one example or other SDK software (e.g. Andor Solis™, iQ™) running at the same time.

RUNNING THE EXAMPLES

C

The C examples are supplied as ready to run executable files (both 32-bit and 64-bit) and with complete source code. The code has been tested with Microsoft VC++ 5.0 and Borland Developer Studio 2006.

You are free to modify the example source code in the "C" directory to be compatible with your own compiler.

In order to compile your own C or C++ programs you will need the following files:

- **ATMCD32D.H** C Header File
- **ATMCD32D.LIB** / **ATMCD64D.LIB** Import Library (Borland compatible)
- **ATMCD32M.LIB / ATMCD64M.LIB** Import Library (Microsoft compatible)

LabVIEW

The LabVIEW examples are contained in the sub-directory "LabVIEW" of the installation directory. The LabVIEW examples are in the form of VI's and **must** be run through LabVIEW 7.0 or higher (32-bit).

Visual Basic

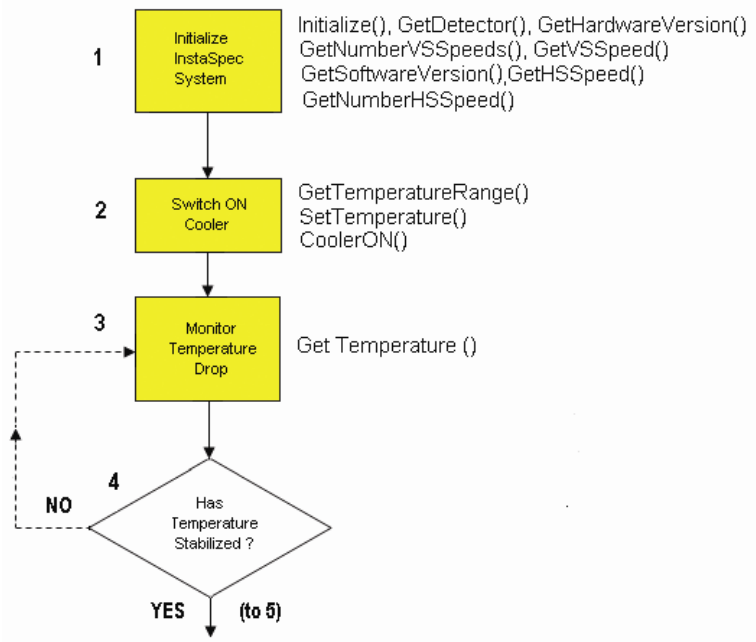
The Visual Basic examples are contained in the sub-directory **VBasic** of the installation directory. Each example contains all the source code, forms and project files to re-build executable files.

Each of the Visual Basic examples comes with a ready to run executable file.
When building you own projects you must include the file **ATMCD32D.BAS**. This file contains the Andor SDK function prototypes for interfacing with the dynamic link library **ATMCD32D.DLL**

NOTE: To run any of the examples you will need the following files:

- **ATMCD32D.DLL** / **ATMCD64D.DLL** (depending on system)
- **DETECTOR.INI: Contains initialization information (not required on iDus, iXon or Newton systems)**

FLOW CHART OF THE FUNCTION CALLS NEEDED TO CONTROL ANDOR CAMERA



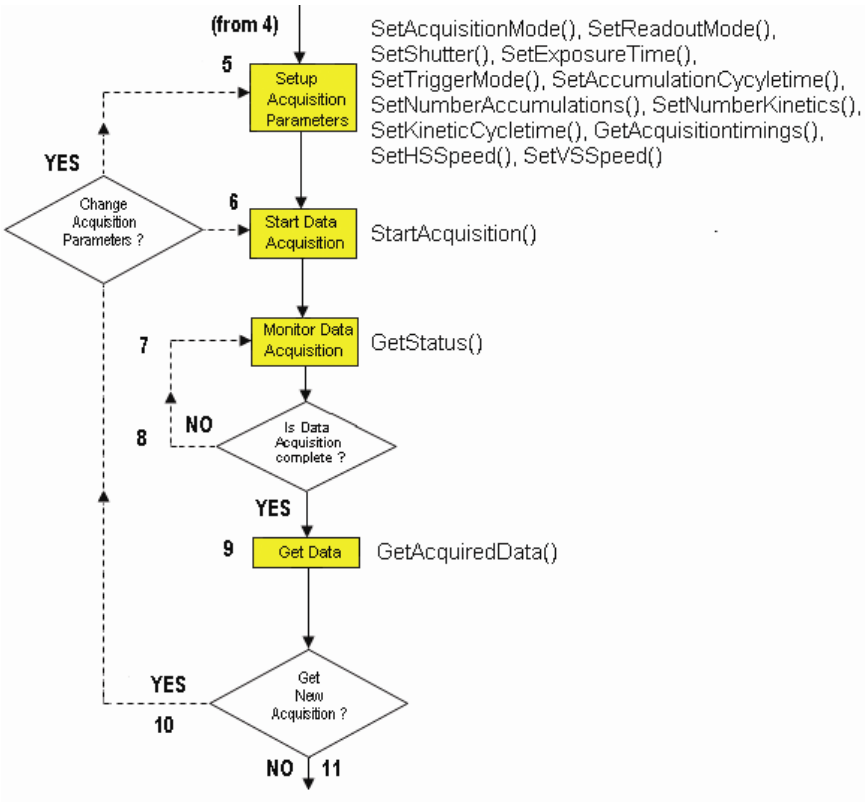
1. The application initializes the camera then obtains information relating to the capabilities of the system.

NOTE: The Andor SDK takes several seconds to Auto-Calibrate the on-board A/D converter whenever the Initialize function is called.

2. The CCD sensor's operating temperature is set to some value within the allowed temperature range (e.g. -2 °C), and the cooler is switched on.

3 - 4. The current temperature is periodically monitored to check if the temperature has stabilized to the set value. The temperature can take several minutes to stabilize and with the appropriate programming techniques the user should be able to set up other tasks, as illustrated in the C examples.

Once the CCD sensor temperature has stabilized you can start acquiring data.



5. The acquisition parameters are programmed to match the specifications of the user, e.g. acquisition mode (single scan etc.), readout mode (full vertical binning etc.) and the trigger mode (Internal etc.).

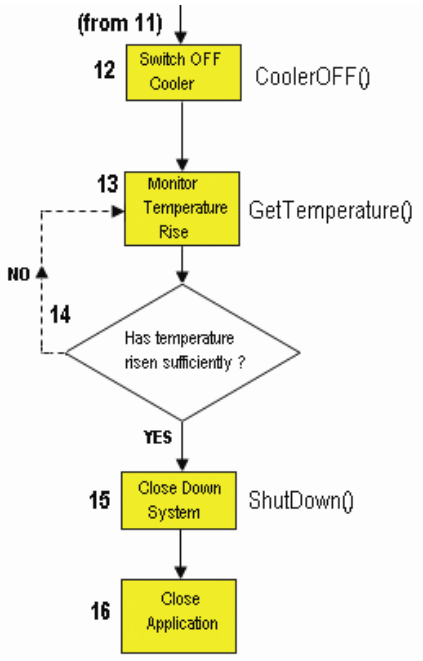
6. You are now ready to start an acquisition.

7 - 8. The current acquisition status is periodically monitored to check if the data acquisition is complete.

9. After a successful data acquisition the data is transferred from the Andor driver into the application.

10. At this point the user may choose to capture a new acquisition or not.

11. Yes: capture a new scan. The user may decide to alter the acquisition set-up (e.g. change the exposure time) or simply use the current parameters.



12. When the user has completely finished acquiring data the shutdown procedure is started. The cooler is switched off. It is important to control both the heating and cooling rates of the CCD sensor otherwise the temperature gradients may damage the sensor. Thus it is **highly recommended** that the user uses the correct exiting procedure rather than, for example, simply switching off the computer.

13 – 14. The current temperature is periodically monitored to check if the temperature has risen to a sufficiently high value.

15. For Classic & ICCD systems wait until the temperature has risen above **-20˚C**. The user may now shut down the Andor SDK system.

16. The program releases any memory still being used and exits the application.

Cooler

This example is different from all the previous examples in that its main goal is not to acquire data but to demonstrate the proper use of the cooling capabilities of the Andor SDK System. It includes the taking of a single FVB scan for completeness. This example is an expanded version of Example 1.

DDG™

The digital delay generator for iStar systems is demonstrated by this example. The user can control the gate times, gain level and integrate on chip parameters. The acquisition is set to a kinetic series of full vertically binned scans.

EMCCD

This example demonstrates acquisitions with an EMCCD detector, and in particular the Gain setting that can be applied to these devices

Events

The events example shows the alternative method of handling acquisitions, using Windows events to signal when the acquisition is complete instead of timer polling used in other examples. A kinetic series of full vertically binned scans is taken and the events signalled by the Andor SDK are indicated in the status window as they arrive

Frame Transfer

The frame transfer example is similar to the kinetics example, except that the accumulate cycle and kinetic series times can not be set independently, as they rely solely on the exposure time setting

FVB

This example illustrates the simplest mode of operation of the Andor system. It initializes the system and then acquires a single spectrum using the Full Vertical Binning readout mode. The user is given the ability to specify the trigger mode and exposure time (as the examples progress the user is given more and more options to set).

Image

This example is slightly more complicated than the first example with the addition of a shutter. In general a shutter must be used whenever the readout mode is anything other than Full Vertical Binning. For this example we will use the readout mode Image with the horizontal and vertical binning set to 1. The user is given the ability to specify the exposure time, trigger mode and some of the shutter details.

Image Binning

This example shows how to acquire single images with possible binning. The sub image to be read can be entered and the binning for each dimension can be set.

Kinetics/Accumulate

For this example we go back to the Full Vertical Binning readout mode as in example 1. However, we introduce a new acquisition mode, Kinetic Series. Kinetic Series is the most complex acquisition mode with up to 5 parameters to be set. The user is given the ability to specify the number of accumulations per scan, accumulation cycle time, number of scans in Kinetic series, Kinetic cycle time and the exposure time.

Kinetic Image

This example is a combination of the imaging and kinetic examples.

Multi-Track

This example illustrates the use of the Multi-Track readout mode. The acquisition mode is constrained to Single Scan and uses internal triggering. As this example uses imaging we again use a shutter. The user has the ability to specify both the shutter and Multi-Track parameters

Random-Track

This example is similar to **Multi-Track** readout mode as described above. The user has the ability to add/select their own track parameters, i.e. Start & Stop, number of tracks (Maximum of 20 tracks for iDus) and they can also select the shutter parameters.

Spool

This example demonstrates the use of spooling to disk. Spooling can be enabled or disabled and the stem of the created spool files can be entered. The acquisition mode is set to Kinetic Series

Continuous mode

This is a simple example to demonstrate the iCam functionality that some cameras may have.

SECTION 11 - FUNCTIONS

This section provides details of the various Functions available.

AbortAcquisition

unsigned int WINAPI AbortAcquisition(void)

Description	This function aborts the current acquisition if one is active.	
Parameters	NONE	
Return	unsigned int	
	DRV_SUCCESS	Acquisition aborted.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_IDLE	The system is not currently acquiring.
	DRV_VXDNOTINSTALLED	VxD not loaded.
	DRV_ERROR_ACK	Unable to communicate with card.
See also	GetStatus StartAcquisition	

CancelWait

unsigned int WINAPI CancelWait(void)

Description	This function restarts a thread which is sleeping within the WaitForAcquisition function. The sleeping thread will return from WaitForAcquisition with a value not equal to DRV_SUCCESS.	
Parameters	NONE	
Return	unsigned int	
	DRV_SUCCESS	Thread restarted successfully.
See also	WaitForAcquisition	

CoolerOFF

unsigned int WINAPI CoolerOFF(void)

Description Switches OFF the cooling. The rate of temperature change is controlled in some models until the temperature reaches 0º. Control is returned immediately to the calling application.

Parameters NONE

Return unsigned int

DRV_SUCCESS	Temperature controller switched OFF.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_NOT_SUPPORTED	Camera does not support switching cooler off.

See also CoolerON, SetTemperature, GetTemperature, GetTemperatureF, GetTemperatureRange, GetStatus

NOTE: Not available on Luca R cameras – always cooled to -20.

NOTE: (Classic & ICCD only)

1. **When the temperature control is switched off the temperature of the sensor is gradually raised to 0ºC to ensure no thermal stresses are set up in the sensor.**
2. **When closing down the program via ShutDown you must ensure that the temperature of the detector is above -20ºC, otherwise calling ShutDown while the detector is still cooled will cause the temperature to rise faster than certified.**

CoolerON

unsigned int WINAPI CoolerON(void)

Description Switches ON the cooling. On some systems the rate of temperature change is controlled until the temperature is within 3º of the set value. Control is returned immediately to the calling application.

Parameters NONE

Return unsigned int

DRV_SUCCESS	Temperature controller switched ON.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

See also CoolerOFF, SetTemperature, GetTemperature, GetTemperatureF, GetTemperatureRange, GetStatus

NOTE:

The temperature to which the detector will be cooled is set via SetTemperature. The temperature stabilization is controlled via hardware, and the current temperature can be obtained via GetTemperature. The temperature of the sensor is gradually brought to the desired temperature to ensure no thermal stresses are set up in the sensor.

Can be called for certain systems during an acquisition. This can be tested for using GetCapabilities.

DemosaicImage

unsigned int WINAPI DemosaicImage(WORD* grey, WORD* red, WORD* green, WORD* blue, ColorDemosaicInfo* info)

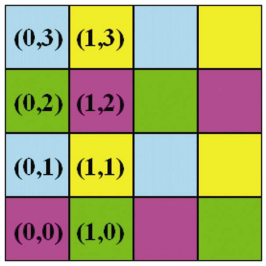
Description **For colour sensors only**

Demosaics an image taken with a CYMG CCD into RGB using the parameters stored in info. Below is the ColorDemosaicInfo structure definition and a description of its members:

```
typedef struct COLORDEMOSAICINFO {
 int iX;   // Number of X pixels. Must be >2.
 int iY;   // Number of Y pixels. Must be >2.
 int iAlgorithm;  // Algorithm to demosaic image.
 int iXPhase;  // First pixel in data (Cyan or Yellow/Magenta or Green).
 int iYPhase;  // First pixel in data (Cyan or Yellow/Magenta or Green).
 int iBackground;  // Background to remove from raw data when demosaicing.
} ColorDemosaicInfo;
```

- **iX** and **iY** are the image dimensions. The number of elements in the input red, green and blue arrays is **iX** x **iY**.
- **iAlgorithm** sets the algorithm to use: 0 for a 2x2 matrix demosaic algorithm or 1 for a 3x3 one.

The CYMG CCD pattern can be broken into cells of 2x4 pixels, e.g.:



- **iXPhase** and **iYPhase** store what colour is the bottom-left pixel.
- **iBackground** sets the numerical value to be removed from every pixel in the input image before demosaicing is done.

Parameters WORD* grey: pointer to image to demosaic

WORD* red: pointer to the red plane storage allocated by the user.

WORD* green: pointer to the green plane storage allocated by the user.

WORD* blue: pointer to the blue plane storage allocated by the user.

ColorDemosaicInfo* info: pointer to demosaic information structure.

Return unsigned int

DRV_SUCCESS	Image demosaiced
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Invalid pointer (i.e. NULL).
DRV_P3INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Invalid pointer (i.e. NULL).
DRV_P5INVALID	One or more parameters in info is out of range

See also GetMostRecentColorImage16, WhiteBalance

EnableKeepCleans

unsigned int WINAPI EnableKeepCleans (int mode)

Description	This function is only available on certain cameras operating in FVB external trigger mode. It determines if the camera keep clean cycle will run between acquisitions. When keep cleans are disabled in this way the exposure time is effectively the exposure time between triggers. The Keep Clean cycle is enabled by default. The feature capability AC_FEATURES_KEEPCLEANCONTROL determines if this function can be called for the camera.
Parameters	int mode: mode 0 OFF 1 ON
Return	unsigned int DRV_SUCCESS — Keep clean cycle mode set. DRV_NOT_INITIALIZED — System not initialized. DRV_NOT_AVAILABLE — Feature not available.
See also	GetCapabilities
NOTE:	**Currently only available on Newton and iKon cameras operating in FVB external trigger mode.**

FreeInternalMemory

unsigned int WINAPI FreeInternalMemory(void)

Description	The FreeInternalMemory function will deallocate any memory used internally to store the previously acquired data. Note that once this function has been called, data from last acquisition cannot be retrived.
Parameters	NONE
Return	unsigned int DRV_SUCCESS — Memory freed. DRV_NOT_INITIALIZED — System not initialized. DRV_ACQUIRING — Acquisition in progress. DRV_ERROR_ACK — Unable to communicate with card.
See also	GetImages, PrepareAcquisition

Filter_GetAveragingFactor

unsigned int WINAPI Filter_GetAveragingFactor (int * averagingFactor)

Description	Returns the current averaging factor value.
Parameters	int * averagingFactor: The current averaging factor value.
Return	unsigned int DRV_SUCCESS — Averaging factor returned.

	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_P1INVALID	Invalid averagingFactor (i.e. NULL pointer).

See also Filter_SetAveragingFactor

Filter_GetAveragingFrameCount

unsigned int WINAPI Filter_GetAveragingFrameCount (int * frames)

Description Returns the current frame count value.

Parameters int * frames: The current frame count value.

Return unsigned int

DRV_SUCCESS	Frame count returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid frame count (i.e. NULL pointer).

See also Filter_SetAveragingFrameCount

Filter_GetDataAveragingMode

unsigned int WINAPI Filter_GetDataAveragingMode (int * mode)

Description Returns the current averaging mode.

Parameters int * mode: The current averaging mode.

Return unsigned int

DRV_SUCCESS	Averaging mode returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid threshold (i.e. NULL pointer).

See also Filter_SetDataAveragingMode

Filter_GetMode

unsigned int WINAPI Filter_GetMode (unsigned int * mode)

Description Returns the current Noise Filter mode.

Parameters unsigned int * mode: Noise Filter mode.

Return unsigned int

DRV_SUCCESS	Filter mode returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Noise Filter processing not available for this camera.
DRV_P1INVALID	Invalid mode (i.e. NULL pointer)

See also Filter_SetMode

Filter_GetThreshold

unsigned int WINAPI Filter_GetThreshold (float * threshold)

Description Returns the current Noise Filter threshold value.

Parameters float * threshold: The current threshold value.

Return unsigned int

DRV_SUCCESS	Threshold returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Noise Filter processing not available for this camera.
DRV_P1INVALID	Invalid threshold (i.e. NULL pointer).

See also Filter_SetThreshold

Filter_SetAveragingFactor

unsigned int WINAPI Filter_SetAveragingFactor (int averagingFactor)

Description Sets the averaging factor.

Parameters int averagingFactor: The averaging factor to use.

Return unsigned int

DRV_SUCCESS	Averaging factor set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid averagingFactor.

See also Filter_GetAveragingFactor

Filter_SetAveragingFrameCount

unsigned int WINAPI Filter_SetAveragingFrameCount (int frames)

Description Sets the averaging frame count.

Parameters int frames: The averaging frame count to use.

Return unsigned int

DRV_SUCCESS	Averaging frame count set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid frame count.

See also Filter_GetAveragingFrameCount

Filter_SetDataAveragingMode

unsigned int WINAPI Filter_SetDataAveragingMode (int mode)

Description Sets the current data averaging mode.

Parameters int mode: The averaging factor mode to use.

Return unsigned int

DRV_SUCCESS	Averaging mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid mode.

See also Filter_GetDataAveragingMode

Filter_SetMode

unsigned int WINAPI Filter_SetMode (unsigned int mode)

Description Set the Noise Filter to use.

Parameters unsigned int mode: Filter mode to use.

Valid options are: 0 – No Filter
1 – Median Filter
2 – Level Above Filter
3 – Interquartile Range Filter
4 – Noise Threshold Filter

Return unsigned int

DRV_SUCCESS	Filter set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Noise Filter processing not available for this camera.
DRV_P1INVALID	Invalid mode.

See also Filter_GetMode

Filter_SetThreshold

unsigned int WINAPI Filter_SetThreshold (float threshold)

Description Sets the threshold value for the Noise Filter.

Parameters float threshold: Threshold value used to process image.

Valid values are: 0 – 65535 for Level Above filter.
0 – 10 for all other filters.

unsigned int

DRV_SUCCESS	Threshold set.

DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Noise Filter processing not available for this camera.
DRV_P1INVALID	Invalid threshold.

See also Filter_GetThreshold

GetAcquiredData

unsigned int WINAPI GetAcquiredData(at_32* arr, unsigned long size)

Description This function will return the data from the last acquisition. The data are returned as long integers (32-bit signed integers). The "array" must be large enough to hold the complete data set.

Parameters at_32* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Data copied.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	No acquisition has taken place

See also GetStatus, StartAcquisition, GetAcquiredData16

GetAcquiredData16

unsigned int WINAPI GetAcquiredData16(WORD* arr, unsigned long size)

Description 16-bit version of the **GetAcquiredData** function. The "array" must be large enough to hold the complete data set.

Parameters WORD* arr: pointer to data storage allocated by the user.
long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Data copied.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size isincorrect.
DRV_NO_NEW_DATA	No acquisition has taken place

See also GetStatus, StartAcquisition, GetAcquiredData

GetAcquiredFloatData

unsigned int WINAPI GetAcquiredFloatData (float* arr, unsigned long size)

Description **THIS FUNCTION IS RESERVED.**

GetAcquisitionProgress

unsigned int WINAPI GetAcquisitionProgress(long* acc, long* series)

Description This function will return information on the progress of the current acquisition. It can be called at any time but is best used in conjunction with SetDriverEvent.

The values returned show the number of completed scans in the current acquisition.
If 0 is returned for both accum and series then either:-

- No acquisition is currently running
- The acquisition has just completed
- The very first scan of an acquisition has just started and not yet completed

GetStatus can be used to confirm if the first scan has just started, returning DRV_ACQUIRING, otherwise it will return DRV_IDLE.

For example, if *accum*=2 and *series*=3 then the acquisition has completed 3 in the series and 2 accumulations in the 4 scan of the series.

Parameters long* acc: returns the number of accumulations completed in the current kinetic scan.
long* series: return the number of kinetic scans completed

Return unsigned int

DRV_SUCCESS	Number of accumulation and series scans completed.
DRV_NOT_INITIALIZED	System not initialized.

See also SetAcquisitionMode, SetNumberAccumulations, SetNumberKinetics, SetDriverEvent

GetAcquisitionTimings

unsigned int WINAPI GetAcquisitionTimings(float* exposure, float* accumulate, float* kinetic)

Description This function will return the current "valid" acquisition timing information. This function should be used after all the acquisitions settings have been set, e.g. SetExposureTime, SetKineticCycleTime and SetReadMode etc. The values returned are the actual times used in subsequent acquisitions.

This function is required as it is possible to set the exposure time to 20ms, accumulate cycle time to 30ms and then set the readout mode to full image. As it can take 250ms to read out an image it is not possible to have a cycle time of 30ms.

Parameters float* exposure: valid exposure time in seconds
float* accumulate: valid accumulate cycle time in seconds
float* kinetic: valid kinetic cycle time in seconds

Return unsigned int

DRV_SUCCESS	Timing information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_INVALID_MODE	Acquisition or readout mode is not available.

See also SetAccumulationCycleTime, SetAcquisitionMode, SetExposureTime, SetHSSpeed, SetKineticCycleTime, SetMultiTrack, SetNumberAccumulations, SetNumberKinetics, SetReadMode, SetSingleTrack, SetTriggerMode, SetVSSpeed

---

GetAdjustedRingExposureTimes

unsigned int WINAPI GetAdjustedRingExposureTimes (int inumTimes, float * fptimes)

Description This function will return the actual exposure times that the camera will use. There may be differences between requested exposures and the actual exposures.

Parameters int inumTimes: Numbers of times requested.
float * fptimes: Pointer to an array large enough to hold _inumTimes floats.

Return unsigned int

DRV_SUCCESS	Success.
DRV_NOT_INITIALIZED	System not initialized
DRV_P1INVALID	Invalid number of exposures requested

See also GetNumberRingExposureTimes, SetRingExposureTimes

GetAllDMAData

unsigned int WINAPI GetAllDMAData (at_32* arr, unsigned long size)

Description **THIS FUNCTION IS RESERVED.**

GetAmpDesc

unsigned int WINAPI GetAmpDesc (int index , char* name, int len)

Description This function will return a string with an amplifier description. The amplifier is selected using the index. The SDK has a string associated with each of its amplifiers. The maximum number of characters needed to store the amplifier descriptions is 21. The user has to specify the number of characters they wish to have returned to them from this function.

Parameters Int index: The amplifier index.

char* name: A user allocated array of characters for storage of the description.

int len: The length of the user allocated character array.

Return unsigned int

DRV_SUCCESS	Description returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	The amplifier index is not valid.
DRV_P2INVALID	The desc pointer is null.
DRV_P3INVALID	The len parameter is invalid (less than 1)

See also GetNumberAmp

GetAmpMaxSpeed

unsigned int WINAPI GetAmpMaxSpeed (int index , float* speed)

Description This function will return the maximum available horizontal shift speed for the amplifier selected by the index parameter.

Parameters Int index:amplifier index
float* speed:horizontal shift speed

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	The amplifier index is not valid

See also GetNumberAmp

GetAvailableCameras

unsigned int WINAPI GetAvailableCameras(long* totalCameras)

Description This function returns the total number of Andor cameras currently installed. It is possible to call this function before any of the cameras are initialized.

Parameters long* totalCameras: the number of cameras currently installed

Return unsigned int

DRV_SUCCESS	Number of available cameras returned.
DRV_GENERAL_ERRORS	An error occurred while obtaining the number of available cameras.

See also SetCurrentCamera, GetCurrentCamera, GetCameraHandle

GetBackground

unsigned int WINAPI GetBackground (at_32* arr, unsigned long size)

Description **THIS FUNCTION IS RESERVED.**

GetBaselineClamp

unsigned int WINAPI GetBaselineClamp(int* state)

Description This function returns the status of the baseline clamp functionality. With this feature enabled the baseline level of each scan in a kinetic series will be more consistent across the sequence.

Parameters int * state: Baseline clamp functionality Enabled/Disabled

1 – Baseline Clamp Enabled

0 – Baseline Clamp Disabled

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_SUPPORTED	Baseline Clamp not supported on this camera
DRV_P1INVALID	State parameter was not zero or one.

See also SetBaselineClamp, SetBaselineOffset

GetBitDepth

unsigned int WINAPI GetBitDepth(int channel, int* depth)

Description This function will retrieve the size in bits of the dynamic range for any available AD channel.

Parameters int channel: the AD channel.
int* depth: dynamic range in bits

Return unsigned int

DRV_SUCCESS	Depth returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Invalid channel

See also GetNumberADChannels, SetADChannel

GetCameraEventStatus

unsigned int WINAPI GetCameraEventStatus (DWORD * camStatus)

Description This function will return if the system is exposing or not.

Parameters DWORD * camStatus: The status of the firepulse will be returned that the firepulse is low

0 Fire pulse low

1 Fire pulse high

Return unsigned int

DRV_SUCCESS	Status returned
DRV_NOT_INITIALIZED	System not initialized

See also SetAcqStatusEvent, SetPCIMode

NOTE **This is only supported by the CCI23 card.**

GetCameraHandle

unsigned int WINAPI GetCameraHandle(long cameraIndex, long* cameraHandle)

Description This function returns the handle for the camera specified by cameraIndex. When multiple Andor cameras are installed the handle of each camera must be retrieved in order to select a camera using the SetCurrentCamera function.

The number of cameras can be obtained using the GetAvailableCameras function.

Parameters long cameraIndex: index of any of the installed cameras.

Valid values 0 to NumberCameras-1 where NumberCameras is the value returned by the GetAvailableCameras function.

long* cameraHandle: handle of the camera.

Return unsigned int

DRV_SUCCESS	Camera handle returned.
DRV_P1INVALID	Invalid camera index.

See also SetCurrentCamera, GetAvailableCameras, GetCurrentCamera

---

GetCameraInformation

unsigned int WINAPI GetCameraInformation (int index, long * information)

Description This function will return information on a particular camera denoted by the index.

Parameters Int index: (reserved)

Long* information: current state of camera

Bit:1 1 - USB camera present

Bit:2 1 - All dlls loaded properly

Bit:3 1 - Camera Initialized correctly

Return unsigned int

DRV_SUCCESS	Driver status return
DRV_VXDNOTINSTALLED	Driver not installed
DRV_USBERROR	USB device error

See also GetCameraHandle, GetHeadModel, GetCameraSerialNumber, GetCapabilities

NOTE **Only available in iDus. The index parameter is not used at present so should be set to 0. For any camera except the iDus The value of information following a call to this function will be zero.**

GetCameraSerialNumber

unsigned int WINAPI GetCameraSerialNumber (int* number)

Description This function will retrieve camera's serial number.

Parameters int *number: Serial Number.

Return unsigned int

DRV_SUCCESS	Serial Number returned.
DRV_NOT_INITIALIZED	System not initialized.

See also GetCameraHandle, GetHeadModel, GetCameraInformation, GetCapabilities

GetCapabilities

unsigned int WINAPI GetCapabilities(AndorCapabilities* caps)

Description This function will fill in an AndorCapabilities structure with the capabilities associated with the connected camera. Before passing the address of an AndorCapabilites structure to the function the ulSize member of the structure should be set to the size of the structure. In C++ this can be done with the line:

caps->ulSize = sizeof(AndorCapabilities);

Individual capabilities are determined by examining certain bits and combinations of bits in the member variables of the AndorCapabilites structure. The next few pages contain a summary of the capabilities currently returned.

Parameters Andor capabilities* caps: the capabilities structure to be filled in.

Return unsigned int

DRV_NOT_INITIALIZED	System not initialized
DRV_SUCCESS	Capabilities returned.
DRV_P1INVALID	Invalid caps parameter (i.e. NULL).

See also GetCameraHandle, GetCameraSerialNumber, GetHeadModel, GetCameraInformation

Acquisition Modes - AndorCapabilities Member: ulAcqModes

Capability: AC_ACQMODE_SINGLE
Description: Single Scan Acquisition Mode available using SetAcquisitionMode.
Bit: 0
State: 1

Capability: AC_ACQMODE_VIDEO
Description: Video (Run Till Abort) Acquisition Mode available using SetAcquisitionMode.
Bit: 1
State: 1

Capability: AC_ACQMODE_ACCUMULATE
Description: Accumulation Acquisition Mode available using SetAcquisitionMode.
Bit: 2
State: 1

Capability: AC_ACQMODE_KINETIC
Description: Kinetic Series Acquisition Mode available using SetAcquisitionMode.
Bit: 3
State: 1

Capability: AC_ACQMODE_FRAMETRANSFER
Description: Frame Transfer Acquisition Mode available using SetAcquisitionMode.
Bit: 4
State: 1

Capability: AC_ACQMODE_FASTKINETICS
Description: Fast Kinetics Acquisition Mode available using SetAcquisitionMode.
Bit: 5
State: 1

Capability: AC_ACQMODE_OVERLAP
Description: Overlap Acquisition Mode available using SetAcquisitionMode.
Bit: 6
State: 1

Read Modes - AndorCapabilities Member: ulReadModes

Capability: AC_READMODE_FULLIMAGE
Description: Full Image Read Mode available using SetReadMode.
Bit: 0
State: 1

Capability: AC_READMODE_SUBIMAGE
Description: Sub Image Read Mode available using SetReadMode.
Bit: 1
State: 1

Capability: AC_READMODE_SINGLETRACK
Description: Single track Read Mode available using SetReadMode.
Bit: 2
State: 1

Capability: AC_READMODE_FVB
Description: Full Vertical Binning Read Mode available using SetReadMode.
Bit: 3
State: 1

Capability: AC_READMODE_MULTITRACK
Description: Multi Track Read Mode available using SetReadMode.
Bit: 4
State: 1

Capability: AC_READMODE_RANDOMTRACK
Description: Random-Track Read Mode available using SetReadMode.
Bit: 5
State: 1

Read Modes - AndorCapabilities Member: ulFTReadModes

Capability: AC_READMODE_FULLIMAGE
Description: Full Image Read Mode available using SetReadMode.
Bit: 0
State: 1

Capability: AC_READMODE_SUBIMAGE
Description: Sub Image Read Mode available using SetReadMode.
Bit: 1
State: 1

Capability: AC_READMODE_SINGLETRACK
Description: Single track Read Mode available using SetReadMode.
Bit: 2
State: 1

Capability: AC_READMODE_FVB
Description: Full Vertical Binning Read Mode available using SetReadMode.
Bit: 3
State: 1

Capability: AC_READMODE_MULTITRACK
Description: Multi Track Read Mode available using SetReadMode.
Bit: 4
State: 1

Capability: AC_READMODE_RANDOMTRACK
Description: Random-Track Read Mode available using SetReadMode.
Bit: 5
State: 1

Trigger Modes - AndorCapabilities Member: ulTriggerModes

Capability: AC_TRIGGERMODE_INTERNAL
Description: Internal Trigger Mode available using SetTriggerMode.
Bit: 0
State: 1

Capability: AC_TRIGGERMODE_EXTERNAL
Description: External Trigger Mode available using SetTriggerMode.
Bit: 1
State: 1

Capability: AC_TRIGGERMODE_EXTERNAL_FVB_EM
Description: External FVB EM Trigger Mode available using SetTriggerMode.
Bit: 2
State: 1

Capability: AC_TRIGGERMODE_CONTINUOUS
Description: Continuous Trigger Mode available using SetTriggerMode.
Bit: 3
State: 1

Capability: AC_TRIGGERMODE_EXTERNALSTART
Description: External Start Trigger Mode available using SetTriggerMode.
Bit: 4
State: 1

Capability: AC_TRIGGERMODE_BULB
Description: Bulb Trigger Mode available using SetTriggerMode.
Bit: 5
State: 1
Note: This capability is deprecated by AC_TRIGGERMODE_EXTERNALEXPOSURE.

Capability: AC_TRIGGERMODE_EXTERNALEXPOSURE
Description: External Exposure Trigger Mode available using SetTriggerMode.
Bit: 5
State: 1

Capability: AC_TRIGGERMODE_INVERTED
Description: Inverted Trigger Mode available using SetTriggerInvert.
Bit: 6
State: 1

Camera Type - AndorCapabilities Member: ulCameraType

Capability: AC_CAMERATYPE_PDA
Description: Camera is an Andor PDA.
Bits: 0-31
Value: 0

Capability: AC_CAMERATYPE_IXON
Description: Camera is an Andor iXon.
Bits: 0-31
Value: 1

Capability: AC_CAMERATYPE_ICCD
Description: Camera is an Andor ICCD.
Bits: 0-31
Value: 2

Capability: AC_CAMERATYPE_EMCCD
Description: Camera is an Andor EMCCD.
Bits: 0-31
Value: 3

Capability: AC_CAMERATYPE_CCD
Description: Camera is an Andor CCD.
Bits: 0-31
Value: 4

Capability: AC_CAMERATYPE_ISTAR
Description: Camera is an Andor iStar.
Bits: 0-31
Value: 5

Capability: AC_CAMERATYPE_VIDEO
Description: Camera is a third party camera.
Bits: 0-31
Value: 6

GetCapabilities (Camera Type) - continued

Capability: AC_CAMERATYPE_IDUS
Description: Camera is an Andor iDus.
Bits: 0-31
Value: 7

Capability: AC_CAMERATYPE_NEWTON
Description: Camera is an Andor Newton.
Bits: 0-31
Value: 8

Capability: AC_CAMERATYPE_SURCAM
Description: Camera is an Andor Surcam.
Bits: 0-31
Value: 9

Capability: AC_CAMERATYPE_USBISTAR
Description: Camera is an Andor USBiStar.
Bits: 0-31
Value: 10

Capability: AC_CAMERATYPE_LUCA
Description: Camera is an Andor Luca.
Bits: 0-31
Value: 11

Capability: AC_CAMERATYPE_RESERVED
Description: Reserved.
Bits: 0-31
Value: 12

Capability: AC_CAMERATYPE_IKON
Description: Camera is an Andor iKon.
Bits: 0-31
Value: 13

Capability: AC_CAMERATYPE_INGAAS
Description: Camera is an Andor InGaAs.
Bits: 0-31
Value: 14

Capability: AC_CAMERATYPE_IVAC
Description: Camera is an Andor iVac.
Bits: 0-31

Value: 15

Capability: AC_CAMERATYPE_CLARA
Description: Camera is an Andor Clara.
Bits: 0-31
Value: 17

All other values reserved.

Pixel Mode - AndorCapabilities Member: ulPixelModes

Capability: AC_PIXELMODE_8BIT
Description: Camera can acquire in 8-bit mode.
Bit: 0
State: 1

Capability: AC_PIXELMODE_14BIT
Description: Camera can acquire in 14-bit mode.
Bit: 1
State: 1

Capability: AC_PIXELMODE_16BIT
Description: Camera can acquire in 16-bit mode.
Bit: 2
State: 1

Capability: AC_PIXELMODE_32BIT
Description: Camera can acquire in 32-bit mode.
Bit: 3
State: 1

Capability: AC_PIXELMODE_MONO
Description: Camera acquires data in grey scale.
Bits: 16-31
Value: 0

Capability: AC_PIXELMODE_RGB
Description: Camera acquires data in RGB mode.
Bits: 16-31
Value: 1

Capability: AC_PIXELMODE_CMY
Description: Camera acquires data in CMY mode.
Bits: 16-31
Value: 2

Available Set Functions - AndorCapabilities Member: ulSetFunctions

Capability: AC_SETFUNCTION_VREADOUT
Description: The vertical readout speed can be set with the SetVSSpeed function.
Bit: 0
State: 1

Capability: AC_SETFUNCTION_HREADOUT
Description: The horizontal readout speed can be set with the SetHSSpeed function.
Bit: 1
State: 1

Capability: AC_SETFUNCTION_TEMPERATURE
Description: The target temperature can be set using the SetTemperature function.
Bit: 2
State: 1

Capability: AC_SETFUNCTION_MCPGAIN (AC_SETFUNCTION_GAIN Deprecated)
Description: Gain through the SetMCPGain function is available.
Bit: 3
State: 1

Capability: AC_SETFUNCTION_EMCCDGAIN
Description: Gain through the SetEMCCDGain function is available.
Bit: 4
State: 1

Capability: AC_SETFUNCTION_BASELINECLAMP
Description: Baseline clamp can be turned on or off with the SetBaselineClamp function.
Bit: 5
State: 1

Capability: AC_SETFUNCTION_VSAMPLITUDE
Description: The vertical clock voltage can be set with the SetVSAmplitude function.
Bit: 6
State: 1

Capability: AC_SETFUNCTION_HIGHCAPACITY
Description: High capacity mode can be turned on or off with the SetHighCapacity function.
Bit: 7
State: 1

GetCapabilities (Available Set Functions) - Continued

Capability: AC_SETFUNCTION_BASELINEOFFSET
Description: The baseline offset can be set with the SetBaselineOffset function.
Bit: 8
State: 1

Capability: AC_SETFUNCTION_PREAMPGAIN
Description: The pre amp gain can be set with the SetPreAmpGain function.
Bit: 9
State: 1

Capability: AC_SETFUNCTION_CROPMODE
Description: Crop mode can be selected using the SetCropMode or SetIsolatedCropMode functions.
Bit: 10
State: 1

Capability: AC_SETFUNCTION_DMAPARAMETERS
Description: The DMA parameters can be set with the SetDMAParameters function.
Bit: 11
State: 1

Capability: AC_SETFUNCTION_HORIZONTALBIN
Description: The horizontal binning can be set for the relative read mode.
Bit: 12
State: 1 **See Note.**

Capability: AC_SETFUNCTION_MULTITRACKHRANGE
Description: The multitrack horizontal range can be set using the SetMultiTrackHRange function.
Bit: 13
State: 1

Capability: AC_SETFUNCTION_RANDOMTRACKNOGAPS
Description: Random tracks can be set with no gaps inbetween with the SetRandomTracks or SetComplexImage functions.
Bit: 14
State: 1
NOTE: For iDus, the horizontalbin capability will be 0, as it is not recommended, but it is possible.

Capability: AC_SETFUNCTION_EMADVANCED
Description: Extended EM gain range can be accessed using SetEMAdvanced.
Bit: 15
State: 1

Available Get Functions - AndorCapabilities Member: ulGetFunctions

Capability: AC_GETFUNCTION_TEMPERATURE
Description: The current temperature can be determined using the GetTemperature function.
Bit: 0
State: 1

Capability: AC_GETFUNCTION_TEMPERATURERANGE
Description: The range of possible temperatures can be determined using the GetTemperatureRange function.
Bit: 2
State: 1

Capability: AC_GETFUNCTION_DETECTORSIZE
Description: The dimensions of the detector can be determined using the GetDetector function.
Bit: 3
State: 1

Capability: AC_GETFUNCTION_MCPGAIN (AC_GETFUNCTION_GAIN deprecated)
Description: Reserved capability.
Bit: 4
State: 1

Capability: AC_GETFUNCTION_EMCCDGAIN
Description: The gain can be determined using the GetEMCCDGain function.
Bit: 5
State: 1

Capability: AC_GETFUNCTION_BASELINECLAMP
Description: The gain can be determined using the GetBaselineClamp function.
Bit: 15
State: 1

SDK Features Available - AndorCapabilities Member: ulFeatures

Capability: AC_FEATURES_POLLING
Description: The status of the current acquisition can be determined through the GetStatus function call.
Bit: 0
State: 1

Capability: AC_FEATURES_EVENTS
Description: A Windows Event can be passed to the SDK to alert the user at certain stages of the Acquisition. See SetDriverEvent
Bit: 1
State: 1

Capability: AC_FEATURES_SPOOLING
Description: Acquisition Data can be made to spool to disk using the SetSpool function.
Bit: 2
State: 1

Capability: AC_FEATURES_SHUTTER
Description: Shutter settings can be adjusted through the SetShutter function.
Bit: 3
State: 1

Capability: AC_FEATURES_SHUTTEREX
Description: Shutter settings can be adjusted through the SetShutterEx function.
Bit: 4
State: 1

Capability: AC_FEATURES_EXTERNAL_I2C
Description: The camera has its own dedicated external I2C bus.
Bit: 5
State: 1

Capability: AC_FEATURES_SATURATIONEVENT
Description: Sensor saturation can be determined through the SetSaturationEvent function.
Bit: 6
State: 1

GetCapabilities (SDK Features Available) - Continued

Capability: AC_FEATURES_FANCONTROL
Description: Fan settings can be adjusted through the SetFanMode function.
Bit: 7
State: 1

Capability: AC_FEATURES_MIDFANCONTROL
Description: It is possible to select a low fan setting through the SetFanMode function.
Bit: 8
State: 1

Capability: AC_FEATURES_TEMPERATUREDURINGACQUISITION
Description: It is possible to read the camera temperature during an acquisition with the GetTemperature function.

Bit: 9
State: 1

Capability: AC_FEATURES_KEEPCLEANCONTROL
Description: It is possible to turn off keep cleans between scans.

Bit: 10
State: 1

Capability: AC_FEATURES_DDGLITE
Description: Reserved for internal use.

Bit: 11
State: 1

Capability: AC_FEATURES_FTEXTERNALEXPOSURE
Description: The combination of Frame Transfer and External Exposure modes is available.

Bit: 12
State: 1

Capability: AC_FEATURES_KINETICEXTERNALEXPOSURE
Description: External Exposure trigger mode is available in Kinetic acquisition mode.

Bit: 13
State: 1

Capability: AC_FEATURES_DACCONTROL
Description: Reserved for internal use.

Bit: 14
State: 1

Capability: AC_FEATURES_METADATA
Description: Reserved for internal use.

Bit: 15
State: 1

Capability: AC_FEATURES_IOCONTROL
Description: Configurable IO's available. See SetIOLevel.

Bit: 16
State: 1

Capability: AC_FEATURES_PHOTONCOUNTING
Description: System supports photon counting. See SetPhotonCounting .

Bit: 17
State: 1

Capability: AC_FEATURES_COUNTCONVERT
Description: System supports Count Convert.

Bit: 18
State: 1

Capability: AC_FEATURES_DUALMODE
Description: Dual exposure mode. See SetDualExposureMode.

Bit: 19
State: 1

PCI Card Capabilities - AndorCapabilities Member: ulPCICard

Description: Maximum speed in Hz PCI controller card is capable of.

Gain Features Available - AndorCapabilities Member: ulEMGainCapability

Capability: AC_EMGAIN_8BIT
Description:.8-bit DAC settable.
Bit: 0
State: 1

Capability: AC_EMGAIN_12BIT
Description:.12-bit DAC settable
Bit: 1
State: 1

Capability: AC_EMGAIN_LINEAR12
Description:.Gain setting represent a linear gain scale. 12-bit DAC used internally.
Bit: 2
State: 1

Capability: AC_EMGAIN_REAL12
Description:.Gain setting represents the real EM Gain value. 12-bit DAC used internally.
Bit: 3
State: 1

GetControllerCardModel

unsigned int WINAPI GetControllerCardModel (char* controllerCardModel)

Description This function will retrieve the type of PCI controller card included in your system. This function is not applicable for USB systems. The maximum number of characters that can be returned from this function is 10.

Parameters char* controllerCardModel: A user allocated array of characters for storage of the controller card model.

Return unsigned int

DRV_SUCCESS	Name returned.
DRV_NOT_INITIALIZED	System not initialized

See also GetHeadModel, GetCameraSerialNumber, GetCameraInformation, GetCapabilities

---

GetCountConvertWavelengthRange

unsigned int WINAPI GetCountConvertWavelengthRange(float* min_wave, float* max_wave)

Description This function returns the valid wavelength range available in Count Convert mode.

Parameters float* min_wave: minimum wavelength permited.
float* max_wave: maximum wavelength permited.

Return unsigned int

DRV_SUCCESS	Count Convert wavelength set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Count Convert not supported on this camera

See also GetCapabilities, SetCountConvertMode, SetCountConvertWavelength

---

GetCurrentCamera

unsigned int WINAPI GetCurrentCamera(long* cameraHandle)

Description When multiple Andor cameras are installed this function returns the handle of the currently selected one.

Parameters long* cameraHandle: handle of the currently selected camera

Return unsigned int

DRV_SUCCESS	Camera handle returned.

See also SetCurrentCamera, GetAvailableCameras, GetCameraHandle

---

GetDDGPulse

unsigned int WINAPI GetDDGPulse(double width, double resolution, double* Delay, double* Width)

Description This function attempts to find a laser pulse in a user-defined region with a given resolution. The values returned will provide an estimation of the location of the pulse.

Parameters double width: the time in picoseconds of the region to be searched.
double resolution: the minimum gate pulse used to locate the laser.
double* Delay: the approximate start of the laser pulse.
double* Width: the pulse width, which encapsulated the laser pulse.

Return unsigned int

DRV_SUCCESS	Location returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

NOTE: Available in iStar.

GetDDGIOCFrequency

unsigned int WINAPI GetDDGIOCFrequency(double*frequency)

Description This function can be used to return the actual IOC frequency that will be triggered. It should only be called once all the conditions of the experiment have been defined.

Parameters double*frequency: the number of integrate on chip pulses triggered within the fire pulse.

Return unsigned int

DRV_SUCCESS	Number returned
DRV_NOT_INITIALIZED	System not initialized
DRV_ERROR_ACK	Unable to communicate with card

See also SetDDGIOCFrequency SetDDGIOCNumber GetDDGIOCNumber GetDDGIOCPulses SetDDGIOC

NOTE: Available in iStar.

GetDDGIOCNumber

unsigned int WINAPI GetDDGIOCNumber(unsigned long* numberPulses)

Description This function can be used to return the actual number of pulses that will be triggered. It should only be called once all the conditions of the experiment have been defined.

Parameters unsigned long* numberPulses: the number of integrate on chip pulses triggered within the fire pulse.

Return unsigned int

DRV_SUCCESS	Number returned
DRV_NOT_INITIALIZED	System not initialized
DRV_ERROR_ACK	Unable to communicate with card

See also SetDDGIOCFrequency GetDDGIOCFrequency SetDDGIOCNumber GetDDGIOCPulses SetDDGIOC

NOTE: Available in iStar.

GetDDGIOCPulses

unsigned int WINAPI GetDDGIOCPulses(int* pulses)

Description	This function can be used to calculate the number of pulses that will be triggered with the given exposure time, readout mode, acquisition mode and integrate on chip frequency. It should only be called once all the conditions of the experiment have been defined.
Parameters	int* pulses: the number of integrate on chip pulses triggered within the fire pulse.
Return	unsigned int

DRV_SUCCESS	Number returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

See also SetDDGIOCFrequency GetDDGIOCFrequency SetDDGIOCNumber GetDDGIOCNumber SetDDGIOC

NOTE: Available in iStar.

GetDetector

unsigned int WINAPI GetDetector(int* xpixels, int* ypixels)

Description This function returns the size of the detector in pixels. The horizontal axis is taken to be the axis parallel to the readout register.

Parameters int* xpixels: number of horizontal pixels.
int* ypixels: number of vertical pixels.

Return unsigned int

DRV_SUCCESS	Detector size returned.
DRV_NOT_INITIALIZED	System not initialized.

GetDICameraInfo

unsigned int WINAPI GetDICameraInfo (void *info)

Description **THIS FUNCTION IS RESERVED.**

GetDualExposureTimes

unsigned int WINAPI GetDualExposureTimes(float* exposure1, float* exposure2)

Description This function will return the current “valid” acquisition timing information for dual exposure mode. This mode is only available for certain sensors in run till abort mode, external trigger, full image.

Parameters float* exposure1: valid exposure time in seconds for each odd numbered frame.
float* exposure2: valid exposure time in seconds for each even numbered frame.

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized. .
DRV_NOT_SUPPORTED	Dual exposure mode not supported on this camera.
DRV_NOT_AVAILABLE	Dual exposure mode not configured correctly.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	exposure1 has invalid memory address.
DRV_P2INVALID	exposure2 has invalid memory address.

See also GetCapabilities, SetDualExposureMode, SetDualExposureTimes

---

GetEMCCDGain

unsigned int WINAPI GetEMCCDGain(int* gain)

Description Returns the current gain setting. The meaning of the value returned depends on the EM Gain mode.

Parameters Int*gain: current EM gain setting

Return

DRV_SUCCESS	Gain returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.

GetEMGainRange

unsigned int WINAPI GetEMGainRange(int* low, int* high)

Description Returns the minimum and maximum values of the current selected EM Gain mode and temperature of the sensor.

Parameters int* low: lowest gain setting

int* high: highest gain setting

Return

DRV_SUCCESS	Gain range returned.
DRV_NOT_INITIALIZED	System not initialized.

GetFastestRecommendedVSSpeed

unsigned int WINAPI GetFastestRecommendedVSSpeed (int* index, float* speed)

Description As your Andor SDK system may be capable of operating at more than one vertical shift speed this function will return the fastest recommended speed available. The very high readout speeds, may require an increase in the amplitude of the Vertical Clock Voltage using SetVSAmplitude. This function returns the fastest speed which does not require the Vertical Clock Voltage to be adjusted. The values returned are the vertical shift speed index and the actual speed in microseconds per pixel shift.

Parameters Int* index: index of the fastest recommended vertical shift speed
float* speed: speed in microseconds per pixel shift.

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetVSSpeed, GetNumberVSSpeeds, SetVSSpeed

GetFIFOUsage

unsigned int WINAPI GetFIFOUsage (int* FIFOusage)

Description **THIS FUNCTION IS RESERVED.**

GetFilterMode

unsigned int WINAPI GetFilterMode(int* mode)

Description This function returns the current state of the cosmic ray filtering mode.

Parameters int* mode: current state of filter

0	OFF
2	ON

Return unsigned int

DRV_SUCCESS	Filter mode returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also SetFilterMode

GetFKExposureTime

unsigned int WINAPI GetFKExposureTime(float* time)

Description This function will return the current "valid" exposure time for a fast kinetics acquisition. This function should be used after all the acquisitions settings have been set, i.e. SetFastKinetics and SetFKVShiftSpeed. The value returned is the actual time used in subsequent acquisitions.

Parameters float* time: valid exposure time in seconds

Return unsigned int

DRV_SUCCESS	Timing information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_INVALID_MODE	Fast kinetics is not available.

See also SetFastKinetics, SetFKVShiftSpeed

GetFKVShiftSpeed

unsigned int WINAPI GetFKVShiftSpeed(int index, int* speed)

Description

Deprecated see Note:

As your Andor SDK system is capable of operating at more than one fast kinetics vertical shift speed this function will return the actual speeds available. The value returned is in microseconds per pixel shift.

Parameters int index: speed required

Valid values 0 to GetNumberFKVShiftSpeeds()-1

int* speed: speed in micro-seconds per pixel shift

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also GetNumberFKVShiftSpeeds, SetFKVShiftSpeed

NOTE: Deprecated by GetFKVShiftSpeedF

GetFKVShiftSpeedF

unsigned int WINAPI GetFKVShiftSpeedF(int index, float* speed)

Description As your Andor system is capable of operating at more than one fast kinetics vertical shift speed this function will return the actual speeds available. The value returned is in microseconds per pixel shift.

Parameters int index: speed required

Valid values: 0 to GetNumberFKVShiftSpeeds()-1

float* speed: speed in micro-seconds per pixel shift

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also GetNumberFKVShiftSpeeds, SetFKVShiftSpeed

NOTE: Only available if camera is Classic or iStar.

GetHardwareVersion

unsigned int WINAPI GetHardwareVersion(unsigned int* PCB, unsigned int* Decode, unsigned int* dummy1, unsigned int* dummy2, unsigned int* CameraFirmwareVersion, unsigned int* CameraFirmwareBuild)

Description This function returns the Hardware version information.

Parameters Unsigned int* PCB: Plug-in card version
unsigned int* Decode: Flex 10K file version
unsigned int* dummy1
unsigned int* dummy2
unsigned int* CameraFirmwareVersion: Version number of camera firmware
unsigned int* CameraFirmwareBuild: Build number of camera firmware

Return unsigned int

DRV_SUCCESS	Version information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

GetHeadModel

unsigned int WINAPI GetHeadModel(char* name)

Description This function will retrieve the type of CCD attached to your system.

Parameters char* name: A user allocated array of characters for storage of the Head Model. This should be declared as size MAX_PATH.

Return unsigned int

DRV_SUCCESS	Name returned.
DRV_NOT_INITIALIZED	System not initialized.

GetHorizontalSpeed

unsigned int WINAPI GetHorizontalSpeed(int index, int* speed)

Description

Deprecated see Note:

As your Andor system is capable of operating at more than one horizontal shift speed this function will return the actual speeds available. The value returned is in microseconds per pixel shift.

Parameters

int index: speed required

Valid values: 0 to **NumberSpeeds**-1, where **NumberSpeeds** is the parameter returned by GetNumberHorizontalSpeeds.

int* speed: speed in micro-seconds per pixel shift

Return

unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also

GetNumberHorizontalSpeeds, SetHorizontalSpeed

NOTE: Deprecated by GetHSSpeed

GetHSSpeed

unsigned int WINAPI GetHSSpeed(int channel, int typ, int index, float* speed)

Description As your Andor system is capable of operating at more than one horizontal shift speed this function will return the actual speeds available. The value returned is in MHz.

Parameters int channel: the AD channel.
int typ: output amplification.
Valid values: 0 electron multiplication/Conventional(clara).
1 conventional/Extended NIR Mode(clara).
int index: speed required
Valid values 0 to NumberSpeeds-1 where NumberSpeeds is value returned in first parameter after a call to GetNumberHSSpeeds().
float* speed: speed in in MHz.

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Invalid channel.
DRV_P2INVALID	Invalid horizontal read mode
DRV_P3INVALID	Invalid index

See also GetNumberHSSpeeds, SetHSSpeed

NOTE: The speed is returned in microseconds per pixel shift for iStar and Classic systems.

GetHVflag

unsigned int WINAPI GetHVflag (int* bFlag)

Description This function will retrieve the High Voltage flag from your USB iStar intensifier. A 0 value indicates that the high voltage is abnormal.

Parameters int* bFlag: pointer to High Voltage flag.

Return unsigned int

DRV_SUCCESS	HV flag returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Not a USB iStar.

NOTE **Available only on USB iStar.**

GetID

unsigned int WINAPI GetID (int devNum, int* id)

Description **THIS FUNCTION IS RESERVED.**

GetImageFlip

unsigned int WINAPI GetImageFlip(int* iHFlip, int* iVFlip)

Description This function will obtain whether the acquired data output is flipped in either the horizontal or vertical direction.

Parameters int* iHFlip: Gets horizontal flipping.
int* iVFlip: Gets vertical flipping.

1 – Flipping Enabled

0 – Flipping Disabled

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	HFlip parameter invalid.
DRV_P2INVALID	VFlip parameter invalid

See also SetImageRotate SetImageFlip

GetImageRotate

unsigned int WINAPI GetImageRotate(int* iRotate)

Description This function will obtain whether the acquired data output is rotated in any direction.

Parameters int* iRotate: Rotation setting

0 - No rotation
1 - Rotate 90 degrees clockwise
2 - Rotate 90 degrees anti-clockwise

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Rotate parameter invalid.

See also SetImageFlip SetImageRotate

GetImages

unsigned int WINAPI GetImages(long first, long last, at_32* arr, unsigned long size, long* validfirst, long* validlast)

Description This function will update the data array with the specified series of images from the circular buffer. If the specified series is out of range (i.e. the images have been overwritten or have not yet been acquired then an error will be returned.

Parameters
long first: index of first image in buffer to retrieve.
long last: index of last image in buffer to retrieve.
at_32* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.
long* validfirst: index of the first valid image.
long* validlast: index of the last valid image.

Return unsigned int

DRV_SUCCESS	Images have been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_GENERAL_ERRORS	The series is out of range.
DRV_P3INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetImages16, GetNumberNewImages

GetImages16

unsigned int WINAPI GetImages16(long first, long last, WORD* arr, unsigned long size, long* validfirst, long* validlast)

Description 16-bit version of the GetImages function.

Parameters
long first: index of first image in buffer to retrieve.
long last: index of last image in buffer to retrieve.
WORD* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.
long* validfirst: index of the first valid image.
long* validlast: index of the last valid image.

Return unsigned int

DRV_SUCCESS	Images have been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_GENERAL_ERRORS	The series is out of range.
DRV_P3INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetImages, GetNumberNewImages

GetImagesPerDMA

unsigned int WINAPI GetImagesPerDMA (unsigned long* images)

Description This function will return the maximum number of images that can be transferred during a single DMA transaction.

Parameters unsigned long* images:
Return unsigned int
DRV_SUCCESS

GetIRQ

unsigned int WINAPI GetIRQ (int* IRQ)

Description **THIS FUNCTION IS RESERVED.**

GetKeepCleanTime

unsigned int WINAPI GetKeepCleanTime(float* KeepCleanTime)

Description This function will return the time to perform a keep clean cycle. This function should be used after all the acquisitions settings have been set, e.g. SetExposureTime, SetKineticCycleTime and SetReadMode etc. The value returned is the actual times used in subsequent acquisitions.

Parameters float* KeepCleanTime: valid readout time in seconds

Return unsigned int

DRV_SUCCESS	Timing information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_CODES	Error communicating with camera.

See also GetAcquisitionTimings GetReadOutTime

NOTES **NOTE: Available on iDus, iXon, Luca & Newton.**

GetMaximumBinning

unsigned int WINAPI GetMaximumBinning (int ReadMode, int HorzVert, int* MaxBinning)

Description This function will return the maximum binning allowable in either the vertical or horizontal dimension for a particular readout mode.

Parameters int ReadMode: The readout mode for which to retrieve the maximum binning (see SetReadMode for possible values).
int HorzVert: 0 to retrieve horizontal binning limit, 1 to retreive limit in the vertical.
int* MaxBinning: Will contain the Maximum binning value on return.

Return unsigned int

DRV_SUCCESS	Maximum Binning returned
DRV_NOT_INITIALIZED	System not initialized
DRV_P1INVALID	Invalid Readmode
DRV_P2INVALID	HorzVert not equal to 0 or 1
DRV_P3INVALID	Invalid MaxBinning address (i.e. NULL)

See also GetMinimumImageLength, SetReadMode

GetMaximumExposure

unsigned int WINAPI GetMaximumExposure (float* MaxExp)

Description This function will return the maximum Exposure Time in seconds that is settable by the SetExposureTime function.

Parameters Float int* MaxExp: Will contain the Maximum exposure value on return.

Return unsigned int

DRV_SUCCESS	Maximum Exposure returned.
DRV_P1INVALID	Invalid MaxExp value (i.e. NULL)

See also SetExposureTime

GetMCPGain

unsigned int WINAPI GetMCPGain (int* pi_gain)

Description This function will retrieve the set value for the MCP Gain.

Parameters int* pi_gain: Returned gain value.

Return unsigned int

DRV_SUCCESS	Table returned
DRV_NOT_INITIALIZED	System not initialized
DRV_ACQUIRING	Acquisition in progress
DRV_P1INVALID	Invalid pointer (i.e. NULL)
DRV_NOT_AVAILABLE	Not a USB iStar

See also SetMCPGain

NOTE **Available only on USB iStar.**

This function previously returned a table of MCP gain values against photoelectrons per count. This is now retrieved using GetMCPGainTable.

GetMCPGainRange

unsigned int WINAPI GetMCPGainRange(int* iLow, int* iHigh)

Description Returns the minimum and maximum values of the SetMCPGain function.

Parameters int* iLow: lowest gain setting

int* iHigh: highest gain setting

Return

DRV_SUCCESS	Gain range returned.
DRV_NOT_INITIALIZED	System not initialized.

See also SetMCPGain

NOTE **Available only iStar.**

GetMCPVoltage

unsigned int WINAPI GetMCPVoltage (int* iVoltage)

Description	This function will retrieve the current Micro Channel Plate voltage.
Parameters	int* iVoltage: Will contain voltage on return. The unit is in Volts and should be between the range 600 – 1100 Volts.
Return	unsigned int

DRV_SUCCESS	Voltage returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Not a USB iStar.
DRV_GENERAL_ERRORS	EEPROM not valid

See also GetMCPGain

NOTE **Available only on USB iStar.**

GetMetaDataInfo

unsigned int WINAPI GetMetaDataInfo(SYSTEMTIME* TimeOfStart ,float *TimeFromStart, int index)

Description	This function will return the time of the initial frame and the time in milliseconds of further frames from this point.
Parameters	SYSTEMTIME* TimeOfStart: Structure with start time details. float *TimeFromStart: time in milliseconds for a particular frame from time of start. int index: frame for which time is required.
Return	unsigned int

DRV_SUCCESS	Timings returned
DRV_NOT_INITIALIZED	System not initialized
DRV_MSTIMINGS_ERROR	Invalid timing request

See also SetMetaData

GetMinimumImageLength

unsigned int WINAPI GetMinimumImageLength (int* MinImageLength)

Description	This function will return the minimum number of pixels that can be read out from the chip at each exposure. This minimum value arises due the way in which the chip is read out and will limit the possible sub image dimensions and binning sizes that can be applied.	
Parameters	int* MinImageLength: Will contain the minimum number of super pixels on return.	
Return	unsigned int	
	DRV_SUCCESS	Minimum Number of Pixels returned
	DRV_NOT_INITIALIZED	System not initialized
	DRV_P1INVALID	Invalid MinImageLength value (i.e. NULL)
See also	SetImage	

GetMostRecentColorImage16

unsigned int WINAPI GetMostRecentColorImage16 (unsigned long size, int algorithm, WORD* red, WORD* green, WORD* blue)

Description **For colour sensors only.**

Color version of the GetMostRecentImage16 function. The CCD is sensitive to Cyan, Yellow, Magenta and Green (CYMG). The Red, Green and Blue (RGB) are calculated and Data is stored in 3 planes/images, one for each basic color.

Parameters unsigned long size: total number of pixels.
int algorithm: algorithm used to extract the RGB from the original CYMG CCD.
0: basic algorithm combining Cyan, Yellow and Magenta.
1: algorithm combining Cyan, Yellow, Magenta and Green.
WORD* red: pointer to red data storage allocated by the user.
WORD* green: pointer to red data storage allocated by the user.
WORD* blue: pointer to red data storage allocated by the user.

Return unsigned int

DRV_SUCCESS	Image RGB has been copied into arrays.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Arrays size is incorrect.
DRV_P2INVALID	Invalid algorithm.
DRV_P3INVALID	Invalid red pointer (i.e. NULL)..
DRV_P4INVALID	Invalid green pointer (i.e. NULL)..
DRV_P5INVALID	Invalid bluepointer (i.e. NULL)..
DRV_NO_NEW_DATA	There is no new data yet.

See also GetMostRecentImage16, DemosaicImage, WhiteBalance.

GetMostRecentImage

unsigned int WINAPI GetMostRecentImage(at_32* arr, unsigned long size)

Description This function will update the data array with the most recently acquired image in any acquisition mode. The data are returned as long integers (32-bit signed integers). The "array" must be exactly the same size as the complete image.

Parameters long* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Image has been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetMostRecentImage16, GetOldestImage, GetOldestImage16, GetImages

GetMostRecentImage16

unsigned int WINAPI GetMostRecentImage16(WORD* arr, unsigned long size)

Description 16-bit version of the GetMostRecentImage function.

Parameters WORD* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Image has been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetMostRecentImage, GetOldestImage16, GetOldestImage, GetImages

GetMSTimingsData

unsigned int WINAPI GetMSTimingsData(SYSTEMTIME *TimeOfStart ,float *pfDifferences, int inoOfimages)

Description **THIS FUNCTION IS RESERVED.**

GetMSTimingsEnabled

unsigned int WINAPI GetMSTimingsEnabled(void)

Description **THIS FUNCTION IS RESERVED.**

GetNewData

unsigned int WINAPI GetNewData(at_32* arr, unsigned long size)

Description

Deprecated see Note:

This function will update the data array to hold data acquired so far. The data are returned as long integers (32-bit signed integers). The “array” must be large enough to hold the complete data set. When used in conjunction with the SetDriverEvent and GetAcquisitonProgress functions, the data from each scan in a kinetic series can be processed while the acquisition is taking place.

Parameters At_* array: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Data copied.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also SetDriverEvent, GetAcquisitionProgress, SetAcquisitionMode, GetNewData8, GetNewData16

NOTE: Deprecated by the following functions:

- **GetImages**
- **GetMostRecentImage**
- **GetOldestImage**

GetNewData16

unsigned int WINAPI GetNewData16(WORD* arr, unsigned long size)

Description **Deprecated see Note:**

16-bit version of the GetNewData function.

Parameters WORD* arr: pointer to data storage allocated by the user.

unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Data copied.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

NOTE: Deprecated by the following functions:

- GetImages
- GetMostRecentImage
- GetOldestImage

GetNewData8

unsigned int WINAPI GetNewData8(unsigned char* arr, unsigned long size)

Description **Deprecated see Note:**

8-bit version of the GetNewData function. This function will return the data in the lower 8 bits of the acquired data.

Parameters unsigned char* arr: pointer to data storage allocated by the user.

unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Data copied.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

NOTE: Deprecated by the following functions:

- GetImages
- GetMostRecentImage
- GetOldestImage

GetNewFloatData

unsigned int WINAPI GetNewFloatData(float* arr, unsigned long size)

Description **THIS FUNCTION IS RESERVED.**

GetNumberADChannels

unsigned int WINAPI GetNumberADChannels(int* channels)

Description As your Andor SDK system may be capable of operating with more than one A-D converter, this function will tell you the number available.

Parameters int* channels: number of allowed channels

Return unsigned int

DRV_SUCCESS	Number of channels returned.

See also SetADChannel

GetNumberAmp

unsigned int WINAPI GetNumberAmp(int* amp)

Description As your Andor SDK system may be capable of operating with more than one output amplifier, this function will tell you the number available.

Parameters int* amp: number of allowed channels

Return unsigned int

DRV_SUCCESS	Number of output amplifiers returned.

See also SetOutputAmplifier

GetNumberAvailableImages

unsigned int WINAPI GetNumberAvailableImages (at_32* first, at_32* last)

Description This function will return information on the number of available images in the circular buffer. This information can be used with GetImages to retrieve a series of images. If any images are overwritten in the circular buffer they no longer can be retrieved and the information returned will treat overwritten images as not available.

Parameters at_32* first: returns the index of the first available image in the circular buffer.
at_32* last: returns the index of the last available image in the circular buffer.

Return unsigned int

DRV_SUCCESS	Number of acquired images returned
DRV_NOT_INITIALIZED	System not initialized
DRV_ERROR_ACK	Unable to communicate with card
DRV_NO_NEW_DATA	There is no new data yet

See also GetImages, GetImages16, GetNumberNewImages.

GetNumberDevices

unsigned int WINAPI GetNumberDevices (int* numDevs)

Description **THIS FUNCTION IS RESERVED.**

GetNumberFKVShiftSpeeds

unsigned int WINAPI GetNumberFKVShiftSpeeds(int* number)

Description As your Andor SDK system is capable of operating at more than one fast kinetics vertical shift speed this function will return the actual number of speeds available.

Parameters int* number: number of allowed speeds

Return unsigned int

DRV_SUCCESS	Number of speeds returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetFKVShiftSpeedF, SetFKVShiftSpeed

NOTE: Only available if camera is Classic or iStar.

GetNumberHorizontalSpeeds

unsigned int WINAPI GetNumberHorizontalSpeeds(int* number)

Description **Deprecated see Note:**

As your Andor SDK system is capable of operating at more than one horizontal shift speed this function will return the actual number of speeds available.

Parameters int* number: number of allowed horizontal speeds

Return unsigned int

DRV_SUCCESS	Number of speeds returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetHorizontalSpeed, SetHorizontalSpeed

NOTE: Deprecated by GetNumberHSSpeeds

GetNumberHSSpeeds

unsigned int WINAPI GetNumberHSSpeeds(int channel, int typ, int* speeds)

Description As your Andor SDK system is capable of operating at more than one horizontal shift speed this function will return the actual number of speeds available.

Parameters int channel: the AD channel.
int typ: output amplification.
Valid values: 0 electron multiplication.
1 conventional.
int* speeds: number of allowed horizontal speeds

Return unsigned int

DRV_SUCCESS	Number of speeds returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Invalid channel.
DRV_P2INVALID	Invalid horizontal read mode

See also GetHSSpeed, SetHSSpeed, GetNumberADChannels

GetNumberNewImages

unsigned int WINAPI GetNumberNewImages(long* first, long* last)

Description This function will return information on the number of new images (i.e. images which have not yet been retrieved) in the circular buffer. This information can be used with GetImages to retrieve a series of the latest images. If any images are overwritten in the circular buffer they can no longer be retrieved and the information returned will treat overwritten images as having been retrieved.

Parameters long* first: returns the index of the first available image in the circular buffer.
long* last: returns the index of the last available image in the circular buffer.

Return unsigned int

DRV_SUCCESS	Number of acquired images returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetImages, GetImages16, GetNumberAvailableImages

Note: This index will increment as soon as a single accumulation has been completed within the current acquisition.

GetNumberPhotonCountingDivisions

unsigned int WINAPI GetNumberPhotonCountingDivisions(unsigned long * noOfDivisions)

Description Available in some systems is photon counting mode. This function gets the number of photon counting divisions available. The functions SetPhotonCounting and SetPhotonCountingThreshold can be used to specify which of these divisions is to be used.

Parameters unsigned long* noOfDivisions: number of allowed photon counting divisions

Return unsigned int

DRV_SUCCESS	Number of photon counting divisions returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Invalid parameter.
DRV_NOT_AVAILABLE	Photon Counting not available

See also SetPhotonCounting, SetPhotonCountingThreshold, GetCapabilities

GetNumberPreAmpGains

unsigned int WINAPI GetNumberPreAmpGains(int* noGains)

Description Available in some systems are a number of pre amp gains that can be applied to the data as it is read out. This function gets the number of these pre amp gains available. The functions GetPreAmpGain and SetPreAmpGain can be used to specify which of these gains is to be used.

Parameters int* noGains: number of allowed pre amp gains

Return unsigned int

DRV_SUCCESS	Number of pre amp gains returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also IsPreAmpGainAvailable, GetPreAmpGain, SetPreAmpGain, GetCapabilities

GetNumberRingExposureTimes

unsigned int WINAPI GetNumberRingExposureTimes (int * ipnumTimes)

Description Gets the number of exposures in the ring at this moment.

Parameters int * ipnumTimes: Numberof exposure times.

Return unsigned int

DRV_SUCCESS	Success
DRV_NOT_INITIALIZED	System not initialized

See also SetRingExposureTimes

GetNumberIO

unsigned int WINAPI GetNumberIO(int* iNumber)

Description Available in some systems are a number of IO's that can be configured to be inputs or outputs. This function gets the number of these IO's available. The functions GetIODirection, GetIOLevel, SetIODirection and SetIOLevel can be used to specify the configuration.

Parameters int* iNumber: number of allowed IO's

Return unsigned int

DRV_SUCCESS	Number of IO's returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid parameter.
DRV_NOT_AVAILABLE	Feature not available.

See also GetIOLevel GetIODirection SetIODirection SetIOLevel

GetNumberVerticalSpeeds

unsigned int WINAPI GetNumberVerticalSpeeds(int* number)

Description **Deprecated see Note:**

As your Andor system may be capable of operating at more than one vertical shift speed this function will return the actual number of speeds available.

Parameters int* number: number of allowed vertical speeds

Return unsigned int

DRV_SUCCESS	Number of speeds returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetVerticalSpeed, SetVerticalSpeed

NOTE: Deprecated by GetNumberVSSpeeds

GetNumberVSAmplitudes

unsigned int WINAPI GetNumberVSAmplitudes (int* number)

Description This function will normally return the number of vertical clock voltage amplitues that the camera has.

Parameters int *number:

Return unsigned int

DRV_SUCCESS	Number returned
DRV_NOT_INITIALIZED	System not initialized
DRV_NOT_AVAILABLE	Your system does not support this feature

GetNumberVSSpeeds

unsigned int WINAPI GetNumberVSSpeeds(int* speeds)

Description As your Andor system may be capable of operating at more than one vertical shift speed this function will return the actual number of speeds available.

Parameters int* speeds: number of allowed vertical speeds

Return unsigned int

DRV_SUCCESS	Number of speeds returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetVSSpeed, SetVSSpeed, GetFastestRecommendedVSSpeed

GetOldestImage

unsigned int WINAPI GetOldestImage(at_32* arr, unsigned long size)

Description This function will update the data array with the oldest image in the circular buffer. Once the oldest image has been retrieved it no longer is available. The data are returned as long integers (32-bit signed integers). The "array" must be exactly the same size as the full image.

Parameters at_32* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Image has been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetOldestImage16, GetMostRecentImage, GetMostRecentImage16

GetOldestImage16

unsigned int WINAPI GetOldestImage16(WORD* arr, unsigned long size)

Description 16-bit version of the GetOldestImage function.

Parameters WORD* arr: pointer to data storage allocated by the user.
unsigned long size: total number of pixels.

Return unsigned int

DRV_SUCCESS	Image has been copied into array.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Array size is incorrect.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetOldestImage, GetMostRecentImage16, GetMostRecentImage

GetPhysicalDMAAddress

unsigned int WINAPI GetPhysicalDMAAddress (unsigned long* Address1, unsigned long* Address2)

Description **THIS FUNCTION IS RESERVED.**

GetPixelSize

unsigned int WINAPI GetPixelSize(float* xSize, float* ySize)

Description This function returns the dimension of the pixels in the detector in microns.

Parameters float* xSize: width of pixel.

float* ySize: height of pixel.

Return unsigned int

DRV_SUCCESS	Pixel size returned.

GetPreAmpGain

unsigned int WINAPI GetPreAmpGain(int index, float* gain)

Description For those systems that provide a number of pre amp gains to apply to the data as it is read out; this function retrieves the amount of gain that is stored for a particular index. The number of gains available can be obtained by calling the GetNumberPreAmpGains function and a specific Gain can be selected using the function SetPreAmpGain.

Parameters int index: gain index

Valid values: 0 to GetNumberPreAmpGains()-1

float* gain: gain factor for this index.

Return unsigned int

DRV_SUCCESS	Gain returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also IsPreAmpGainAvailable, GetNumberPreAmpGains, SetPreAmpGain, GetCapabilities

GetPreAmpGainText

unsigned int WINAPI GetPreAmpGainText (int index , char* name, int len)

Description This function will return a string with a pre amp gain description. The pre amp gain is selected using the index. The SDK has a string associated with each of its pre amp gains. The maximum number of characters needed to store the pre amp gain descriptions is 30. The user has to specify the number of characters they wish to have returned to them from this function.

Parameters int index: gain index

Valid values: 0 to GetNumberPreAmpGains()-1

char* name: A user allocated array of characters for storage of the description.

int len: The length of the user allocated character array.

Return unsigned int

DRV_SUCCESS	Description returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Invalid index.
DRV_P2INVALID	Array size is incorrect
DRV_NOT_SUPPORTED	Function not supported with this camera

See also IsPreAmpGainAvailable, GetNumberPreAmpGains, SetPreAmpGain, GetCapabilities

GetQE

unsigned int WINAPI GetQE(char * sensor, float wavelength, unsigned int mode, float * QE)

Description Returns the percentage QE for a particular head model at a user specified wavelength.

Parameters char* sensor: head model

float wavelength: wavelength at which QE is required

unsigned int mode: Clara mode (Normal (0) or Extended NIR (1)). 0 for all other systems

float* QE: requested QE

Return unsigned int

DRV_SUCCESS	QE returned.
DRV_NOT_INITIALIZED	System not initialized.

See also _GetHeadModel, GetCapabilities

GetReadOutTime

unsigned int WINAPI GetReadOutTime(float* ReadoutTime)

Description This function will return the time to readout data from a sensor. This function should be used after all the acquisitions settings have been set, e.g. SetExposureTime, SetKineticCycleTime and SetReadMode etc. The value returned is the actual times used in subsequent acquisitions.

Parameters float* ReadoutTime: valid readout time in seconds

Return unsigned int

DRV_SUCCESS	Timing information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_CODES	Error communicating with camera.

See also GetAcquisitionTimings GetKeepCleanTime

NOTES **NOTE: Available on iDus, iXon, Luca & Newton.**

GetRegisterDump

unsigned int WINAPI GetRegisterDump (int* mode)

Description **THIS FUNCTION IS RESERVED.**

GetRingExposureRange

unsigned int WINAPI GetRingExposureRange (float * fpMin, float * fpMax)

Description With the Ring Of Exposure feature there may be a case when not all exposures can be met. The ring of exposure feature will guarantee that the highest exposure will be met but this may mean that the lower exposures may not be. If the lower exposures are too low they will be increased to the lowest value possible. This function will return these upper and lower values.

Parameters float * fpMin: Minimum exposure

float * fpMax: Maximum exposure.

Return unsigned int

DRV_SUCCESS	Min and max returned
DRV_NOT_INITIALIZED	System not initialize
DRV_INVALID_MODE	Trigger mode is not available

See also GetCapabilities, GetNumberRingExposureTimes, IsTriggerModeAvailable, SetRingExposureTimes

---

GetSensitivity

unsigned int WINAPI GetSensitivity(int channel, int index, int amplifier, int pa, float* sensitivity)

Description This function returns the sensitivity for a particular speed.

Parameters
int channel: AD channel index.
int amplifier: Type of output amplifier.
int index: Channel speed index.
int pa: PreAmp gain index.
float* sensitivity: requested sensitivity.

Return unsigned int

DRV_SUCCESS	Sensitivity returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid channel.
DRV_P2INVALID	Invalid amplifier.
DRV_P3INVALID	Invalid speed index.
DRV_P4INVALID	Invalid gain.

See also GetCapabilities

NOTE: Available only on iXon+ and Clara.

GetSizeOfCircularBuffer

unsigned int WINAPI GetSizeOfCircularBuffer(long* index)

Description	This function will return the maximum number of images the circular buffer can store based on the current acquisition settings.
Parameters	long* index: returns the maximum number of images the circular buffer can store.
Return	unsigned int
	DRV_SUCCESS — Maximum number of images returned.
	DRV_NOT_INITIALIZED — System not initialized.

GetSlotBusDeviceFunction

unsigned int WINAPI GetSlotBusDeviceFunction (DWORD *dwSlot, DWORD *dwBus, DWORD *dwDevice, DWORD *dwFunction)

Description **THIS FUNCTION IS RESERVED**

GetSoftwareVersion

unsigned int WINAPI GetSoftwareVersion(unsigned int* eprom, unsigned int* cofFile, unsigned int* vxdRev, unsigned int* vxdVer, unsigned int* dllRev, unsigned int* dllVer)

Description This function returns the Software version information for the microprocessor code and the driver.

Parameters
unsigned int* eprom: EPROM version
unsigned int* cofFile: COF file version
unsigned int *vxdRev: Driver revision number
unsigned int *vxdVer: Driver version number
unsigned int *dllRev: DLL revision number
unsigned int *dllVer: DLL version number

Return unsigned int

DRV_SUCCESS	Version information returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

GetSpoolProgress

unsigned int WINAPI GetSpoolProgress(long* index)

Description

Deprecated see Note:

This function will return information on the progress of the current spool operation. The value returned is the number of images that have been saved to disk during the current kinetic scan.

Parameters long* index: returns the number of files saved to disk in the current kinetic scan.

Return unsigned int

DRV_SUCCESS	Spool progress returned.
DRV_NOT_INITIALIZED	System not initialized.

See also SetSpool

NOTE: Deprecated by GetTotalNumberImagesAcquired

GetStatus

unsigned int WINAPI GetStatus(int* status)

Description This function will return the current status of the Andor SDK system. This function should be called before an acquisition is started to ensure that it is IDLE and during an acquisition to monitor the process.

Parameters int* status: current status

DRV_IDLE	IDLE waiting on instructions.
DRV_TEMPCYCLE	Executing temperature cycle.
DRV_ACQUIRING	Acquisition in progress.
DRV_ACCUM_TIME_NOT_MET	Unable to meet Accumulate cycle time.
DRV_KINETIC_TIME_NOT_MET	Unable to meet Kinetic cycle time.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_ACQ_BUFFER	Computer unable to read the data via the ISA slot at the required rate.
DRV_SPOOLERROR	Overflow of the spool buffer.

Return unsigned int

DRV_SUCCESS	Status returned
DRV_NOT_INITIALIZED	System not initialized

See also SetTemperature, StartAcquisition

NOTE: If the status is one of the following:

- **DRV_ACCUM_TIME_NOT_MET**
- **DRV_KINETIC_TIME_NOT_MET**
- **DRV_ERROR_ACK**
- **DRV_ACQ_BUFFER**

then the current acquisition will be aborted automatically.

GetTemperature

unsigned int WINAPI GetTemperature(int* temperature)

Description This function returns the temperature of the detector to the nearest degree. It also gives the status of cooling process.

Parameters int* temperature: temperature of the detector

Return unsigned int

DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_TEMP_OFF	Temperature is OFF.
DRV_TEMP_STABILIZED	Temperature has stabilized at set point.
DRV_TEMP_NOT_REACHED	Temperature has not reached set point.
DRV_TEMP_DRIFT	Temperature had stabilized but has since drifted
DRV_TEMP_NOT_STABILIZED	Temperature reached but not stabilized

See also GetTemperatureF, SetTemperature, CoolerON, CoolerOFF, GetTemperatureRange

GetTemperatureF

unsigned int WINAPI GetTemperatureF(float* temperature)

Description This function returns the temperature in degrees of the detector. It also gives the status of cooling process.

Parameters float* temperature: temperature of the detector

Return unsigned int

DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_TEMP_OFF	Temperature is OFF.
DRV_TEMP_STABILIZED	Temperature has stabilized at set point.
DRV_TEMP_NOT_REACHED	Temperature has not reached set point.
DRV_TEMP_DRIFT	Temperature had stabilised but has since drifted
DRV_TEMP_NOT_STABILIZED	Temperature reached but not stabilized

See also GetTemperature, SetTemperature, CoolerON, CoolerOFF, GetTemperatureRange

GetTemperatureRange

unsigned int WINAPI GetTemperatureRange(int* mintemp, int* maxtemp)

Description This function returns the valid range of temperatures in centigrads to which the detector can be cooled.

Parameters int* mintemp: minimum temperature
int* maxtemp: maximum temperature

Return unsigned int

DRV_SUCCESS	Temperature range returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetTemperature, GetTemperatureF, SetTemperature, CoolerON, CoolerOFF

GetTemperatureStatus

unsigned int WINAPI GetTemperatureStatus (float *SensorTemp, float *TargetTemp, float *AmbientTemp, float *CoolerVolts)

Description **THIS FUNCTION IS RESERVED.**

GetTotalNumberImagesAcquired

unsigned int WINAPI GetTotalNumberImagesAcquired(long* index)

Description This function will return the total number of images acquired since the current acquisition started. If the camera is idle the value returned is the number of images acquired during the last acquisition.

Parameters long* index: returns the total number of images acquired since the acquisition started.

Return unsigned int

DRV_SUCCESS	Number of acquired images returned.
DRV_NOT_INITIALIZED	System not initialized.

GetIODirection

unsigned int WINAPI GetIODirection(int index, int* iDirection)

Description Available in some systems are a number of IO's that can be configured to be inputs or outputs. This function gets the current state of a particular IO.

Parameters int index: IO index
Valid values: 0 toGetNumberIO() - 1
int* iDirection: current direction for this index.
0: Output
1: Input

Return unsigned int

DRV_SUCCESS	IO direction returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.
DRV_P2INVALID	Invalid parameter.
DRV_NOT_AVAILABLE	Feature not available.

See also GetNumberIO GetIOLevel SetIODirection SetIOLevel

GetIOLevel

unsigned int WINAPI GetIOLevel(int index, int* iLevel)

Description Available in some systems are a number of IO's that can be configured to be inputs or outputs. This function gets the current state of a particular IO.

Parameters int index: IO index
Valid values: 0 toGetNumberIO() - 1
int* iLevel: current level for this index.
0: Low
1: High

Return unsigned int

DRV_SUCCESS	IO level returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.
DRV_P2INVALID	Invalid parameter.
DRV_NOT_AVAILABLE	Feature not available.

See also GetNumberIO GetIODirection SetIODirection SetIOLevel

GetVersionInfo

unsigned int WINAPI GetVersionInfo (AT_VersionInfoId arr, char* szVersionInfo, at_u32 ui32BufferLen)

Description This function retrieves version information about different aspects of the Andor system. The information is copied into a passed string buffer. Currently, the version of the SDK and the Device Driver (USB or PCI) is supported.

Parameters AT_VersionInfoId arr:

AT_SDKVersion: requests the SDK version information

AT_DeviceDriverVersion: requests the device driver version

char* szVersionInfo: A user allocated array of characters for storage of the information

at_u32 ui32BufferLen: The size of the passed character array, *versionInfo*.

Return unsigned int

DRV_SUCCESS	Information returned
DRV_NOT_INITIALIZED	System not initialized
DRV_P1INVALID	Invalid information type requested
DRV_P2INVALID	Storage array pointer is NULL
DRV_P3INVALID	Size of the storage array is zero

See also GetHeadModel, GetCameraSerialNumber, GetCameraInformation, GetCapabilities

GetVerticalSpeed

unsigned int WINAPI GetVerticalSpeed(int index, int* speed)

Description

Deprecated see Note:

As your Andor system may be capable of operating at more than one vertical shift speed this function will return the actual speeds available. The value returned is in microseconds per pixel shift.

Parameters int index: speed required

Valid values 0 to GetNumberVerticalSpeeds()-1

int* speed: speed in microseconds per pixel shift.

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also GetNumberVerticalSpeeds, SetVerticalSpeed

NOTE: Deprecated by GetVSSpeed.

GetVirtualDMAAddress

unsigned int WINAPI GetVirtualDMAAddress(void Address1, void** Address2)**

Description **THIS FUNCTION IS RESERVED.**

GetVSSpeed

unsigned int WINAPI GetVSSpeed(int index, float* speed)

Description As your Andor SDK system may be capable of operating at more than one vertical shift speed this function will return the actual speeds available. The value returned is in microseconds.

Parameters int index: speed required

Valid values 0 to GetNumberVSSpeeds()-1

float* speed: speed in microseconds per pixel shift.

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.

See also GetNumberVSSpeeds, SetVSSpeed, GetFastestRecommendedVSSpeed

GPIBReceive

unsigned int WINAPI GPIBReceive(int id, short address, char* text, int size)

Description This function reads data from a device until a byte is received with the EOI line asserted or until size bytes have been read.

Parameters int id: The interface board number
short address: Address of device to send data
char* text: The data to be sent
int size: Number of characters to read

Return unsigned int

DRV_SUCCESS	Data received.
DRV_P3INVALID	Invalid pointer (e.g. NULL). Other errors may be returned by the GPIB device. Consult the help documentation supplied with these devices

See also GPIBSend

GPIBSend

unsigned int WINAPI GPIBSend(int id, short address, char* text)

Description This function initializes the GPIB by sending interface clear. Then the device described by address is put in a listen-active state. Finally the string of characters, text, is sent to the device with a newline character and with the EOI line asserted after the final character.

Parameters int id: The interface board number
short address: Address of device to send data
char* text: The data to be sent

Return unsigned int

DRV_SUCCESS	Data sent.
DRV_P3INVALID	Invalid pointer (e.g. NULL). The GPIB device may return other errors. Consult the help documentation supplied with these devices

See also GPIBReceive

I2CBurstRead

unsigned int WINAPI I2CBurstRead(BYTE i2cAddress, long nBytes, BYTE* data)

Description This function will read a specified number of bytes from a chosen device attached to the I^2C data bus.

Parameters BYTE i2cAddress: The address of the device to read from.
long nBytes: The number of bytes to read from the device.
BYTE* data: The data read from the device.

Return unsigned int

DRV_SUCCESS	Read successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_I2CDEVNOTFOUND	Could not find the specified device.
DRV_I2CTIMEOUT	Timed out reading from device.
DRV_UNKNOWN_FUNC	Unknown function, incorrect cof file.

See also I2CBurstWrite, I2CRead, I2CWrite, I2cReset

I2CBurstWrite

unsigned int WINAPI I2CBurstWrite(BYTE i2cAddress, long nBytes, BYTE* data)

Description This function will write a specified number of bytes to a chosen device attached to the I^2C data bus.

Parameters BYTE i2cAddress: The address of the device to write to.
long nBytes: The number of bytes to write to the device.
BYTE* data: The data to write to the device.

Return unsigned int

DRV_SUCCESS	Write successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_I2CDEVNOTFOUND	Could not find the specified device.
DRV_I2CTIMEOUT	Timed out reading from device.
DRV_UNKNOWN_FUNC	Unknown function, incorrect cof file.

See also I2CBurstRead, I2CRead, I2CWrite, I2cReset

I2CRead

unsigned int WINAPI I2CRead(BYTE deviceID, BYTE intAddress, BYTE* pdata)

Description This function will read a single byte from the chosen device.

Parameters BYTE deviceID: The device to read from.

BYTE intAddress: The internal address of the device to be read from.

BYTE* pdata: The byte read from the device.

Return unsigned int

DRV_SUCCESS	Read successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_I2CDEVNOTFOUND	Could not find the specified device.
DRV_I2CTIMEOUT	Timed out reading from device.
DRV_UNKNOWN_FUNC	Unknown function, incorrect cof file.

See also I2CBurstWrite, I2CBurstRead, I2CWrite, I2cReset

I2CReset

unsigned int WINAPI I2CReset(void)

Description This function will reset the I^2C data bus.

Parameters

Return unsigned int

DRV_SUCCESS	Reset successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_I2CTIMEOUT	Timed out reading from device.
DRV_UNKNOWN_FUNC	Unknown function, incorrect cof file.

See also I2CBurstWrite, I2CBurstRead,I2CRead, I2CWrite

I2CWrite

unsigned int WINAPI I2CWrite(BYTE deviceID, BYTE intAddress, BYTE data)

Description This function will write a single byte to the chosen device.

Parameters BYTE deviceID: The device to write to.
BYTE intAddress: The internal address of the device to write to.
BYTE data: The byte to be written to the device.

Return unsigned int

DRV_SUCCESS	Write successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_I2CDEVNOTFOUND	Could not find the specified device.
DRV_I2CTIMEOUT	Timed out reading from device.
DRV_UNKNOWN_FUNC	Unknown function, incorrect cof file.

See also I2CBurstWrite, I2CBurstRead, I2CRead, I2cReset

IdAndorDll

unsigned int WINAPI IdAndorDll (void)

Description **THIS FUNCTION IS RESERVED.**

InAuxPort

unsigned int WINAPI InAuxPort(int port, int* state)

Description This function returns the state of the TTL Auxiliary Input Port on the Andor plug-in card.

Parameters int port: Number of AUX in port on Andor card
Valid Values 1 to 4
int* state: current state of port
0 OFF/LOW
all otherON/HIGH

Return unsigned int

DRV_SUCCESS	AUX read.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid port id.

See also OutAuxPort

Initialize

unsigned int WINAPI Initialize(char* dir)

Description This function will initialize the Andor SDK System. As part of the initialization procedure on some cameras (i.e. Classic, iStar and earlier iXion) the **DLL** will need access to a **DETECTOR.INI** which contains information relating to the detector head, number pixels, readout speeds etc. If your system has multiple cameras then see the section Controlling multiple cameras

Parameters char* dir: Path to the directory containing the files

Return unsigned int

DRV_SUCCESS	Initialisation successful.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_INIERROR	Unable to load “DETECTOR.INI”.
DRV_COFERROR	Unable to load “*.COF”.
DRV_FLEXERROR	Unable to load “*.RBF”.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_ERROR_FILELOAD	Unable to load “*.COF” or “*.RBF” files.
DRV_ERROR_PAGELOCK	Unable to acquire lock on requested memory.
DRV_USBERROR	Unable to detect USB device or not USB2.0.
DRV_ERROR_NOCAMERA	No camera found

See also GetAvailableCameras, SetCurrentCamera, GetCurrentCamera

InitializeDevice

unsigned int WINAPI InitializeDevice(char * dir)

Description **THIS FUNCTION IS RESERVED.**

IsCoolerOn

unsigned int WINAPI IsCoolerOn (int* iCoolerStatus)

Description This function checks the status of the cooler.

Parameters int* iCoolerStatus: 0: Cooler is OFF.
1: Cooler is ON.

Return unsigned int

DRV_SUCCESS	Status returned.
DRV_NOT_INITIALIZED	System not initialized
DRV_P1INVALID	Parameter is NULL

See also CoolerON CoolerOFF

IsCountConvertModeAvailable

unsigned int WINAPI IsCountConvertModeAvailable (int mode)

Description This function checks if the hardware and current settings permit the use of the specified Count Convert mode.

Parameters int mode: Count Convert mode to be checked

Return unsigned int

DRV_SUCCESS	Count Convert mode available.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Count Convert not supported on this camera
DRV_INVALID_COUNTCONVERT_MODE	Count Convert mode not available with current acquisition settings

See also GetCapabilities, SetCountConvertMode, SetCountConvertWavelength

IsInternalMechanicalShutter

unsigned int WINAPI IsInternalMechanicalShutter (int* InternalShutter)

Description This function checks if an iXon camera has a mechanical shutter installed.

Parameters int* InternalShutter: 0: Mechanical shutter not installed.
1: Mechanical shutter installed.

Return unsigned int

DRV_SUCCESS	Internal Shutter state returned
DRV_NOT_AVAILABLE	Not an iXon camera.
DRV_P1INVALID	Parameter is NULL

NOTE **Available only on iXon.**

IsAmplifierAvailable

unsigned int WINAPI IsAmplifierAvailable(int iamp)

Description This function checks if the hardware and current settings permit the use of the specified amplifier.

Parameters int iamp: amplifier to check.

Return unsigned int

DRV_SUCCESS	Amplifier available
DRV_NOT_INITIALIZED	System not initialized
DRV_INVALID_AMPLIFIER	Not a valid amplifier

See also SetHSSpeed

IsPreAmpGainAvailable

unsigned int WINAPI IsPreAmpGainAvailable(int channel, int amplifier, int index, int pa, int* status)

Description This function checks that the AD channel exists, and that the amplifier, speed and gain are available for the AD channel.

Parameters
int channel: AD channel index.
int amplifier: Type of output amplifier.
int index: Channel speed index.
int pa: PreAmp gain index.
int* status: 0: PreAmpGain not available.
1: PreAmpGain Available.

Return unsigned int

DRV_SUCCESS	PreAmpGain status returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid channel.
DRV_P2INVALID	Invalid amplifier.
DRV_P3INVALID	Invalid speed index.
DRV_P4INVALID	Invalid gain.

See also GetNumberPreAmpGains, GetPreAmpGain, SetPreAmpGain

NOTE: Available only on iXon.

IsTriggerModeAvailable

unsigned int WINAPI IsTriggerModeAvailable(int iTriggerMode)

Description	This function checks if the hardware and current settings permit the use of the specified trigger mode.	
Parameters	int iTriggerMode: Trigger mode to check.	
Return	unsigned int	
	DRV_SUCCESS	Trigger mode available
	DRV_NOT_INITIALIZED	System not initialize
	DRV_INVALID_MODE	Not a valid mode
See also	SetTriggerMode	

Merge

unsigned int WINAPI Merge(const at_32* arr, long nOrder, long nPoint, long nPixel, float* coeff, long fit,long hbin, at_32* output, float* start, float* step)

Description	**THIS FUNCTION IS RESERVED.**

OA_AddMode

unsigned int WINAPI OA_AddMode (char* ModeName, unsigned int ModeNameLen, char * ModeDescription, unsigned int ModeDescriptionLen)

Description	This function will add a mode name and description to memory. Note that this will not add the mode to file, a subsequent call to OA_WriteToFile must be made.	
Parameters	char* ModeName:	A name for the mode to be defined.
	unsigned int ModeNameLen:	Mode name string length.
	char* modeDescription:	A description of the user defined mode.
	unsigned int ModeDescriptionLen:	Mode Description string length.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted
	DRV_P1INVALID	Null mode name.
	DRV_P3INVALID	Null mode description.
	DRV_OA_INVALID_STRING_LENGTH	One or more parameters have an invalid length, i.e. > 255.
	DRV_OA_INVALID_NAMING	Mode and description have the same name, this is not valid.
	DRV_OA_MODE_BUFFER_FULL	Number of modes exceeds limit.
	DRV_OA_INVALID_CHARS_IN_NAME	Mode name and/or description contain invalid characters.
	DRV_OA_MODE_ALREADY_EXISTS	Mode name already exists in the file.
	DRV_OA_INVALID_CHARS_IN_NAME	Invalid charcters in Mode Name or Mode Description
See also	OA_DeleteMode, OA_WriteToFile	

OA_DeleteMode

unsigned int WINAPI OA_DeleteMode (const char* const ModeName, unsigned int ModeNameLen)

Description	This function will remove a mode from memory. To permanently remove a mode from file, call OA_WriteToFile after OA_DeleteMode. The Preset file will not be affected.	
Parameters	const char* const ModeName:	The name of the mode to be removed.
	unsigned int ModeNameLen:	Mode name string length.
Return	unsigned int	All parameters accepted
	DRV_SUCCESS	Null mode name.
	DRV_P1INVALID	The mode name parameter has an invalid length, i.e. > 256.
	DRV_OA_INVALID_STRING_LENGTH	Mode does not exist.
	DRV_OA_MODE_DOES_NOT_EXIST	
See also	OA_AddMode, OA_WriteToFile	

OA_EnableMode

unsigned int WINAPI OA_EnableMode (const char* const ModeName)

Description	This function will set all the parameters associated with the specified mode to be used for all subsequent acquisitions. The mode specified by the user must be in either the Preset file or the User defined file.	
Parameters	const char* const ModeName: The mode to be used for all subsequent acquisitions.	
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted
	DRV_P1INVALID	Null mode name.
	DRV_OA_MODE_DOES_NOT_EXIST	Mode name does not exist.
	DRV_OA_CAMERA_NOT_SUPPORTED	Camera not supported.
See also	OA_AddMode	

OA_GetFloat

unsigned int WINAPI OA_GetFloat (const char* const ModeName, const char* const ModeParam, float* FloatValue)

Description	This function is used to get the values for floating point type acquisition parameters. Values are retrieved from memory for the specified mode name.	
Parameters	const char* const ModeName:	The name of the mode for which an acquisition parameter will be retrieved.
	const char* const ModeParam:	The name of the acquisition parameter for which a value will be retrieved.
	float* FloatValue:	The value of the acquisition parameter.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted
	DRV_P1INVALID	Null mode parameter.
	DRV_P2INVALID	Null mode parameter.
	DRV_P3INVALID	Null float value.
See also	OA_SetFloat	

OA_GetInt

unsigned int WINAPI OA_GetInt (const char* const ModeName, const char* const ModeParam, int* IntValue)

Description	This function is used to get the values for integer type acquisition parameters. Values are retrieved from memory for the specified mode name.	
Parameters	const char* const ModeName:	The name of the mode for which an acquisition parameter will be retrieved.
	const char* const ModeParam:	The name of the acquisition parameter for which a value will be retrieved.
	int* IntValue:	The buffer to return the value of the acquisition parameter.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null mode parameter.
	DRV_P3INVALID	Null integer value.

See also OA_SetInt

OA_GetModeAcqParams

unsigned int WINAPI OA_GetModeAcqParams (const char* const ModeName, char * const ListOfParams)

Description	This function will return all acquisition parameters associated with the specified mode. The mode specified by the user must be in either the Preset file or the User defined file. The user must allocate enough memory for all of the acquisition parameters.	
Parameters	const char* const ModeName:	The mode for which all acquisition parameters must be returned.
	char * const ListOfParams:	A user allocated array of characters for storage of the acquisition parameters. Parameters will be delimited by a ‘,’.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null mode parameter.
	DRV_OA_NO_USER_DATA	No data for selected mode.

See also OA_GetNumberOfAcqParams

OA_GetNumberOfAcqParams

unsigned int WINAPI OA_GetNumberOfAcqParams (const char* const ModeName, unsigned int* const NumberOfParams)

Description	This function will return the parameters associated with a specified mode. The mode must be present in either the Preset file or the User defined file.	
Parameters	const char* const ModeName	The mode to search for a list of acquisition parameters.
	unsigned int* const NumberOfParams:	The number of acquisition parameters for the specified mode.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null number of parameters.
	DRV_OA_NULL_ERROR	Invalid pointer.
See also	OA_GetModeAcqParams	

OA_GetNumberOfPreSetModes

unsigned int WINAPI OA_GetNumberOfPreSetModes (unsigned int* const NumberOfModes)

Description	This function will return the number of modes defined in the Preset file. The Preset file must exist.	
Parameters	unsigned int* const NumberOfModes:	The number of modes in the Andor file.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null number of modes.
	DRV_OA_NULL_ERROR	Invalid pointer.
	DRV_OA_BUFFER_FULL	Number of modes exceeds limit.
See also	OA_GetPreSetModeNames	

OA_GetNumberOfUserModes

unsigned int WINAPI OA_GetNumberOfUserModes (unsigned int* const NumberOfModes)

Description	This function will return the number of modes defined in the User file. The user defined file must exist.	
Parameters	unsigned int* const NumberOfModes:	The number of modes in the user file.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null number of modes.
	DRV_OA_NULL_ERROR	Invalid pointer.
	DRV_OA_BUFFER_FULL	Number of modes exceeds limit.
See also	OA_GetUserModeNames	

OA_GetPreSetModeNames

unsigned int WINAPI OA_GetPreSetModeNames (char * ListOfModes)

Description This function will return the available mode names from the Preset file. The mode and the Preset file must exist. The user must allocate enough memory for all of the acquisition parameters.

Parameters char * ListOfModes: A user allocated array of characters for storage of the mode names. Mode names will be delimited by a ‘,’.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_P1INVALID	Null list of modes.
DRV_OA_NULL_ERROR	Invalid pointer.

See also OA_GetNumberOfPreSetModes

OA_GetString

unsigned int WINAPI OA_GetString (const char* const ModeName, const char* const ModeParam, char* StringValue, const unsigned int StringLen)

Description This function is used to get the values for string type acquisition parameters. Values are retrieved from memory for the specified mode name.

Parameters

const char* const ModeName:	The name of the mode for which an acquisition parameter will be retrieved.
const char* const ModeParam:	The name of the acquisition parameter for which a value will be retrieved.
char* StringValue:	The buffer to return the value of the acquisition parameter.
const unsigned int StringLen:	The length of the buffer.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_P1INVALID	Null mode name.
DRV_P2INVALID	Null mode parameter.
DRV_P3INVALID	Null string value.
DRV_P4INVALID	Invalid string length

See also OA_SetString

OA_GetUserModeNames

unsigned int WINAPI OA_GetUserModeNames (char * ListOfModes)

Description This function will return the available mode names from a User defined file. The mode and the User defined file must exist. The user must allocate enough memory for all of the acquisition parameters.

Parameters char * ListOfModes: A user allocated array of characters for storage of the mode names. Mode names will be delimited by a ‘,’.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_P1INVALID	Null list of modes.
DRV_OA_NULL_ERROR	Invalid pointer.

See also OA_GetNumberOfUserModes

OA_Initialize

unsigned int WINAPI OA_Initialize (const char * const Filename, unsigned int FileNameLen)

Description This function will initialise the OptAcquire settings from a Preset file and a User defined file if it exists.

Parameters char* const Filename: The name of a user xml file. If the file exists then data will be read from the file. If the file does not exist the file name may be used when the user calls WriteToFile().

unsigned int FileNameLen: The length of the filename.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_P1INVALID	Null filename.
DRV_OA_CAMERA_NOT_SUPPORTED	Camera not supported.
DRV_OA_GET_CAMERA_ERROR	Unable to retrieve information about the model of the Camera.
DRV_OA_INVALID_STRING_LENGTH	The parameter has an invalid length, i.e. > 255.
DRV_OA_ANDOR_FILE_NOT_LOADED	Preset Andor file failed to load.
DRV_OA_USER_FILE_NOT_LOADED	Supplied User file failed to load.
DRV_OA_FILE_ACCESS_ERROR	Failed to determine status of file.
DRV_OA_PRESET_AND_USER_FILE_NOT_LOADED	Failed to load Andor and User file.

See also OA_WriteToFile

OA_SetFloat

unsigned int WINAPI OA_SetFloat (const char* const ModeName, const char * ModeParam, const float FloatValue)

Description	This function is used to set values for floating point type acquisition parameters where the new values are stored in memory. To commit changes to file call WriteToFile().	
Parameters	const char* const ModeName:	The name of the mode for which an acquisition parameter will be edited.
	const char * const ModeParam:	The name of the acquisition parameter to be edited.
	const float FloatValue:	The value to assign to the acquisition parameter.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null mode parameter.
	DRV_OA_INVALID_STRING_LENGTH	One or more of the string parameters has an invalid length, i.e. > 255.
	DRV_OA_MODE_DOES_NOT_EXIST	The Mode does not exist.
See also	OA_GetFloat, OA_EnableMode, OA_WriteToFile	

OA_SetInt

unsigned int WINAPI OA_SetInt (const char* const ModeName, const char* ModeParam, const int IntValue)

Description	This function is used to set values for integer type acquisition parameters where the new values are stored in memory. To commit changes to file call WriteToFile().	
Parameters	const char* const ModeName:	The name of the mode for which an acquisition parameter will be edited.
	const char* const ModeParam:	The name of the acquisition parameter to be edited.
	const int IntValue:	The value to assign to the acquisition parameter.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null mode parameter.
	DRV_OA_INVALID_STRING_LENGTH	One or more of the string parameters has an invalid length, i.e. > 255.
	DRV_OA_MODE_DOES_NOT_EXIST	The Mode does not exist.
See also	OA_GetInt, OA_EnableMode, OA_WriteToFile	

OA_SetString

unsigned int WINAPI OA_SetString (const char* const ModeName, const char* ModeParam, char* StringValue, const unsigned int StringLen)

Description	This function is used to set values for string type acquisition parameters where the new values are stored in memory. To commit changes to file call WriteToFile().	
Parameters	const char* const ModeName:	The name of the mode for which an acquisition parameter is to be edited.
	const char* const ModeParam:	The name of the acquisition parameter to be edited.
	char* StringValue:	The value to assign to the acquisition parameter.
	const unsigned int StringLen:	The length of the input string.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null mode name.
	DRV_P2INVALID	Null mode parameter.
	DRV_P3INVALID	Null string value.
	DRV_P4INVALID	Invalid string length
	DRV_OA_INVALID_STRING_LENGTH	One or more of the string parameters has an invalid length, i.e. > 255.
	DRV_OA_MODE_DOES_NOT_EXIST	The Mode does not exist.
See also	OA_GetString, OA_EnableMode, OA_WriteToFile	

OA_WriteToFile

unsigned int WINAPI OA_WriteToFile (const char * const FileName , unsigned int FileNameLen)

Description	This function will write a User defined list of modes to the User file. The Preset file will not be affected.	
Parameters	const char* const FileName:	The name of the file to be written to.
	unsigned int FileNameLen:	File name string length.
Return	unsigned int	
	DRV_SUCCESS	All parameters accepted.
	DRV_P1INVALID	Null filename
	DRV_OA_INVALID_STRING_LENGTH	One or more of the string parameters has an invalid length, i.e. > 255.
	DRV_OA_INVALID_FILE	Data cannot be written to the Preset Andor file.
	DRV_ERROR_FILESAVE	Failed to save data to file.
	DRV_OA_FILE_HAS_BEEN_MODIFIED	File to be written to has been modified since last write, local copy of file may not be the same.
	DRV_OA_INVALID_CHARS_IN_NAME	File name contains invalid characters.
See also	OA_AddMode, OA_DeleteMode	

OutAuxPort

unsigned int WINAPI OutAuxPort(int port, int state)

Description This function sets the TTL Auxiliary Output port (P) on the Andor plug-in card to either ON/HIGH or OFF/LOW.

Parameters int port: Number of AUX out port on Andor card

Valid Values 1 to 4

int state: state to put port in

0	OFF/LOW
all others	ON/HIGH

Return unsigned int

DRV_SUCCESS	AUX port set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid port id.

See also InAuxPort

PrepareAcquisition

unsigned int WINAPI PrepareAcquisition(void)

Description This function reads the current acquisition setup and allocates and configures any memory that will be used during the acquisition. The function call is not required as it will be called automatically by the StartAcquisition function if it has not already been called externally.

However for long kinetic series acquisitions the time to allocate and configure any memory can be quite long which can result in a long delay between calling StartAcquisition and the acquisition actually commencing. For iDus, there is an additional delay caused by the camera being set-up with any new acquisition parameters. Calling PrepareAcquisition first will reduce this delay in the StartAcquisition call.

Parameters NONE

Return unsigned int

DRV_SUCCESS	Acquisition prepared.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_INIERROR	Error reading “DETECTOR.INI”.
DRV_ACQERROR	Acquisition settings invalid.
DRV_ERROR_PAGELOCK	Unable to allocate memory.
DRV_INVALID_FILTER	Filter not available for current acquisition.
DRV_IOCERROR	Integrate On Chip setup error.
DRV_BINNING_ERROR	Range not multiple of horizontal binning.

See also StartAcquisition, FreeInternalMemory,

PostProcessCountConvert

unsigned int WINAPI PostProcessCountConvert(at_32 * InputImage, at_32 * OutputImage, int OutputBufferSize, int NumImages, int Baseline, int Mode, int EmGain, float QE, float Sensitivity, int Height, int Width)

Description This function will convert the input image data to either Photons or Electrons based on the mode selected by the user. The input data should be in counts.

Parameters

at32* InputImage:	The input image data to be processed.
at32* OutputImage:	The output buffer to return the processed image.
int OutputBufferSize:	The size of the output buffer.
int NumImages:	The number of images if a kinetic series is supplied as the input data.
int Baseline:	The baseline associated with the image.
int Mode:	The mode to use to process the data. Valid options are: 1 – Convert to Electrons 2 – Convert to Photons
int EmGain:	The gain level of the input image.
float QE:	The Quantum Efficiency of the sensor.
float Sensitivity:	The Sensitivity value used to acquire the image.
int Height:	The height of the image.
int Width:	The width of the image.

Return unsigned int

DRV_SUCCESS	Acquisition prepared.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Number of images less than zero.
DRV_P5INVALID	Baseline less than zero.
DRV_P6INVALID	Invalid count convert mode.
DRV_P7INVALID	EMGain less than zero.
DRV_P8INVALID	QE less than zero.
DRV_P9INVALID	Sensitivity less than zero.
DRV_P10INVALID	Height less than zero.
DRV_P11INVALID	Width less than zero.
DRV_ERROR_BUFFSIZE	Output buffer size too small.

See also

PostProcessNoiseFilter

unsigned int WINAPI PostProcessNoiseFilter(at_32 * InputImage, at_32 * OutputImage, int OutputBufferSize, int Baseline, int Mode, float Threshold, int Height, int Width)

Description

This function will apply a filter to the input image and return the processed image in the output buffer. The filter applied is chosen by the user by setting Mode to a permitted value.

Parameters

at32* InputImage:	The input image data to be processed.
at32* OutputImage:	The output buffer to return the processed image.
int OutputBufferSize:	The size of the output buffer.
int Baseline:	The baseline associated with the image.
int Mode:	The mode to use to process the data. Valid options are: 1 – Use Median Filter. 2 – Use Level Above Filter. 3 – Use Interquartile Range Filter. 4 – Use Noise Threshold Filter.
float Threshold:	This is the Threshold multiplier for the Median, Interquartile and Noise Threshold filters. For the Level Above filter this is Threshold count above the baseline.
int Height:	The height of the image.
int Width:	The width of the image.

Return

unsigned int

DRV_SUCCESS	Acquisition prepared.
DRV_NOT_SUPPORTED	Camera does not support Noise filter processing.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Baseline less than zero.
DRV_P5INVALID	Invalid Filter mode.
DRV_P6INVALID	Threshold value not valid for selected mode.
DRV_P7INVALID	Height less than zero.
DRV_P8INVALID	Width less than zero.
DRV_ERROR_BUFFSIZE	Output buffer size too small.

See also

PostProcessPhotonCounting

unsigned int WINAPI PostProcessPhotonCounting(at_32 * InputImage, at_32 * OutputImage, int OutputBufferSize, int NumImages, int NumFrames, int NumberOfThresholds, float * Threshold, int Height, int Width)

Description This function will convert the input image data to photons and return the processed image in the output buffer.

Parameters

at32* InputImage:	The input image data to be processed.
at32* OutputImage:	The output buffer to return the processed image.
int OutputBufferSize:	The size of the output buffer.
int NumImages:	The number of images if a kinetic series is supplied as the input data.
int NumFrames:	The number of frames per output image.
int NumberOfThresholds:	The number of thresholds provided by the user.
float * Threshold:	The Thresholds used to define a photon.
int Height:	The height of the image.
int Width:	The width of the image.

Return unsigned int

DRV_SUCCESS	Acquisition prepared.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Number of images less than zero.
DRV_P5INVALID	Invalid Number of Frames requested.
DRV_P6INVALID	Invalid number of thresholds.
DRV_P7INVALID	Invalid pointer (i.e. NULL).
DRV_P8INVALID	Height less than zero.
DRV_P9INVALID	Width less than zero.
DRV_ERROR_BUFFSIZE	Output buffer size too small.

See also

SaveAsBmp

unsigned int WINAPI SaveAsBmp(char* path, char* palette, long ymin, long ymax)

Description This function saves the last acquisition as a bitmap file, which can be loaded into an imaging package. The palette parameter specifies the location of a **.PAL** file, which describes the colors to use in the bitmap. This file consists of **256 lines** of **ASCII text**; each line containing three numbers separated by spaces indicating the red, green and blue component of the respective color value.

The **ymin** and **ymax** parameters indicate which data values will map to the first and last colors in the palette:

- All data values below or equal to ymin will be colored with the first color.
- All values above or equal to ymax will be colored with the last color
- All other palette colors will be scaled across values between these limits.

Parameters char* path: The filename of the bitmap.
char* palette: The filename of a palette file (.PAL) for applying color to the bitmap.
long ymin, long ymax: Range of data values that palette will be scaled across. If set to 0, 0 the palette will scale across the full range of values.

Return unsigned int

DRV_SUCCESS	Data successfully saved as bitmap.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsSif SaveAsEDF SaveAsFITS SaveAsRaw SaveAsSPC SaveAsTiff

NOTE: If the last acquisition was in Kinetic Series mode, each image will be saved in a separate Bitmap file. The filename specified will have an index number appended to it, indicating the position in the series.

SaveAsCommentedSif

unsigned int WINAPI SaveAsCommentedSif(char* path, char* comment)

Description This function will save the data from the last acquisition into a file. The comment text will be added to the user text portion of the Sif file.

Parameters char* path: pointer to a filename specified by the user.
char* comment: comment text to add to the sif file

Return unsigned int

DRV_SUCCESS	Data saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid filename.

See also SetSifComment SaveAsSif SaveAsEDF SaveAsFITS SaveAsRaw SaveAsSPC SaveAsTiff SaveAsBmp

NOTE: The comment used in SIF files created with this function is discarded once the call completes, i.e. future calls to SaveAsSif will not use this comment. To set a persistent comment use the SetSifComment function.

SaveAsEDF

unsigned int WINAPI SaveAsEDF (char* szPath, int iMode)

Description This function saves the last acquisition in the European Synchotron Radiation Facility Data Format (*.edf).

Parameters char* szPath: the filename to save too.
int iMode: option to save to multiple files.
Valid values: 0 Save to 1 file
1 Save kinetic series to multiple files

Return unsigned int

DRV_SUCCESS	Data successfully saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_P2INVALID	Invalid mode
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsSif SaveAsFITS SaveAsRaw SaveAsSPC SaveAsTiff SaveAsBmp

---

SaveAsFITS

unsigned int WINAPI SaveAsFITS (char* szFileTitle, int typ)

Description This function saves the last acquisition in the FITS (Flexible Image Transport System) Data Format (*.fits) endorsed by NASA.

Parameters char* szFileTitle: the filename to save too.
int typ:
Valid values: 0 Unsigned 16
1 Unsigned 32
2 Signed 16
3 Signed 32
4 Float

Return unsigned int

DRV_SUCCESS	Data successfully saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_P2INVALID	Invalid mode
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsSif SaveAsEDF SaveAsRaw SaveAsSPC SaveAsTiff SaveAsBmp

SaveAsRaw

unsigned int WINAPI SaveAsRaw(char* szFileTitle, int typ)

Description This function saves the last acquisition as a raw data file.

Parameters char* szFileTitle: the filename to save too.
int typ:
Valid values: 1 Signed 16
2 Signed 32
3 Float

Return unsigned int

DRV_SUCCESS	Data successfully saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_P2INVALID	Invalid mode
DRV_ERROR_PAGELOCK	File too large to be generated in memory

See also SaveAsSif SaveAsEDF SaveAsFITS SaveAsSPC SaveAsTiff SaveAsBmp

SaveAsSif

unsigned int WINAPI SaveAsSif(char* path)

Description	This function will save the data from the last acquisition into a file, which can be read in by the main application. User text can be added to sif files using the SaveAsCommentedSif and SetSifComment functions.
Parameters	char* path: pointer to a filename specified by the user.
Return	unsigned int

DRV_SUCCESS	Data saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid filename.
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsEDF SaveAsFITS SaveAsRaw SaveAsSPC SaveAsTiff SaveAsBmp SetSifComment, SaveAsCommentedSif

SaveAsSPC

unsigned int WINAPI SaveAsSPC (char* path)

Description This function saves the last acquisition in the GRAMS .spc file format

Parameters char* path: the filename to save too.

Return unsigned int

DRV_SUCCESS	Data successfully saved.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsSif SaveAsEDF SaveAsFITS SaveAsRaw SaveAsTiff SaveAsBmp

SaveAsTiff

unsigned int WINAPI SaveAsTiff(char* path, char* palette, int position, int typ)

Description This function saves the last acquisition as a tiff file, which can be loaded into an imaging package. The palette parameter specifies the location of a .PAL file, which describes the colors to use in the tiff. This file consists of **256 lines** of **ASCII text**; each line containing three numbers separated by spaces indicating the red, green and blue component of the respective color value.

The parameter position can be changed to export different scans in a kinetic series. If the acquisition is any other mode, position should be set to 1. The parameter typ can be set to 0, 1 or 2 which correspond to 8-bit, 16-bit and color, respectively

Parameters char* path: The filename of the tiff.
char* palette: The filename of a palette file (.PAL) for applying color to the tiff.
int position: The number in the series, should be 1 for a single scan.
int typ: The type of tiff file to create.

Return unsigned int

DRV_SUCCESS	Data successfully saved as tiff.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_P2INVALID	Invalid palette file
DRV_P3INVALID	position out of range
DRV_P4INVALID	type not valid
DRV_ERROR_PAGELOCK	File too large to be generated in memory.

See also SaveAsSif SaveAsEDF SaveAsFITS SaveAsRaw SaveAsSPC SaveAsBmp SaveAsTiffEx

SaveAsTiffEx

unsigned int WINAPI SaveAsTiffEx(char* path, char* palette, int position, int typ, int mode)

Description This function saves the last acquisition as a tiff file, which can be loaded into an imaging package. This is an extended version of the SaveAsTiff function. The palette parameter specifies the location of a .PAL file, which describes the colors to use in the tiff. This file consists of 256 lines of ASCII text; each line containing three numbers separated by spaces indicating the red, green and blue component of the respective color value. The parameter position can be changed to export different scans in a kinetic series. If the acquisition is any other mode, position should be set to 1. The parameter typ can be set to 0, 1 or 2 which correspond to 8-bit, 16-bit and color, respectively. The mode parameter specifies the mode of output. Data can be output scaled from the min and max count values across the entire range of values (mode 0) or can remain unchanged (mode 1).Of course if the count value is higher or lower than the output data range then even in mode 1 data will be scaled.

Parameters

char* path: The filename of the tiff.
char* palette: The filename of a palette file (.PAL) for applying color to the tiff.
int position: The number in the series, should be 1 for a single scan.
int typ: The type of tiff file to create.
int mode: The output mode

Return unsigned int

DRV_SUCCESS	Data successfully saved as tiff
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Path invalid.
DRV_P2INVALID	Invalid palette file
DRV_P3INVALID	position out of range
DRV_P4INVALIDDRV_P5INVALID	type not validmode not valid
DRV_ERROR_PAGELOCK	File too large to be generated in memory

See also SaveAsSif SaveAsEDF SaveAsFITS SaveAsRaw SaveAsSPC SaveAsTiff SaveAsBmp

SaveEEPROMToFile

unsigned int WINAPI SaveEEPROMToFile(char *cFileName)

Description **THIS FUNCTION IS RESERVED.**

SaveToClipBoard

unsigned int WINAPI SaveToClipBoard(char* palette)

Description **THIS FUNCTION IS RESERVED.**

SelectDevice

unsigned int WINAPI SelectDevice(int devNum)

Description **THIS FUNCTION IS RESERVED.**

SendSoftwareTrigger

unsigned int WINAPI SendSoftwareTrigger ()

Description This function sends an event to the camera to take an acquisition when in Software Trigger mode. Not all cameras have this mode available to them. To check if your camera can operate in this mode check the GetCapabilities function for the Trigger Mode AC_TRIGGERMODE_CONTINUOUS. If this mode is physically possible and other settings are suitable (IsTriggerModeAvailable) and the camera is acquiring then this command will take an acquisition.

Parameters NONE

Return unsigned int

DRV_SUCCESS	Trigger sent
DRV_NOT_INITIALIZED	System not initialized
DRV_INVALID_MODE	Not in SoftwareTrigger mode
DRV_IDLE	Not Acquiring
DRV_ERROR_CODES	Error communicating with camera
DRV_ERROR_ACK	Previous acquisition not complete

See also GetCapabilities, IsTriggerModeAvailable, SetAcquisitionMode, SetReadMode, SetTriggerMode

NOTES **The settings of the camera must be as follows:**
ReadOut mode is full image
RunMode is Run Till Abort
TriggerMode is 10

SetAccumulationCycleTime

unsigned int WINAPI SetAccumulationCycleTime(float time)

Description This function will set the accumulation cycle time to the nearest valid value not less than the given value. The actual cycle time used is obtained by GetAcquisitionTimings. Please refer to SECTION 5 – ACQUISITION MODES for further information.

Parameters float time: the accumulation cycle time in seconds.

Return unsigned int

DRV_SUCCESS	Cycle time accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Exposure time invalid.

See also SetNumberAccumulations, GetAcquisitionTimings

SetAcqStatusEvent

unsigned int WINAPI SetAcqStatusEvent(HANDLE statusEvent)

Description This function passes a Win32 Event handle to the driver via which the driver can inform the user software that the camera has started exposing or that the camera has finished exposing. To determine what event has actually occurred call the GetCameraEventStatus funtion. This may give the user software an opportunity to perform other actions that will not affect the readout of the current acquisition. The SetPCIMode function must be called to enable/disable the events from the driver.

Parameters HANDLE statusEvent: Win32 event handle.

Return unsigned int

DRV_SUCCESS	Mode set
DRV_NOT_INITIALIZED	System not initialized
DRV_NOT_SUPPORTED	Function not supported for operating system

See also GetCameraEventStatus SetPCIMode

NOTE **This is only available with the CCI23 PCI card.**

SetAcquisitionMode

unsigned int WINAPI SetAcquisitionMode(int mode)

Description This function will set the acquisition mode to be used on the next StartAcquisition.

Parameters int mode: the acquisition mode.
Valid values:

1	Single Scan
2	Accumulate
3	Kinetics
4	Fast Kinetics
5	Run till abort

Return unsigned int

DRV_SUCCESS	Acquisition mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Acquisition Mode invalid.

See also StartAcquisition

NOTE: In Mode 5 the system uses a "Run Till Abort" acquisition mode. In Mode 5 only, the camera continually acquires data until the AbortAcquisition function is called. By using the SetDriverEvent function you will be notified as each acquisition is completed.

SetAcquisitionType

unsigned int WINAPI SetAcquisitionType (int typ)

Description **THIS FUNCTION IS RESERVED.**

SetADChannel

unsigned int WINAPI SetADChannel(int channel)

Description This function will set the AD channel to one of the possible A-Ds of the system. This AD channel will be used for all subsequent operations performed by the system.

Parameters int index: the channel to be used
Valid values: 0 to GetNumberADChannels-1

Return unsigned int

DRV_SUCCESS	AD channel set.
DRV_P1INVALID	Index is out off range.

See also GetNumberADChannels

SetAdvancedTriggerModeState

unsigned int WINAPI SetAdvancedTriggerModeState (int iState)

Description	This function will set the state for the iCam functionality that some cameras are capable of. There may be some cases where we wish to prevent the software using the new functionality and just do it the way it was previously done.
Parameters	int iState: 0: turn off iCam 1: Enable iCam.
Return	unsigned int

DRV_SUCCESS	State set
DRV_NOT_INITIALIZED	System not initialized
DRV_P1INVALID	state invalid

See also	iCam
NOTE	**By default the advanced trigger functionality is enabled.**

SetBackground

unsigned int WINAPI SetBackground(at_32* arr, unsigned long size)

Description **THIS FUNCTION IS RESERVED.**

SetBaselineClamp

unsigned int WINAPI SetBaselineClamp(int state)

Description This function turns on and off the baseline clamp functionality. With this feature enabled the baseline level of each scan in a kinetic series will be more consistent across the sequence.

Parameters int state: Enables/Disables Baseline clamp functionality

1 – Enable Baseline Clamp

0 – Disable Baseline Clamp

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_SUPPORTED	Baseline Clamp not supported on this camera
DRV_P1INVALID	State parameter was not zero or one.

SetBaselineOffset

unsigned int WINAPI SetBaselineOffset(int offset)

Description This function allows the user to move the baseline level by the amount selected. For example **"+100"** will add approximately 100 counts to the default baseline value. The value entered should be a multiple of 100 between -1000 and +1000 inclusively.

Parameters Int offset: Amount to offset baseline by

Return unsigned int

DRV_SUCCESS	Parameters set
DRV_NOT_INITIALIZED	System not initialized
DRV_NOT_AVAILABLE	Baseline Clamp not available for this camera
DRV_ACQUIRING	Acquisition in progress
DRV_P1INVALID	Offset out of range

NOTE Only available on iXon range

SetCameraStatusEnable

unsigned int WINAPI SetCameraStatusEnable(DWORD Enable)

Description Use this function to Mask out certain types of acquisition status events. The default is to notify on every type of event but this may cause missed events if different types of event occur very close together. The bits in the mask correspond to the following event types:

0 – Fire pulse down event

1 – Fire pulse up event

Set the corresponding bit to 0 to disable the event type and 1 to enable the event type.

Parameters DWORD Enable: bitmask with bits set for those events about which you wish to be notified.

Return unsigned int

DRV_SUCCESS	Mask Set.
DRV_VXDNOTINSTALLED	Device Driver not installed.

See also SetAcqStatusEvent SetPCIMode

NOTE Only available with PCI systems using the CCI-23 controller card

Fire pulse up event not available on USB systems.

SetComplexImage

unsigned int WINAPI SetComplexImage(int numAreas, int* areas)

Description

This is a function that allows the setting up of random tracks with more options that the SetRandomTracks function.

The minimum number of tracks is 1. The maximum number of tracks is the number of vertical pixels.

There is a further limit to the number of tracks that can be set due to memory constraints in the camera. It is not a fixed number but depends upon the combinations of the tracks. For example, 20 tracks of different heights will take up more memory than 20 tracks of the same height.

If attempting to set a series of random tracks and the return code equals DRV_RANDOM_TRACK_ERROR, change the makeup of the tracks to have more repeating heights and gaps so less memory is needed.

Each track must be defined by a group of six integers.

-The top and bottom positions of the tracks.
-The left and right positions for the area of interest within each track
-The horizontal and vertical binning for each track.

The positions of the tracks are validated to ensure that the tracks are in increasing order.

The left and right positions for each track must be the same.
For iXon the range is between 8 and CCD width, inclusive
For idus the range must be between 257 and CCD width, inclusive.

Horizontal binning must be an integer between 1 and 64 inclusive, for iXon.
Horizontal binning is not implementated for iDus and must be set to 1.
Vertical binning is used in the following way. A track of:
1 10 1 1024 1 2

is actually implemented as 5 tracks of height 2. . Note that a vertical binning of 1 will have the effect of vertically binning the entire track; otherwise vertical binning will operate as normal.
1 2 1 1024 1 1
3 4 1 1024 1 1
5 6 1 1024 1 1
7 8 1 1024 1 1
9 10 1 1024 1 1

Parameters

int numAreas:

int * areas:

Return

Unsigned int

DRV_SUCCESS	Success
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number of tracks invalid.
DRV_P2INVALID	Track positions invalid.
DRV_ERROR_FILELOAD	Serious internal error
DRV_RANDOM_TRACK_ERROR	Invalid combination of tracks, out of memory or mode not available.

See also SetRandomTracks

NOTE **Only available with iXon+ and USB cameras.**

SetCoolerMode

unsigned int WINAPI SetCoolerMode(int mode)

Description This function determines whether the cooler is switched off when the camera is shut down.

Parameters int mode:

1 – Temperature is maintained on ShutDown

0 – Returns to ambient temperature on ShutDown

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	State parameter was not zero or one.
DRV_NOT_SUPPORTED	Camera does not support

NOTE: Mode 0 not available on Luca R cameras – always cooled to -20.

SetCountConvertMode

unsigned int WINAPI SetCountConvertMode(int mode)

Description This function configures the Count Convert mode.

Parameters int mode:
0 – Data in Counts
1 – Data in Electrons
2 – Data in Photons

Return unsigned int

DRV_SUCCESS	Count Convert mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Count Convert not available for this camera
DRV_P1INVALID	Mode parameter was out of range.

See also GetCapabilities, SetCountConvertWavelength

SetCountConvertWavelength

unsigned int WINAPI SetCountConvertWavelength(float wavelength)

Description This function configures the wavelength used in Count Convert mode.

Parameters float wavelength: wavelength used to determine QE

Return unsigned int

DRV_SUCCESS	Count Convert wavelength set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Count Convert not available for this camera
DRV_P1INVALID	Wavelength value was out of range.

See also GetCapabilities, SetCountConvertMode

SetCropMode

unsigned int WINAPI SetCropMode (int active, int cropHeight, int reserved)

Description This function effectively reduces the height of the CCD by excluding some rows to achieve higher frame rates. This is currently only available on Newton cameras when the selected read mode is Full Vertical Binning. The cropHeight is the number of active rows measured from the bottom of the CCD.

Note: it is important to ensure that no light falls on the excluded region otherwise the acquired data will be corrupted.

Parameters

int active: 1 - Crop mode is ON
0 – Crop mode is OFF

int cropHeight: The selected crop height. This value must be between 1 and the CCD height

int reserved: This value should be set to 0.

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIAILIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Active parameter is not zero or one.
DRV_P2INVALID	Cropheight parameter is less than one or greater than the CCD height.
DRV_P3INVALID	Reserved parameter is not equal to zero.
DRV_NOT_SUPPORTED	Either the camera is not a Newton or the read mode is not Full Vertical Binning.

See also GetDetector SetIsolatedCropMode

NOTE : Available on Newton

SetCurrentCamera

unsigned int WINAPI SetCurrentCamera(long cameraHandle)

Description When multiple Andor cameras are installed this function allows the user to select which camera is currently active. Once a camera has been selected the other functions can be called as normal but they will only apply to the selected camera. If only 1 camera is installed calling this function is not required since that camera will be selected by default.

Parameters long cameraHandle: Selects the active camera

Return unsigned int

DRV_SUCCESS	Camera successfully selected.
DRV_P1INVALID	Invalid camera handle.

SEE ALSO : GetCurrentCamera, GetAvailableCameras, GetCameraHandle

SetCustomTrackHBin

unsigned int WINAPI SetCustomTrackHBin(int bin)

Description This function sets the horizontal binning value to be used when the readout mode is set to Random Track.

Parameters Int bin: Binning size.

Return unsigned int

DRV_SUCCESS	Binning set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid binning size.

See also SetReadMode

NOTE: For iDus, it is recommended that you set horizontal binning to 1

SetDACOutputScale

unsigned int WINAPI SetDACOutputScale(int scale)

Description Clara offers 2 configurable precision 16-bit DAC outputs. This function should be used to select the active one.

Parameters int scale: 5 or 10 volt DAC range (1/2).

Return unsigned int

DRV_SUCCESS	DAC Scale option accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Feature not available
DRV_P1INVALID	DAC Scale value invalid.

See also SetDACOutput

NOTE: Only available on Andor Clara

SetDACOutput

unsigned int WINAPI SetDACOutput(int option, int resolution, int value)

Description Clara offers 2 configurable precision 16-bit DAC outputs. This function should be used to set the required voltage.

Parameters int option: DAC Output Scale 1 or 2 (1/2).
int resolution: resolution of DAC can be set from 2 to 16-bit in steps of 2
int value: requested DAC value (for particular resolution)

Return unsigned int

DRV_SUCCESS	DAC Scale option accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_NOT_AVAILABLE	Feature not available.
DRV_P1INVALID	DAC range value invalid.
DRV_P2INVALID	Resolution unavailable.
DRV_P3INVALID	Requested value not within DAC range.

See also SetDACOutputScale

NOTE: Only available on Andor Clara

SetDataType

unsigned int WINAPI SetDataType (int typ)

Description **THIS FUNCTION IS RESERVED.**

SetDDGAddress

unsigned int WINAPI SetDDGAddress(BYTE t0, BYTE t1, BYTE t2, BYTE t3, BYTE address)

Description **THIS FUNCTION IS RESERVED.**

SetDDGGain

unsigned int WINAPI SetDDGGain(int gain)

Description Deprecated for SetMCPGain.

SetDDGGateStep

unsigned int WINAPI SetDDGGateStep(double step_Renamed)

Description This function will set a constant value for the gate step in a kinetic series. The lowest available resolution is 25 picoseconds and the maximum permitted value is 25 seconds.

Parameters double step_Renamed: gate step in picoseconds.

Return unsigned int

DRV_SUCCESS	Gate step set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Gate step invalid.

See also SetDDGTimes, SetDDGVariableGateStep

NOTE: Available on iStar.

SetDDGInsertionDelay

unsigned int WINAPI SetDDGInsertionDelay(int state)

Description This function controls the length of the insertion delay.

Parameters int state: NORMAL/FAST switch for insertion delay.

Valid values: 0 to set normal insertion delay.
1 to set fast insertion delay.

Return unsigned int

DRV_SUCCESS	Value for delay accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.

See also SetDDGIntelligate

NOTE: Available on iStar.

SetDDGIntelligate

unsigned int WINAPI SetDDGIntelligate(int state)

Description This function controls the MCP gating. Not available when the fast insertion delay option is selected.

Parameters int state: ON/OFF switch for the MCP gating.
Valid values: 0 to switch MCP gating OFF.
1 to switch MCP gating ON.

Return unsigned int

DRV_SUCCESS	Intelligate option accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.

See also SetDDGInsertionDelay

NOTE: Available on iStar.

SetDDGIOC

unsigned int WINAPI SetDDGIOC(int state)

Description	This function activates the integrate on chip (IOC) option.	
Parameters	int integrate: ON/OFF switch for the IOC option. Valid values: 0 to switch IOC OFF. 1 to switch IOC ON.	
Return	unsigned int	
	DRV_SUCCESS	IOC option accepted.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_I2CTIMEOUT	I2C command timed out.
	DRV_I2CDEVNOTFOUND	I2C device not present.
	DRV_ERROR_ACK	Unable to communicate with card.
See also	SetDDGIOCFrequency GetDDGIOCFrequency SetDDGIOCNumber GetDDGIOCNumber GetDDGIOCPulses	

NOTE: Available on iStar.

SetDDGIOCFrequency

unsigned int WINAPI SetDDGIOCFrequency(double frequency)

Description This function sets the frequency of the integrate on chip option. It should be called once the conditions of the experiment have been setup in order for correct operation. The frequency should be limited to 5000Hz when Intelligate is activated to prevent damage to the head and 50000Hz otherwise to prevent the gater from overheating. The recommended order is

…

Experiment setup (exposure time, readout mode, gate parameters, …)

…

SetDDGIOCFrequency (x)

SetDDGIOC(true)

GetDDGIOCPulses(y)

StartAcquisition()

Parameters double frequency: frequency of IOC option in Hz.

Return unsigned int

DRV_SUCCESS	Value for frequency accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.

See also GetDDGIOCFrequency SetDDGIOCNumber GetDDGIOCNumber GetDDGIOCPulses SetDDGIOC

NOTE: Available on iStar.

SetDDGIOCNumber

unsigned int WINAPI SetDDGIOCNumber(unsigned long numberPulses)

Description This function allows the user to limit the number of pulses used in the integrate on chip option at a given frequency. It should be called once the conditions of the experiment have been setup in order for correct operation.

Parameters unsigned long numberPulses: the number of integrate on chip pulses triggered within the fire pulse.

Return unsigned int

DRV_SUCCESS	Value for IOC number accepted
DRV_NOT_INITIALIZED	System not initialized
DRV_ACQUIRING	Acquisition in progress
DRV_I2CTIMEOUT	I2C command timed out
DRV_I2CDEVNOTFOUND	I2C device not present
DRV_ERROR_ACK	Unable to communicate with card

See also SetDDGIOCFrequency GetDDGIOCFrequency GetDDGIOCNumber GetDDGIOCPulses SetDDGIOC

NOTE: Available on iStar.

SetDDGTimes

unsigned int WINAPI SetDDGTimes(double t0, double t1, double t2)

Description This function sets the properties of the gate pulse. t0 has a resolution of 16 nanoseconds whilst t1 and t2 have a resolution of 25 picoseconds.

Parameters double t0: output A delay in nanoseconds.
double t1: gate delay in picoseconds.
double t2: pulse width in picoseconds.

Return unsigned int

DRV_SUCCESS	Values for gate pulse accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.
P1_INVALID	Invalid output A delay.
P2_INVALID	Invalid gate delay.
P3_INVALID	Invalid pulse width.

See also SetDDGGateStep

NOTE: Available on iStar.

SetDDGTriggerMode

unsigned int WINAPI SetDDGTriggerMode(int mode)

Description This function will set the trigger mode of the internal delay generator to either Internal or External

Parameters int mode: trigger mode

Valid values:

0	Internal
1	External

Return unsigned int

DRV_SUCCESS	Trigger mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Trigger mode invalid.

NOTE: Available on iStar.

SetDDGVariableGateStep

unsigned int WINAPI SetDDGVariableGateStep(int mode, double p1, double p2)

Description This function will set a varying value for the gate step in a kinetic series. The lowest available resolution is 25 picoseconds and the maximum permitted value is 25 seconds.

Parameters int mode: the gate step mode.

Valid values:

1	Exponential (p1*exp(p2*n))
2	Logarithmic (p1*log(p2*n))
3	Linear (p1 + p2*n)

n = 1, 2, …, number in kinetic series

Return unsigned int

DRV_SUCCESS	Gate step mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Gate step mode invalid.

See also StartAcquisition

NOTE: Available on iStar.

SetDelayGenerator

unsigned int WINAPI SetDelayGenerator(int board, short address, int typ)

Description This function sets parameters to control the delay generator through the GPIB card in your computer.

Parameters int board: The GPIB board number of the card used to interface with the Delay Generator.
short address: The number that allows the GPIB board to identify and send commands to the delay generator.
Int typ: The type of your Delay Generator.

Return unsigned int

DRV_SUCCESS	Delay Generator set up.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	GPIB board invalid.
DRV_P2INVALID	GPIB address invalid
DRV_P3INVALID	Delay generator type invalid.

See also SetGate

NOTE: Available on ICCD.

SetDMAParameters

unsigned int WINAPI SetDMAParameters(int MaxImagesPerDMA, float SecondsPerDMA)

Description

In order to facilitate high image readout rates the controller card may wait for multiple images to be acquired before notifying the SDK that new data is available. Without this facility, there is a chance that hardware interrupts may be lost as the operating system does not have enough time to respond to each interrupt. The drawback to this is that you will not get the data for an image until all images for that interrupt have been acquired.

There are 3 settings involved in determining how many images will be acquired for each notification (DMA Interrupt) of the controller card and they are as follows:

1. The size of the DMA buffer gives an upper limit on the number of images that can be stored within it and is usually set to the size of one full image when installing the software. This will usually mean that if you acquire full frames there will never be more than one image per DMA.
2. A second setting that is used is the maximum amount of time(SecondsPerDMA) that should expire between interrupts. This can be used to give an indication of the reponsiveness of the operating system to interrupts. Decreasing this value will allow more interrupts per second and should only be done for faster pcs. The default value is 0.03s (30ms), finding the optimal value for your pc can only be done through experimentation.
3. The third setting is an overide to the number of images calculated using the previous settings. If the number of images per dma is calculated to be greater than MaxImagesPerDMA then it will be reduced to MaxImagesPerDMA. This can be used to, for example, ensure that there is never more than 1 image per DMA by setting MaxImagesPerDMA to 1. Setting MaxImagesPerDMA to zero removes this limit. Care should be taken when modifying these parameters as missed interrupts may prevent the acquisition from completing.

Parameters

int MaxImagesPerDMA: Override to the number of images per DMA if the calculated value is higher than this. (Default=0, ie. no override)
float SecondsPerDMA: Minimum amount of time to elapse between interrrupts. (Default=0.03s)

Return

unsigned int

DRV_SUCCESS	DMA Parameters setup successfully.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	MaxImagesPerDMA invalid
DRV_P2INVALID	SecondsPerDMA invalid

SetDriverEvent

unsigned int WINAPI SetDriverEvent(HANDLE driverEvent)

Description This function passes a Win32 Event handle to the SDK via which the the user software can be informed that something has occurred. For example the SDK can “set” the event when an acquisition has completed thus relieving the user code of having to continually pole to check on the status of the acquisition.
The event will be “set” under the follow conditions:
1) Acquisition completed or aborted.
2) As each scan during an acquisition is completed.
3) Temperature as stabilized, drifted from stabilization or could not be reached.
When an event is triggered the user software can then use other SDK functions to determine what actually happened.

Condition 1 and 2 can be tested via GetStatus function, while condition 3 checked via GetTemperature function.
You must reset the event after it has been handled in order to receive additional triggers.
Before deleting the event you must call SetDriverEvent with NULL as the parameter.

Parameters HANDLE driverEvent: Win32 event handle.

Return unsigned int

DRV_SUCCESS	Event set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Function not supported for operating system

See also GetStatus GetTemperature GetAcquisitionProgress

NOTE: Not all programming environments allow the use of multiple threads and WIN32 events.

SetDualExposureMode

unsigned int WINAPI SetDualExposureMode(int mode)

Description This function turns on and off the option to acquire 2 frames for each external trigger pulse. This mode is only available for certain sensors in run till abort mode, external trigger, full image.

Parameters int state: Enables/Disables dual exposure mode
1 – Enable mode
0 – Disable mode

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Dual exposure mode not supported on this camera.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Mode parameter was not zero or one.

See also GetCapabilities, SetDualExposureTimes, GetDualExposureTimes

---

SetDualExposureTimes

unsigned int WINAPI SetDualExposureTimes(float exposure1, float exposure2)

Description This function configures the two exposure times used in dual exposure mode. This mode is only available for certain sensors in run till abort mode, external trigger, full image.

Parameters float exposure1: the exposure time in seconds for each odd numbered frame.
float exposure2: the exposure time in seconds for each even numbered frame.

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Dual exposure mode not supported on this camera.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	First exposure out of range.
DRV_P2INVALID	Second exposure out of range.

See also GetCapabilities, SetDualExposureMode, GetDualExposureTimes

---

SetEMAdvanced

unsigned int WINAPI SetEMAdvanced(int state)

Description This function turns on and off access to higher EM gain levels within the SDK. Typically, optimal signal to noise ratio and dynamic range is achieved between x1 to x300 EM Gain. Higher gains of > x300 are recommended for single photon counting only. Before using higher levels, you should ensure that light levels do not exceed the regime of tens of photons per pixel, otherwise accelerated ageing of the sensor can occur.

Parameters int state: Enables/Disables access to higher EM gain levels
1 – Enable access
0 – Disable access

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_AVAILABLE	Advanced EM gain not available for this camera.
DRV_ACQUIRING.	Acquisition in progress.
DRV_P1INVALID	State parameter was not zero or one.

See also GetCapabilities, GetEMCCDGain, SetEMCCDGain, SetEMGainMode

---

SetEMCCDGain

unsigned int WINAPI SetEMCCDGain(int gain)

Description Allows the user to change the gain value. The valid range for the gain depends on what gain mode the camera is operating in. See SetEMGainMode to set the mode and GetEMGainRange to get the valid range to work with. To access higher gain values (>x300) see SetEMAdvanced.

Parameters int gain: amount of gain applied.

Return unsigned int

DRV_SUCCESS	Value for gain accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Gain value invalid.

See also GetEMCCDGain SetEMGainMode GetEMGainRange SetEMAdvanced

---

NOTE: Only available on EMCCD sensor systems.

SetEMClockCompensation

unsigned int WINAPI SetEMClockCompensation(int EMClockCompensationFlag)

Description **THIS FUNCTION IS RESERVED.**

SetEMGainMode

unsigned int WINAPI SetEMGainMode(int mode)

Description Set the EM Gain mode to one of the following possible settings.

Mode 0: The EM Gain is controlled by DAC settings in the range 0-255. Default mode.

1: The EM Gain is controlled by DAC settings in the range 0-4095.

2: Linear mode.

3: Real EM gain

To access higher gain values (if available) it is necessary to enable advanced EM gain, see SetEMAdvanced.

Parameters int mode: EM Gain mode.

Return

DRV_SUCCESS	Mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	EM Gain mode invalid.

SetExposureTime

unsigned int WINAPI SetExposureTime(float time)

Description This function will set the exposure time to the nearest valid value not less than the given value. The actual exposure time used is obtained by GetAcquisitionTimings. . Please refer to SECTION 5 – ACQUISITION MODES for further information.

Parameters float time: the exposure time in seconds.

Return unsigned int

DRV_SUCCESS	Exposure time accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Exposure Time invalid.

See also GetAcquisitionTimings

NOTE: For Classics, if the current acquisition mode is Single-Track, Multi-Track or Image then this function will actually set the Shutter Time. The actual exposure time used is obtained from the GetAcquisitionTimings function.

SetFanMode

unsigned int WINAPI SetFanMode(int mode)

Description Allows the user to control the mode of the camera fan. If the system is cooled, the fan should only be turned off for short periods of time. During this time the body of the camera will warm up which could compromise cooling capabilities.

If the camera body reaches too high a temperature, depends on camera, the buzzer will sound. If this happens, turn off the external power supply and allow the system to stabilize before continuing.

Parameters int mode: fan on full (0)
fan on low (1)
fan off (2)

Return unsigned int

DRV_SUCCESS	Value for mode accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I^2C command timed out.
DRV_I2CDEVNOTFOUND	I^2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Mode value invalid.

See also GetCapabilities

SetFastKinetics

unsigned int WINAPI SetFastKinetics(int exposedRows int seriesLength, float time, int mode, int hbin, int vbin)

Description This function will set the parameters to be used when taking a fast kinetics acquisition.

Parameters
int exposedRows: sub-area height in rows.
int seriesLength: number in series.
float time: exposure time in seconds.
int mode: binning mode (0 – FVB , 4 – Image).
int hbin: horizontal binning.
int vbin: vertical binning (only used when in image mode).

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid height.
DRV_P2INVALID	Invalid number in series.
DRV_P3INVALID	Exposure time must be greater than 0.
DRV_P4INVALID	Mode must be equal to 0 or 4.
DRV_P5INVALID	Horizontal binning.
DRV_P6INVALID	Vertical binning.

See also SetFKVShiftSpeed SetFastKineticsEx

NOTE: For classic cameras the vertical and horizontal binning must be 1
For iDus, it is recommended that you set horizontal binning to 1

SetFastKineticsEx

unsigned int WINAPI SetFastKineticsEx(int exposedRows, int seriesLength, float time, int mode, int hbin, int vbin, int offset)

Description This function is the same as SetFastKinetics with the addition of an Offset parameter, which will inform the SDK of the first row to be used.

Parameters int exposedRows: sub-area height in rows.
int seriesLength: number in series.
float time: exposure time in seconds.
int mode: binning mode (0 – FVB , 4 – Image).
int hbin: horizontal binning.
int vbin: vertical binning (only used when in image mode).
Int offset: offset of first row to be used in Fast Kinetics from the bottom of the CCD.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid height.
DRV_P2INVALID	Invalid number in series.
DRV_P3INVALID	Exposure time must be greater than 0.
DRV_P4INVALID	Mode must be equal to 0 or 4.
DRV_P5INVALID	Horizontal binning.
DRV_P6INVALID	Vertical binning.
DRV_P7INVALID	Offset not within CCD limits

See also SetFKVShiftSpeed SetFastKinetics

NOTE: For classic cameras the offset must be 0 and the vertical and horizontal binning must be 1
For iDus, it is recommended that you set horizontal binning to 1

SetFastExtTrigger

unsigned int WINAPI SetFastExtTrigger(int mode)

Description This function will enable fast external triggering. When fast external triggering is enabled the system will NOT wait until a “Keep Clean” cycle has been completed before accepting the next trigger. This setting will only have an effect if the trigger mode has been set to External via SetTriggerMode.

Parameters int mode:

0	Disabled
1	Enabled

Return unsigned int

DRV_SUCCESS	Parameters accepted.

See also SetTriggerMode

SetFilterMode

unsigned int WINAPI SetFilterMode(int mode)

Description This function will set the state of the cosmic ray filter mode for future acquisitions. If the filter mode is on, consecutive scans in an accumulation will be compared and any cosmic ray-like features that are only present in one scan will be replaced with a scaled version of the corresponding pixel value in the correct scan.

Parameters int mode: current state of filter

0	OFF
2	ON

Return unsigned int

DRV_SUCCESS	Filter mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Mode is out off range.

See also GetFilterMode

SetFilterParameters

unsigned int WINAPI SetFilterParameters (int width, float sensitivity, int range, float accept, int smooth, int noise)

Description **THIS FUNCTION IS RESERVED.**

SetFKVShiftSpeed

unsigned int WINAPI SetFKVShiftSpeed(int index)

Description This function will set the fast kinetics vertical shift speed to one of the possible speeds of the system. It will be used for subsequent acquisitions.

Parameters int index: the speed to be used

Valid values 0 to GetNumberFKVShiftSpeeds-1

Return unsigned int

DRV_SUCCESS	Fast kinetics vertical shift speed set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Index is out off range.

See also GetNumberFKVShiftSpeeds, GetFKVShiftSpeedF

NOTE: Only available if camera is Classic or iStar.

SetFPDP

unsigned int WINAPI SetFPDP(int state)

Description **THIS FUNCTION IS RESERVED.**

SetFrameTransferMode

unsigned int WINAPI SetFrameTransferMode (int mode)

Description This function will set whether an acquisition will readout in Frame Transfer Mode. If the acquisition mode is Single Scan or Fast Kinetics this call will have no affect.

Parameters int mode: mode

0 OFF

1 ON

Return unsigned int

DRV_SUCCESS	Frame transfer mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid parameter.

See also SetAcquisitionMode

NOTE: Only available if CCD is a Frame Transfer chip.

SetFullImage

unsigned int WINAPI SetFullImage(int hbin, int vbin)

Description **Deprecated see Note:**

This function will set the horizontal and vertical binning to be used when taking a full resolution image.

Parameters int hbin: number of pixels to bin horizontally
int vbin: number of pixels to bin vertically

Return unsigned int

DRV_SUCCESS	Binning parameters accepted
DRV_NOT_INITIALIZED	System not initialized
DRV_ACQUIRING	Acquisition in progress
DRV_P1INVALID	Horizontal binning parameter invalid
DRV_P2INVALID	Vertical binning parameter invalid

See also SetReadMode

NOTE: Deprecated by SetImage

SetFVBHBin

unsigned int WINAPI SetFVBHBin(int bin)

Description This function sets the horizontal binning used when acquiring in Full Vertical Binned read mode.

Parameters Int bin: Binning size.

Return unsigned int

DRV_SUCCESS	Binning set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid binning size.

See also SetReadMode

NOTE: 1) If the detector width is not a multiple of the binning DRV_BINNING_ERROR will be returned from PrepareAcquisition and/or StartAcquisition
2) For iDus, it is recommended that you set horizontal binning to 1

SetGain

unsigned int WINAPI SetGain(int gain)

Description Deprecated for SetMCPGain.

SetGate

unsigned int WINAPI SetGate(float delay, float width, float step_Renamed)

Description This function sets the Gater parameters for an ICCD system. The image intensifier of the Andor ICCD acts as a shutter on nanosecond time-scales using a process known as gating.

Parameters float delay: Sets the delay(>=0) between the T0 and C outputs on the SRS box to delay nanoseconds.
float width: Sets the width(>=0) of the gate in nanoseconds
float step_Renamed: Sets the amount(<>0, in nanoseconds) by which the gate position is moved in time after each scan in a kinetic series.

Return unsigned int

DRV_SUCCESS	Gater parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_ACQUIRING	Acquisition in progress.
DRV_GPIBERROR	Error communicating with GPIB card.
DRV_P1INVALID	Invalid delay
DRV_P2INVALID	Invalid width.
DRV_P3INVALID	Invalid step.

See also SetDelayGenerator

NOTE: Available on ICCD.

SetGateMode

unsigned int WINAPI SetGateMode(int gatemode)

Description	Allows the user to control the photocathode gating mode.		
Parameters	int gatemode: the gate mode.		
	Valid values:	0	Fire ANDed with the Gate input.
		1	Gating controlled from Fire pulse only.
		2	Gating controlled from SMB Gate input only.
		3	Gating ON continuously.
		4	Gating OFF continuously.
		5	Gate using DDG (**iStar only**).

Return	unsigned int	
	DRV_SUCCESS	Gating mode accepted.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_I2CTIMEOUT	I^2C command timed out.
	DRV_I2CDEVNOTFOUND	I^2C device not present.
	DRV_ERROR_ACK	Unable to communicate with card.
	DRV_P1INVALID	Gating mode invalid.

See also SetMCPGain, SetMCPGating

NOTE: Available on iStar.

SetHighCapacity

unsigned int WINAPI SetHighCapacity(int state)

Description This function switches between high sensitivity and high capacity functionality. With high capacity enabled the output amplifier is switched to a mode of operation which reduces the responsivity thus allowing the reading of larger charge packets during binning operations.

Parameters int state: Enables/Disables High Capacity functionality
1 – Enable High Capacity (Disable High Sensitivity)
0 – Disable High Capacity (Enable High Sensitivity)

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	State parameter was not zero or one.

See also GetCapabilities

SetHorizontalSpeed

unsigned int WINAPI SetHorizontalSpeed(int index)

Description **Deprecated see Note:**

This function will set the horizontal speed to one of the possible speeds of the system. It will be used for subsequent acquisitions.

Parameters int index: the horizontal speed to be used
Valid values 0 to GetNumberHorizontalSpeeds-1

Return unsigned int

DRV_SUCCESS	Horizontal speed set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Index is out off range.

See also GetNumberHorizontalSpeeds, GetHorizontalSpeed
NOTE: Deprecated by SetHSSpeed

SetHSSpeed

unsigned int WINAPI SetHSSpeed(int typ, int index)

Description This function will set the speed at which the pixels are shifted into the output node during the readout phase of an acquisition. Typically your camera will be capable of operating at several horizontal shift speeds. To get the actual speed that an index corresponds to use the GetHSSpeed function.

Parameters int typ: output amplification.

Valid values: 0 electron multiplication/Conventional(clara).
1 conventional/Extended NIR mode(clara).

int index: the horizontal speed to be used

Valid values 0 to GetNumberHSSpeeds()-1

Return unsigned int

DRV_SUCCESS	Horizontal speed set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Mode is invalid.
DRV_P2INVALID	Index is out off range.

See also GetNumberHSSpeeds, GetHSSpeed GetNumberAmp

SetImage

unsigned int WINAPI SetImage(int hbin, int vbin, int hstart, int hend, int vstart, int vend)

Description This function will set the horizontal and vertical binning to be used when taking a full resolution image.

Parameters int hbin: number of pixels to bin horizontally.
int vbin: number of pixels to bin vertically.
int hstart: Start column (inclusive).
int hend: End column (inclusive).
int vstart: Start row (inclusive).
int vend: End row (inclusive).

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Binning parameters invalid.
DRV_P2INVALID	Binning parameters invalid.
DRV_P3INVALID	Sub-area co-ordinate is invalid.
DRV_P4INVALID	Sub-area co-ordinate is invalid.
DRV_P5INVALID	Sub-area co-ordinate is invalid.
DRV_P6INVALID	Sub-area co-ordinate is invalid.

See also SetReadMode

NOTE: For iDus, it is recommended that you set horizontal binning to 1

SetImageFlip

unsigned int WINAPI SetImageFlip(int iHFlip, int iVFlip)

Description This function will cause data output from the SDK to be flipped on one or both axes. This flip is not done in the camera, it occurs after the data is retrieved and will increase processing overhead. If flipping could be implemented by the user more efficiently then use of this function is not recomended. E.g writing to file or displaying on screen.

Parameters int iHFlip: Sets horizontal flipping.
int iVFlip: Sets vertical flipping..

1 - Enables Flipping
0 - Disables Flipping

If this function is used in conjunction with the SetImageRotate function the rotation will occur before the flip regardless of which order the functions are called.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	HFlip parameter invalid.
DRV_P2INVALID	VFlip parameter invalid

See also SetImageRotate

SetImageRotate

unsigned int WINAPI SetImageRotate(int iRotate)

Description This function will cause data output from the SDK to be rotated on one or both axes. This rotate is not done in the camera, it occurs after the data is retrieved and will increase processing overhead. If the rotation could be implemented by the user more efficiently then use of this function is not recomended. E.g writing to file or displaying on screen.

Parameters int iRotate: Rotation setting

0 - No rotation
1 - Rotate 90 degrees clockwise
2 - Rotate 90 degrees anti-clockwise

If this function is used in conjunction with the SetImageFlip function the rotation will occur before the flip regardless of which order the functions are called.

180 degree rotation can be achieved using the SetImageFlip function by selecting both horizontal and vertical flipping.

Return unsigned int

DRV_SUCCESS	All parameters accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_P1INVALID	Rotate parameter invalid.

See also SetImageFlip

SetIsolatedCropMode

unsigned int WINAPI SetIsolatedCropMode(int active, int cropheight, int cropwidth, int vbin, int hbin)

Description This function effectively reduces the dimensions of the CCD by excluding some rows or columns to achieve higher throughput. In isolated crop mode iXon, Newton and iKon cameras can operate in either Full Vertical Binning or Imaging read modes. iDus can operate in Full Vertical Binning read mode only.

Note: It is important to ensure that no light falls on the excluded region otherwise the acquired data will be corrupted.

Parameters int active: 1 – Crop mode is ON.

0 – Crop mode is OFF.

int cropheight: The selected crop height. This value must be between 1 and the CCD height.

int cropwidth: The selected crop width. This value must be between 1 and the CCD width.

int vbin: The selected vertical binning.

int hbin: The selected horizontal binning.

Return unsigned int

DRV_SUCCESS	Parameters set
DRV_NOT_INITIALIZED	System not initialized
DRV_ACQUIRING	Acquisition in progress
DRV_P1INVALID	active parameter was not zero or one
DRV_P2INVALID	Invalid crop height
DRV_P3INVALID	Invalid crop width
DRV_P4INVALID	Invalid vertical binning
DRV_P5INVALID	Invalid horizontal binning
DRV_NOT_SUPPORTED	Either the camera does not support isolated Crop mode or the read mode is invalid

See also GetDetector SetReadMode

NOTE: For iDus, it is recommended that you set horizontal binning to 1

SetKineticCycleTime

unsigned int WINAPI SetKineticCycleTime(float time)

Description This function will set the kinetic cycle time to the nearest valid value not less than the given value. The actual time used is obtained by GetAcquisitionTimings. . Please refer to SECTION 5 – ACQUISITION MODES for further information.

Parameters float time: the kinetic cycle time in seconds.

Return unsigned int

DRV_SUCCESS	Cycle time accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Time invalid.

See also SetNumberKinetics

SetMCPGain

unsigned int WINAPI SetMCPGain(int gain)

Description Allows the user to control the voltage across the microchannel plate. Increasing the gain increases the voltage and so amplifies the signal. The gain range can be returned using GetMCPGainRange.

Parameters int gain: amount of gain applied.

Return unsigned int

DRV_SUCCESS	Value for gain accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I2C command timed out.
DRV_I2CDEVNOTFOUND	I2C device not present.
DRV_ERROR_ACK	Unable to communicate with device.
DRV_P1INVALID	Gain value invalid.

See also GetMCPGainRange, SetGateMode, SetMCPGating

NOTE: Available on iStar.

SetMCPGating

unsigned int WINAPI SetMCPGating(int gating)

Description This function controls the MCP gating.

Parameters int gating: ON/OFF switch for the MCP gating.

Valid values:		
	0	to switch MCP gating OFF.
	1	to switch MCP gating ON.

Return unsigned int

DRV_SUCCESS	Value for gating accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_I2CTIMEOUT	I^2C command timed out.
DRV_I2CDEVNOTFOUND	I^2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Value for gating invalid.

See also SetMCPGain, SetGateMode

NOTE: Available on some ICCD models.

SetMessageWindow

unsigned int WINAPI SetMessageWindow (HWND wnd)

Description **This function is reserved.**

SetMetaData

unsigned int WINAPI SetMetaData(int state)

Description This function activates the meta data option.

Parameters int state: ON/OFF switch for the meta data option.

Valid values:		
	0	to switch meta data OFF.
	1	to switch meta data ON.

Return unsigned int

DRV_SUCCESS	Meta data option accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid state.
DRV_NOT_AVAILABLE	Feature not available.

See also GetMetaDataInfo

SetMultiTrack

unsigned int WINAPI SetMultiTrack(int number, int height, int offset, int* bottom, int *gap)

Description

This function will set the multi-Track parameters. The tracks are automatically spread evenly over the detector. Validation of the parameters is carried out in the following order:

- Number of tracks,
- Track height
- Offset.

The first pixels row of the first track is returned via 'bottom'.

The number of rows between each track is returned via 'gap'.

Parameters

int number: number tracks

Valid values 1 to number of vertical pixels

int height: height of each track

Valid values >0 (maximum depends on number of tracks)

int offset: vertical displacement of tracks

Valid values depend on number of tracks and track height

int* bottom: first pixels row of the first track

int* gap: number of rows between each track (could be 0)

Return

unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number of tracks invalid.
DRV_P2INVALID	Track height invalid.
DRV_P3INVALID	Offset invalid.

See also

SetReadMode, StartAcquisition SetRandomTracks

SetMultiTrackHBin

unsigned int WINAPI SetMultiTrackHBin(int bin)

Description This function sets the horizontal binning used when acquiring in Multi-Track read mode.

Parameters int bin: Binning size.

Return unsigned int

DRV_SUCCESS	Binning set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid binning size.

See also SetReadMode SetMultiTrack

NOTE: 1) If the multitrack range is not a multiple of the binning DRV_BINNING_ERROR will be returned from PrepareAcquisition and/or StartAcquisition
2) For iDus, it is recommended that you set horizontal binning to 1

SetMultiTrackHRange

unsigned int WINAPI SetMultiTrackHRange (int iStart, int iEnd)

Description This function sets the horizontal range used when acquiring in Multi Track read mode.

Parameters int iStart: First horizontal pixel in multi track mode.
int iEnd: Last horizontal pixel in multi track mode.

Return unsigned int

DRV_SUCCESS	Range set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_AVAILABLE	Feature not available for this camera.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid start position.
DRV_P2INVALID	Invalid end position.

See also SetReadMode SetMultiTrack

SetNextAddress

unsigned int WINAPI SetNextAddress(at_32* data, long lowAdd, long highAdd, long len, long physical)

Description **THIS FUNCTION IS RESERVED.**

SetNextAddress16

unsigned int WINAPI SetNextAddress16(at_32* data, long lowAdd, long highAdd, long len, long physical)

Description **THIS FUNCTION IS RESERVED.**

SetNumberAccumulations

unsigned int WINAPI SetNumberAccumulations(int number)

Description This function will set the number of scans accumulated in memory. This will only take effect if the acquisition mode is either Accumulate or Kinetic Series.

Parameters int number: number of scans to accumulate

Return unsigned int

DRV_SUCCESS	Accumulations set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number of accumulates.

See also GetAcquisitionTimings, SetAccumulationCycleTime, SetAcquisitionMode, SetExposureTime, SetKineticCycleTime, SetNumberKinetics

SetNumberKinetics

unsigned int WINAPI SetNumberKinetics(int number)

Description This function will set the number of scans (possibly accumulated scans) to be taken during a single acquisition sequence. This will only take effect if the acquisition mode is Kinetic Series.

Parameters int number: number of scans to store

Return unsigned int

DRV_SUCCESS	Series length set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number in series invalid.

See also GetAcquisitionTimings, SetAccumulationCycleTime, SetAcquisitionMode, SetExposureTime, SetKineticCycleTime

---

SetNumberPrescans

unsigned int WINAPI SetNumberPrescans(int iNumber)

Description This function will set the number of scans acquired before data is to be retrieved. This will only take effect if the acquisition mode is Kinetic Series.

Parameters int iNumber: number of scans to ignore

Return unsigned int

DRV_SUCCESS	Prescans set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number of prescans invalid.

See also GetAcquisitionTimings, SetAcquisitionMode, SetKineticCycleTime, SetNumberKinetics

SetOutputAmplifier

unsigned int WINAPI SetOutputAmplifier(int typ)

Description Some EMCCD systems have the capability to use a second output amplifier. This function will set the type of output amplifier to be used when reading data from the head for these systems.

Parameters int typ: the type of output amplifier.

0 – Standard EMCCD gain register (default)/Conventional(clara).

1 – Conventional CCD register/Extended NIR mode(clara).

Return unsigned int

DRV_SUCCESS	Series length set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Output amplifier type invalid.

NOTE:

1. **Available in Clara, iXon & Newton.**
2. **If the current camera HSSpeed is not available when the amplifier is set then it will default to the maximum HSSpeed that is.**

SetOverlapMode

unsigned int WINAPI SetOverlapMode (int mode)

Description	This function will set whether an acquisition will readout in Overlap Mode. If the acquisition mode is Single Scan or Fast Kinetics this call will have no affect.
Parameters	int mode: mode 0 OFF 1 ON
Return	unsigned int

DRV_SUCCESS	Overlap mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid parameter.

See also SetAcquisitionMode

NOTE: Only available if CCD is an Overlap sensor.

SetPCIMode

unsigned int WINAPI SetPCIMode(int mode, int value)

Description With the CCI23 card, events can be sent when the camera is starting to expose and when it has finished exposing. This function will control whether those events happen or not.

Parameters int mode: currently must be set to 1
int value: 0 to disable the events, 1 to enable

Return unsigned int

DRV_SUCCESS	Acquisition mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Acquisition Mode invalid

See also SetAcqStatusEvent SetCameraStatusEnable

NOTE **This is only supported by the CCI23 card. The software must register its event via the SetAcqStatusEvent. To specify which event the software is interested in use the SetCameraStatusEnable.**

SetPhotonCounting

unsigned int WINAPI SetPhotonCounting(int state)

Description This function activates the photon counting option.

Parameters int state: ON/OFF switch for the photon counting option.

Valid values: 0 to switch photon counting OFF.
1 to switch photon counting ON.

Return unsigned int

DRV_SUCCESS	photon counting option accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.

See also SetPhotonCountingThreshold

SetPhotonCountingDivisions

unsigned int WINAPI SetPhotonCountingDivisions(unsigned long noOfDivisions, long* divisions)

Description This function sets the thresholds for the photon counting option.

Parameters

unsigned long noOfDivisions: number of thresholds to be used.

long* divisions: threshold levels.

Return unsigned int

DRV_SUCCESS	Thresholds accepted.
DRV_P1INVALID	Number of thresholds outside valid range
DRV_P2INVALID	Thresholds outside valid range
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_NOT_SUPPORTED	Feature not supported.

See also SetPhotonCounting, GetNumberPhotonCountingDivisions

SetPhotonCountingThreshold

unsigned int WINAPI SetPhotonCountingThreshold(long min, long max)

Description This function sets the minimum and maximum threshold for the photon counting option.

Parameters long min: minimum threshold in counts for photon counting.

long max: maximum threshold in counts for photon counting

Return unsigned int

DRV_SUCCESS	Thresholds accepted.
DRV_P1INVALID	Minimum threshold outside valid range (1-65535)
DRV_P2INVALID	Maximum threshold outside valid range
DRV_NOT_INITIALIZED	System not initialized.

	DRV_ACQUIRING	Acquisition in progress.
	DRV_ERROR_ACK	Unable to communicate with card.

See also SetPhotonCounting

SetPixelMode

unsigned int WINAPI SetPixelMode (int bitdepth, int colormode)

Description **THIS FUNCTION IS RESERVED.**

SetPreAmpGain

unsigned int WINAPI SetPreAmpGain(int index)

Description This function will set the pre amp gain to be used for subsequent acquisitions. The actual gain factor that will be applied can be found through a call to the GetPreAmpGain function.

The number of Pre Amp Gains available is found by calling the GetNumberPreAmpGains function.

Parameters int index: index pre amp gain table

Valid values 0 to GetNumberPreAmpGains-1

Return unsigned int

DRV_SUCCESS	Pre amp gain set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Index out of range.

See also IsPreAmpGainAvailable, GetNumberPreAmpGains, GetPreAmpGain

NOTE: Available on iDus, iXon & Newton.

SetRandomTracks

unsigned int WINAPI SetRandomTracks(int numTracks, int* areas)

Description This function will set the Random-Track parameters. The positions of the tracks are validated to ensure that the tracks are in increasing order and do not overlap. The horizontal binning is set via the SetCustomTrackHBin function. The vertical binning is set to the height of each track.

Some cameras need to have at least 1 row in between specified tracks. Ixon+ and the USB cameras allow tracks with no gaps in between.

Example:

Tracks specified as 20 30 31 40 tells the SDK that the first track starts at row 20 in the CCD and finishes at row 30. The next track starts at row 31 (no gap between tracks) and ends at row 40.

Parameters int numTracks: number tracks

Valid values 1 to number of vertical pixels/2

int* areas: pointer to an array of track positions. The array has the form

bottom1, top1, bottom2, top2 ….. bottomN, topN

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Number of tracks invalid.
DRV_P2INVALID	Track positions invalid.
DRV_RANDOM_TRACK_ERROR	Invalid combination of tracks, out of memory or mode not available.

See also SetCustomTrackHBin, SetReadMode, StartAcquisition, SetComplexImage

SetReadMode

unsigned int WINAPI SetReadMode(int mode)

Description This function will set the readout mode to be used on the subsequent acquisitions.

Parameters int mode: readout mode

Valid values:		
	0	Full Vertical Binning
	1	Multi-Track
	2	Random-Track
	3	Single-Track
	4	Image

Return unsigned int

DRV_SUCCESS	Readout mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid readout mode passed.

See also GetAcquisitionTimings, SetAccumulationCycleTime, SetAcquisitionMode, SetExposureTime, SetKineticCycleTime, SetNumberAccumulations, SetNumberKinetics

SetRegisterDump

unsigned int WINAPI SetRegisterDump(int mode)

Description **THIS FUNCTION IS RESERVED.**

SetRingExposureTimes

unsigned int WINAPI SetRingExposureTimes(int numTimes, float* times)

Description This function will send up an array of exposure times to the camera if the hardware supports the feature. See GetCapabilities. Each acquisition will then use the next exposure in the ring looping round to the start again when the end is reached. There can be a maximum of 16 exposures.

Parameters int numTimes: The number of exosures
float * times: A predeclared pointer to an array of numTimes floats

Return Unsigned int

DRV_SUCCESS	Success
DRV_NOT_INITIALIZED	System not initialized
DRV_INVALID_MODE	This mode is not available.
DRV_P1INVALID	Must be between 1 and 16 exposures inclusive
DRV_P2INVALID	The exposures times are invalid.
DRV_NOTAVAILABLE	System does not support this option

See also GetCapabilities, GetNumberRingExposureTimes, GetAdjustedRingExposureTimes, GetRingExposureRange IsTriggerModeAvailable

SetSaturationEvent

unsigned int WINAPI SetSaturationEvent(HANDLE saturationEvent)

Description This is only supported with the CCI-23 PCI card. USB cameras do not have this feature.

This function passes a Win32 Event handle to the driver via which the driver can inform the main software that an acquisition has saturated the sensor to a potentially damaging level. You must reset the event after it has been handled in order to receive additional triggers. Before deleting the event you must call SetEvent with NULL as the parameter.

Parameters HANDLE saturationEvent: Win32 event handle.

Return unsigned int

DRV_SUCCESS	Acquisition mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_SUPPORTED	Function not supported for operating system

See also SetDriverEvent

NOTE **The programmer must reset the event after it has been handled in order to receive additional triggers, unless the event has been created with auto-reset, e.g. event = CreateEvent(NULL, FALSE, FALSE, NULL). Also, NOT all programming environments allow the use of multiple threads and Win32 events.**

Only supported with the CCI-23 card.

USB cameras do not have this feature.

SetShutter

unsigned int WINAPI SetShutter(int typ, int mode, int closingtime, int openingtime)

Description This function controls the behaviour of the shutter.

The typ parameter allows the user to control the TTL signal output to an external shutter. The mode parameter configures whether the shutter opens & closes automatically (controlled by the camera) or is permanently open or permanently closed.

The opening and closing time specify the time required to open and close the shutter (this information is required for calculating acquisition timings – see SHUTTER TRANSFER TIME).

Parameters int typ:

0	Output TTL low signal to open shutter
1	Output TTL high signal to open shutter

int mode:

0	Auto
1	Open
2	Close

int closingtime: Time shutter takes to close (milliseconds)

int openingtime: Time shutter takes to open (milliseconds)

Return unsigned int

DRV_SUCCESS	Shutter set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid TTL type.
DRV_P2INVALID	Invalid mode.
DRV_P3INVALID	Invalid time to open.
DRV_P4INVALID	Invalid time to close.

NOTE

1. **The opening and closing time can be different.**
2. **For cameras capable of controlling the internal and external shutter independently (capability AC_FEATURES_SHUTTEREX) you MUST use SetShutterEx.**
3. **Cameras with an internal shutter (use function IsInternalMechanicalShutter to test) but no independent shutter control (capability AC_FEATURES_SHUTTEREX) will always output a "HIGH to open" TTL signal through the external shutter port.**

SetShutterEx

unsigned int WINAPI SetShutterEx(int typ, int mode, int closingtime, int openingtime, int extmode)

Description

This function expands the control offered by SetShutter to allow an external shutter and internal shutter to be controlled independently (only available on some cameras – please consult your Camera User Guide). The typ parameter allows the user to control the TTL signal output to an external shutter. The opening and closing times specify the length of time required to open and close the shutter (this information is required for calculating acquisition timings – see SHUTTER TRANSFER TIME).

The mode and extmode parameters control the behaviour of the internal and external shutters. To have an **external** shutter open and close automatically in an experiment, set the mode parameter to “Open” and set the ***extmode*** parameter to “Auto”. To have an **internal** shutter open and close automatically in an experiment, set the *extmode* parameter to “Open” and set the **mode** parameter to “Auto”.

To not use any shutter in the experiment, set both shutter modes to permanently open.

Parameters

Int typ:

0	Output TTL low signal to open shutter
1	Output TTL high signal to open shutter

int mode:

0	Auto
1	Open
2	Close

int closingtime: time shutter takes to close (milliseconds)

int openingtime: Time shutter takes to open (milliseconds)

int mode:

0	Auto
1	Open
2	Close

Return

Unsigned int

DRV_SUCCESS	Shutter set.
DRV_NOT_INITIALIZED	System not initialized
DRV_ACQUIRING	Acquisition in progress
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Invalid TTL type.
DRV_P2INVALID	Invalid internal mode
DRV_P3INVALID	Invalid time to open.
DRV_P4INVALID	Invalid time to close
DRV_P5INVALID	Invalid external mode

NOTE

1. **The opening and closing time can be different.**
2. **For cameras capable of controlling the internal and external shutter independently (capability AC_FEATURES_SHUTTEREX) you MUST use SetShutterEx.**
3. **For cameras with an internal shutter (use function IsInternalMechanicalShutter to test) but**

no independent shutter control (capability AC_FEATURES_SHUTTEREX), the external shutter will always behave like the internal shutter and the externalMode parameter is meaningless.

SetShutters

unsigned int WINAPI SetShutters(int typ, int mode, int closingtime, int openingtime, int exttype, int extmode, int dummy1, int dummy2)

Description **THIS FUNCTION IS RESERVED.**

SetSifComment

unsigned int WINAPI SetSifComment(char* comment)

Description This function will set the user text that will be added to any sif files created with the SaveAsSif function. The stored comment can be cleared by passing NULL or an empty text string.

Parameters char* comment: The comment to add to new sif files.

Return unsigned int

DRV_SUCCESS	Sif comment set.

See also SaveAsSif SaveAsCommentedSif

NOTE: To add a comment to a SIF file that will not be used in any future SIF files that are saved, use the function SaveAsCommentedSif.

SetSingleTrack

unsigned int WINAPI SetSingleTrack(int centre, int height)

Description This function will set the single track parameters. The parameters are validated in the following order: centre row and then track height.

Parameters int centre: centre row of track
Valid range 0 to number of vertical pixels.
int height: height of track
Valid range > 1 (maximum value depends on centre row and number of vertical pixels).

Return unsigned int

DRV_SUCCESS	Parameters set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Center row invalid.
DRV_P2INVALID	Track height invalid.

See also SetReadMode

SetSingleTrackHBin

unsigned int WINAPI SetSingleTrackHBin(int bin)

Description	This function sets the horizontal binning used when acquiring in Single Track read mode.	
Parameters	Int bin: Binning size.	
Return	unsigned int	
	DRV_SUCCESS	Binning set.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_P1INVALID	Invalid binning size.
See also	SetReadMode	

NOTE: 1) If the detector width is not a multiple of the binning DRV_BINNING_ERROR will be returned from PrepareAcquisition and/or StartAcquisition

2) For iDus, it is recommended that you set horizontal binning to 1

SetSpool

unsigned int WINAPI SetSpool(int active, int method, char* path, int framebuffersize)

Description

This function will enable and disable the spooling of acquired data to the hard disk or to the RAM.

With spooling method 0, each scan in the series will be saved to a separate file composed of a sequence of 32-bit integers.

With spooling method 1 the type of data in the output files depends on what type of acquisition is taking place (see below).

Spooling method 2 writes out the data to file as 16-bit integers.

Spooling method 3 creates a directory structure for storing images where multiple images may appear in each file within the directory structure and the files may be spread across multiple directories. Like method 1 the data type of the image pixels depends on whether accumulate mode is being used.

Method 4 Creates a RAM disk for storing images so you should ensure that there is enough free RAM to store the full acquisition.

Methods 5, 6 and 7 can be used to directly spool out to a particular file type, either FITS, SIF or TIFF respectively. In the case of FITS and TIFF the data will be written out as 16-bit values.

Method 8 is similar to method 3, however the data is first compressed before writing to disk. In some circumstances this may improve the maximum rate of writing images to disk, however as the compression can be very CPU intensive this option may not be suitable on slower processors.

The data is stored in row order starting with the row nearest the readout register. With the exception of methods 5, 6 and 7, the data acquired during a spooled acquisition can be retrieved through the normal functions. This is a change to previous versions; it is no longer necessary to load the data from disk from your own application.

Parameters

int active: Enable/disable spooling
Valid values:

0	Disable spooling.
1	Enable spooling.

int method: Indicates the format of the files written to disk
Valid values:

0.	Files contain sequence of 32-bit integers
1	Format of data in files depends on whether multiple accumulations are being taken for each scan. Format will be 32-bit integer if data is being accumulated each scan; otherwise the format will be 16-bit integer.
2.	Files contain sequence of 16-bit integers.
3.	Multiple directory structure with multiple images per file and multiple files per directory.
4.	Spool to RAM disk.
5.	Spool to 16-bit Fits File.
6.	Spool to Andor Sif format.
7.	Spool to 16-bit Tiff File.
8.	Similar to method 3 but with data compression.

char* path: String containing the filename stem. May also contain the path to the directory into which the files are to be stored.

int framebuffersize: This sets the size of an internal circular buffer used as temporary storage. The value is the total number images the buffer can hold, not the size in bytes. Typical value would be 10. This value would be increased in situations where the computer is not able to spool the data to disk at the

required rate.

Return	unsigned int	
	DRV_SUCCESS	Parameters set.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.

See also GetSpoolProgress

SetSpoolThreadCount

unsigned int WINAPI SetSpoolThreadCount(int count)

Description This function sets the number of parallel threads used for writing data to disk when spooling is enabled. Increasing this to a value greater than the default of 1, can sometimes improve the data rate to the hard disk particularly with Solid State hard disks. In other cases increasing this value may actually reduce the rate at which data is written to disk.

Parameters int count: The number of threads to use.

Return	unsigned int	
	DRV_SUCCESS	Thread count is set.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_P1INVALID	Invalid thread count.

See also SetSpool

NOTE: This feature is currently only available when using the Neo camera.

SetStorageMode

unsigned int WINAPI SetStorageMode(long mode)

Description **THIS FUNCTION IS RESERVED.**

SetTemperature

unsigned int WINAPI SetTemperature(int temperature)

Description This function will set the desired temperature of the detector. To turn the cooling ON and OFF use the CoolerON and CoolerOFF function respectively.

Parameters int temperature: the temperature in Centigrade.
Valid range is given by GetTemperatureRange

Return	unsigned int	
	DRV_SUCCESS	Temperature set.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_ERROR_ACK	Unable to communicate with card.
	DRV_P1INVALID	Temperature invalid.
	DRV_NOT_SUPPORTED	The camera does not support setting the temperature.

See also CoolerOFF, CoolerON, GetTemperature, GetTemperatureF, GetTemperatureRange

NOTE: Not available on Luca R cameras – automatically cooled to -20.

SetTriggerInvert

unsigned int WINAPI SetTriggerInvert(int mode)

Description This function will set whether an acquisition will be triggered on a rising or falling edge external trigger.

Parameters int mode: trigger mode

Valid values:

0. Rising Edge
1. Falling Edge

Return unsigned int

DRV_SUCCESS	Trigger mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Trigger mode invalid.
DRV_NOT_AVAILABLE	Feature not available.

See also Trigger Modes SetTriggerMode SetFastExtTrigger

SetTriggerMode

unsigned int WINAPI SetTriggerMode(int mode)

Description This function will set the trigger mode that the camera will operate in.

Parameters int mode: trigger mode

Valid values:

0. Internal
1. External
6. External Start
7. External Exposure (Bulb)
9. External FVB EM (only valid for EM Newton models in FVB mode)
10. Software Trigger

Return unsigned int

DRV_SUCCESS	Trigger mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Trigger mode invalid.

See also Trigger Modes SetFastExtTrigger

SetIODirection

unsigned int WINAPI SetIODirection(int index, int iDirection)

Description Available in some systems are a number of IO's that can be configured to be inputs or outputs. This function sets the current state of a particular IO.

Parameters int index: IO index
Valid values: 0 to GetNumberIO() - 1
int iDirection: requested direction for this index.
0: Output
1: Input

Return unsigned int

DRV_SUCCESS	IO direction set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.
DRV_P2INVALID	Invalid direction.
DRV_NOT_AVAILABLE	Feature not available.

See also GetNumberIO GetIOLevel GetIODirection SetIOLevel

SetIOLevel

unsigned int WINAPI SetIOLevel(int index, int iLevel)

Description Available in some systems are a number of IO's that can be configured to be inputs or outputs. This function sets the current state of a particular IO.

Parameters int index: IO index
Valid values: 0 to GetNumberIO() - 1
int iLevel: current level for this index.
0: Low
1: High

Return unsigned int

DRV_SUCCESS	IO level set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid index.
DRV_P2INVALID	Invalid level.
DRV_NOT_AVAILABLE	Feature not available.

See also GetNumberIO GetIOLevel GetIODirection SetIODirection

SetUserEvent

unsigned int WINAPI SetUserEvent(HANDLE userEvent)

Description **THIS FUNCTION IS RESERVED.**

SetVerticalRowBuffer

unsigned int WINAPI SetVerticalRowBuffer(int rows)

Description **THIS FUNCTION IS RESERVED.**

SetVerticalSpeed

unsigned int WINAPI SetVerticalSpeed(int index)

Description **Deprecated see Note:**
This function will set the vertical speed to be used for subsequent acquisitions

Parameters int index: index into the vertical speed table
Valid values 0 to GetNumberVerticalSpeeds-1

Return unsigned int

DRV_SUCCESS	Vertical speed set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Index out of range.

See also GetNumberVerticalSpeeds, GetVerticalSpeed
NOTE: Deprecated by SetVSSpeed.

SetVirtualChip

unsigned int WINAPI SetVirtualChip(int state)

Description **THIS FUNCTION IS RESERVED.**

SetVSAmplitude

unsigned int WINAPI SetVSAmplitude(int state)

Description If you choose a high readout speed (a low readout time), then you should also consider increasing the amplitude of the Vertical Clock Voltage.

There are five levels of amplitude available for you to choose from:

- **Normal**
- **+1**
- **+2**
- **+3**
- **+4**

Exercise caution when increasing the amplitude of the vertical clock voltage, since higher clocking voltages may result in increased clock-induced charge (noise) in your signal. In general, only the very highest vertical clocking speeds are likely to benefit from an increased vertical clock voltage amplitude.

Parameters int state: desired Vertical Clock Voltage Amplitude

Valid values:

0 - Normal

1->4 – Increasing Clock voltage Amplitude

Return unsigned int

DRV_SUCCESS	Amplitude set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_NOT_AVAILABLE	Your system does not support this feature
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid amplitude parameter.

NOTE: Available in iXon, iKon and Newton – full range of amplitude levels is not available on all compatible cameras.

SetVSSpeed

unsigned int WINAPI SetVSSpeed(int index)

Description	This function will set the vertical speed to be used for subsequent acquisitions	
Parameters	int index: index into the vertical speed table	
	Valid values 0 to GetNumberVSSpeeds-1	
Return	unsigned int	
	DRV_SUCCESS	Vertical speed set.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
	DRV_P1INVALID	Index out of range.
See also	GetNumberVSSpeeds, GetVSSpeed, GetFastestRecommendedVSSpeed	

ShutDown

unsigned int WINAPI ShutDown(void)

Description	This function will close the AndorMCD system down.	
Parameters	NONE	
Return	unsigned int	
	DRV_SUCCESS	System shut down.
See also	CoolerOFF, CoolerON, SetTemperature, GetTemperature	

NOTE:

1. **For Classic & ICCD systems, the temperature of the detector should be above -20ºC before shutting down the system.**
2. **When dynamically loading a DLL which is statically linked to the SDK library, ShutDown MUST be called before unloading.**

StartAcquisition

unsigned int WINAPI StartAcquisition(void)

Description This function starts an acquisition. The status of the acquisition can be monitored via GetStatus().

Parameters NONE

Return unsigned int

DRV_SUCCESS	Acquisition started.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_VXDNOTINSTALLED	VxD not loaded.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_INIERROR	Error reading “DETECTOR.INI”.
DRV_ACQERROR	Acquisition settings invalid.
DRV_ERROR_PAGELOCK	Unable to allocate memory.
DRV_INVALID_FILTER	Filter not available for current acquisition.
DRV_BINNING_ERROR	Range not multiple of horizontal binning.

See also GetStatus, GetAcquisitionTimings, SetAccumulationCycleTime, SetAcquisitionMode, SetExposureTime, SetHSSpeed, SetKineticCycleTime, SetMultiTrack, SetNumberAccumulations, SetNumberKinetics, SetReadMode, SetSingleTrack, SetTriggerMode, SetVSSpeed

UnMapPhysicalAddress

unsigned int WINAPI UnMapPhysicalAddress(void)

Description **THIS FUNCTION IS RESERVED.**

WaitForAcquisition

unsigned int WINAPI WaitForAcquisition(void)

Description WaitForAcquisition can be called after an acquisition is started using StartAcquisition to put the calling thread to sleep until an Acquisition Event occurs. This can be used as a simple alternative to the functionality provided by the SetDriverEvent function, as all Event creation and handling is performed internally by the SDK library.

Like the SetDriverEvent functionality it will use less processor resources than continuously polling with the GetStatus function. If you wish to restart the calling thread without waiting for an Acquisition event, call the function CancelWait.

An Acquisition Event occurs each time a new image is acquired during an Accumulation, Kinetic Series or Run-Till-Abort acquisition or at the end of a Single Scan Acquisition.

If a second event occurs before the first one has been acknowledged, the first one will be ignored. Care should be taken in this case, as you may have to use CancelWait to exit the function.

Parameters NONE

Return unsigned int

DRV_SUCCESS	Acquisition Event occurred
DRV_NOT_INITIALIZED	System not initialized.
DRV_NO_NEW_DATA	Non-Acquisition Event occurred.(e.g. CancelWait () called)

See also StartAcquisition, CancelWait

WaitForAcquisitionByHandle

unsigned int WINAPI WaitForAcquisitionByHandle(long cameraHandle)

Description Whilst using multiple cameras WaitForAcquisitionByHandle can be called after an acquisition is started using StartAcquisition to put the calling thread to sleep until an Acquisition Event occurs. This can be used as a simple alternative to the functionality provided by the SetDriverEvent function, as all Event creation and handling is performed internally by the SDK library. Like the SetDriverEvent functionality it will use less processor resources than continuously polling with the GetStatus function. If you wish to restart the calling thread without waiting for an Acquisition event, call the function CancelWait. An Acquisition Event occurs each time a new image is acquired during an Accumulation, Kinetic Series or Run-Till-Abort acquisition or at the end of a Single Scan Acquisition.

Parameters Long cameraHandle: handle of camera to put into wait state.

Return unsigned int

DRV_SUCCESS	Acquisition Event occurred.
DRV_P1INVALID	Handle not valid.

DRV_NO_NEW_DATA	Non-Acquisition Event occurred.(eg CancelWait () called)

See also CancelWait, GetCameraHandle, StartAcquisition, WaitForAcquisition, WaitForAcquisitionTimeOut, WaitForAcquisitionByHandleTimeOut.

WaitForAcquisitionByHandleTimeOut

unsigned int WINAPI WaitForAcquisitionByHandleTimeOut (long cameraHandle, int iTimeOutMs)

Description Whilst using multiple cameras WaitForAcquisitionByHandle can be called after an acquisition is started using StartAcquisition to put the calling thread to sleep until an Acquisition Event occurs. This can be used as a simple alternative to the functionality provided by the SetDriverEvent function, as all Event creation and handling is performed internally by the SDK library. Like the SetDriverEvent functionality it will use less processor resources than continuously polling with the GetStatus function. If you wish to restart the calling thread without waiting for an Acquisition event, call the function CancelWait. An Acquisition Event occurs each time a new image is acquired during an Accumulation, Kinetic Series or Run-Till-Abort acquisition or at the end of a Single Scan Acquisition. If an Acquisition Event does not occur within _TimeOutMs milliseconds, WaitForAcquisitionTimeOut returns DRV_NO_NEW_DATA

Parameters Long cameraHandle: handle of camera to put into wait state.
int iTimeOutMs: Time before returning DRV_NO_NEW_DATA if no Acquisition Event occurs.

Return unsigned int

DRV_SUCCESS	Acquisition Event occurred.
DRV_P1INVALID	Handle not valid.
DRV_NO_NEW_DATA	Non-Acquisition Event occurred.(eg CancelWait () called, time out)

See also CancelWait, GetCameraHandle, StartAcquisition, WaitForAcquisition, WaitForAcquisitionByHandle, WaitForAcquisitionTimeOut.

WaitForAcquisitionTimeOut

unsigned int WINAPI WaitForAcquisitionTimeOut (int iTimeOutMs)

Description WaitForAcquisitionTimeOut can be called after an acquisition is started using StartAcquisition to put the calling thread to sleep until an Acquisition Event occurs. This can be used as a simple alternative to the functionality provided by the SetDriverEvent function, as all Event creation and handling is performed internally by the SDK library. Like the SetDriverEvent functionality it will use less processor resources than continuously polling with the GetStatus function. If you wish to restart the calling thread without waiting for an Acquisition event, call the function CancelWait. An Acquisition Event occurs each time a new image is acquired during an Accumulation, Kinetic Series or Run-Till-Abort acquisition or at the end of a Single Scan Acquisition. If an Acquisition Event does not occur within _TimeOutMs milliseconds, WaitForAcquisitionTimeOut returns DRV_NO_NEW_DATA

Parameters int iTimeOutMs: Time before returning DRV_NO_NEW_DATA if no Acquisition Event occurs.

Return

DRV_SUCCESS	Acquisition Event occurred.
DRV_NO_NEW_DATA	Non-Acquisition Event occurred.(eg CancelWait () called, time out)

See also CancelWait, StartAcquisition, WaitForAcquisition, WaitForAcquisitionByHandle, WaitForAcquisitionByHandleTimeOut.

WhiteBalance

unsigned int WINAPI WhiteBalance (WORD* wRed, WORD* wGreen, WORD* wBlue, float * fRelR, float * fRelB, WhiteBalanceInfo * info)

Description **For colour sensors only**

Calculates the red and blue relative to green factors to white balance a colour image using the parameters stored in info.

Before passing the address of an WhiteBalanceInfo structure to the function the iSize member of the structure should be set to the size of the structure. In C++ this can be done with the line:

info-> iSize = sizeof(WhiteBalanceInfo);

Below is the WhiteBalanceInfo structure definition and a description of its members:

```
typedef struct WHITEBALANCEINFO {
 int iSize;   // Structure size.
 int iX;   // Number of X pixels. Must be >2.
 int iY;   // Number of Y pixels. Must be >2.
 int iAlgorithm;  // Algorithm to used to calculate white balance.
 int iROI_left;  // Region Of Interest from which white balance is calculated
 int iROI_right;  // Region Of Interest from which white balance is calculated
 int iROI_top;  // Region Of Interest from which white balance is calculated
 int iROI_bottom;  // Region Of Interest from which white balance is calculated
} WhiteBalanceInfo;
```

iX and iY are the image dimensions. The number of elements of the input, red, green and blue arrays are iX x iY.

iAlgorithm sets the algorithm to use. The function sums all the colour values per each colour field within the Region Of Interest (ROI) and calculates the relative to green values as: 0) _fRelR = GreenSum / RedSum and _fRelB = GreenSum / BlueSum; 1) _fRelR = 2/3 GreenSum / RedSum and _fRelB = 2/3 GreenSum / BlueSum, giving more importance to the green field.

iROI_left, iROI_right, iROI_top and iROI_bottom define the ROI with the constraints: 0 £ iROI_left < iROI_right £ iX and 0 £ iROI_ bottom < iROI_ top £ iX

Parameters WORD* wRed: pointer to red field.

WORD* wGreen: pointer to green field.

WORD* wBlue: pointer to blue field.

float* fRelR: pointer to the relative to green red factor.

float* fRelB: pointer to the relative to green blue factor.

WhiteBalanceInfo* info: pointer to white balance information structure

Return unsigned int

SUCCESS	White balance calculated.
DRV_P1INVALID	Invalid pointer (i.e. NULL).
DRV_P2INVALID	Invalid pointer (i.e. NULL).
DRV_P3INVALID	Invalid pointer (i.e. NULL).
DRV_P4INVALID	Invalid pointer (i.e. NULL).
DRV_P5INVALID	Invalid pointer (i.e. NULL).
DRV_P6INVALID	One or more parameters in info is out of range
DRV_DIVIDE_BY_ZERO_ERROR	The sum of the green field within the ROI is zero. _fRelR and _fRelB are set to 1

See also DemosaicImage, GetMostRecentColorImage16

SECTION 12 - ERROR CODES

CODE	ERROR	CODE	ERROR
DRV_ERROR_CODES	20001	DRV_P1INVALID	20066
DRV_SUCCESS	20002	DRV_P2INVALID	20067
DRV_VXDNOTINSTALLED	20003	DRV_P3INVALID	20068
DRV_ERROR_SCAN	20004	DRV_P4INVALID	20069
DRV_ERROR_CHECK_SUM	20005	DRV_INIERROR	20070
DRV_ERROR_FILELOAD	20006	DRV_COFERROR	20071
DRV_UNKNOWN_FUNCTION	20007	DRV_ACQUIRING	20072
DRV_ERROR_VXD_INIT	20008	DRV_IDLE	20073
DRV_ERROR_ADDRESS	20009	DRV_TEMPCYCLE	20074
DRV_ERROR_PAGELOCK	20010	DRV_NOT_INITIALIZED	20075
DRV_ERROR_PAGE_UNLOCK	20011	DRV_P5INVALID	20076
DRV_ERROR_BOARDTEST	20012	DRV_P6INVALID	20077
DRV_ERROR_ACK	20013	DRV_INVALID_MODE	20078
DRV_ERROR_UP_FIFO	20014	DRV_INVALID_FILTER	20079
DRV_ERROR_PATTERN	20015	DRV_I2CERRORS	20080
DRV_ACQUISITION_ERRORS	20017	DRV_DRV_I2CDEVNOTFOUND	20081
DRV_ACQ_BUFFER	20018	DRV_I2CTIMEOUT	20082
DRV_ACQ_DOWNFIFO_FULL	20019	DRV_P7INVALID	20083
DRV_PROC_UNKNOWN_INSTRUCTION	20020	DRV_USBERROR	20089
DRV_ILLEGAL_OP_CODE	20021	DRV_IOCERROR	20090
DRV_KINETIC_TIME_NOT_MET	20022	DRV_NOT_SUPPORTED	20091
DRV_KINETIC_TIME_NOT_MET	20022	DRV_USB_INTERRUPT_ENDPOINT_ERROR	20093
DRV_ACCUM_TIME_NOT_MET	20023	DRV_RANDOM_TRACK_ERROR	20094
DRV_NO_NEW_DATA	20024	DRV_INVALID_TRIGGER_MODE	20095
DRV_SPOOLERROR	20026	DRV_LOAD_FIRMWARE_ERROR	20096
DRV_TEMPERATURE_CODES	20033	DRV_DIVIDE_BY_ZERO_ERROR	20097
DRV_TEMPERATURE_OFF	20034	DRV_INVALID_RINGEXPOSURES	20098
DRV_TEMPERATURE_NOT_STABILIZED	20035	DRV_BINNING_ERROR	20099
DRV_TEMPERATURE_STABILIZED	20036	DRV_ERROR_NOCAMERA	20990
DRV_TEMPERATURE_NOT_REACHED	20037	DRV_NOT_SUPPORTED	20991
DRV_TEMPERATURE_OUT_RANGE	20038	DRV_NOT_AVAILABLE	20992
DRV_TEMPERATURE_NOT_SUPPORTED	20039	DRV_ERROR_MAP	20115
DRV_TEMPERATURE_DRIFT	20040	DRV_ERROR_UNMAP	20116
DRV_GENERAL_ERRORS	20049	DRV_ERROR_MDL	20117
DRV_INVALID_AUX	20050	DRV_ERROR_UNMDL	20118
DRV_COF_NOTLOADED	20051	DRV_ERROR_BUFFSIZE	20119
DRV_FPGAPROG	20052	DRV_ERROR_NOHANDLE	20121
DRV_FLEXERROR	20053	DRV_GATING_NOT_AVAILABLE	20130
DRV_GPIBERROR	20054	DRV_FPGA_VOLTAGE_ERROR	20131
DRV_DATATYPE	20064	DRV_BINNING_ERROR	20099
DRV_DRIVER_ERRORS	20065	DRV_INVALID_AMPLIFIER	20100

SECTION 13 - DETECTOR.INI

DETECTOR.INI EXPLAINED

All systems shipped from Andor contain a configuration file called "**Detector.ini**". This file is used to configure both the Andor software and hardware for the system. It contains information regarding the CCD chip, A/Ds and cooling capabilities.

The file contains four sections. The start of each section is denoted by **[*name*]**, where *name* is the name of the section. The following two sections are common to all **detector.ini** files:

- **[System]**
- **[Cooling]**

The names of the remaining sections are given by entries in the **[System]** section.

[SYSTEM]

This section has 3 entries that describe the controller, head models and the mode for operation. Each entry is described in more detail below:

- **Controller:** gives the section name where the controller (plug-in card) details can be found. Further details on this section are given below.
- **Head:** gives the section name where the detector head details can be found. Further details on this section are given below.
- **Operation:** this item related to the overall system type, i.e. whether the system is a PDA, CCD ICCD or InGaAs. This item has the effect of changing the "Acquisition" dialog within the software so that only those options relating to the system type are displayed.

Possible values are as follows:

- 2 for PDA
- 3 for InGaAs
- 4 for CCD
- 5 for ICCD

EXAMPLE:

[System]
Controller=CC-010
Head=DV437
Operation=4

[COOLING]

This section does not contain a fixed number of entries. However, each entry has the same basic structure and purpose. The purpose being to tell the software the range of temperatures to offer the user and the range of temperature over which the system can measure. The structure of each item is:

Itemname =a,b,c,d

itemname

a

b

c

d

Example:

[Cooling]
Single=28,-30,28,-100
Three=20,-60,28,-100
Vacuum=20,-100,28,-100

[DETECTOR]

This section details the detector head. It is the most complex section in the file and contains 10 or more items.

Format

Format = x,y

Gives the active pixel dimensions as x, y. x is the number of pixels along the readout register axis. y is the number of pixel perpendicular to the readout axis.

DummyPixels

DummyPixels = a, b, c, d

Gives the number of columns and row that are present on the device but do not respond to light. The dummy columns are a combination of dark columns, which run the full height of the sensor, and dummy pixels in the shift register, where:

a number of dummy columns at non-amplifier end
b number of dummy columns at amplifier end
c number of dummy rows at top of CCD
d number of dummy rows at bottom of CCD

DataHShiftSpeed

DataHShiftSpeed = a, b, c, d, e

Lists the speeds at which the charge can be moved in the shift register. This is also equivalent to the digitization speed in microseconds. Where:

a default speed

b, c ,d, e allowed speeds fastest first

DataVShiftSpeed

DataVShiftSpeed = a, b, c, d, e

This lists the speeds, in microseconds, at which the CCD rows can be vertically shifted. These speeds are used during CCD readout. Where:

a default speed

b, c, d, e allowed speeds fastest first

DummyHShiftSpeed

DummyHShiftSpeed = a, b, c, d, e

This lists the speeds, in microseconds, at which the charge can be moved in the shift register. These speeds are used when the charge been shifted in the amplifier does not need to be digitized. This allows faster keep clean cycles and faster readout when pixel skipping is implemented. Where:

a **default speed**

b, c, d, e **allowed speeds fastest first**

DummyVShiftSpeed

DummyVShiftSpeed = a, b, c, d, e

This lists the speeds, in microseconds, at which the CCD rows can be vertically shifted. These speeds are used during CCD keep cleans. Where:

a **default speed**
b, c, d, e **allowed speeds fastest first**

VerticalHorizontalTime

VerticalHorizontalTime = a,b,c,d,e

This lists the time, in microseconds, which must be taken into account when timing calculations are been done. Where:

a default speed

b, c, d, e allowed speeds fastest first

CodeFile

CodeFile = *filename.ext*

This gives the file name of the micro-code uploaded to the microprocessor on the plug-in card. This field is typically **PCI_29k.COF** for standard systems and **PCII29K.COF** for **I^2C** compatible cards.

FlexFile

FlexFile = *filename.ext*

This gives the file name of the logic uploaded to the Field Programmable Gate Array on the plug-in card. (This field is only used by the PCI version of the system.) This field is typically PCI_FPGA.RBF for standard systems and PCIIFPGA.RBF for I^2C compatible cards.

Cooling

Cooling = *type*

This gives the type of cooling. The type relates back to the cooling section.

Type

Type = *type*

This value specifies whether the head contains a Standard (0) or a Frame Transfer (1) CCD. The default is Standard.

FKVerticalShiftSpeed

FKVerticalShiftSpeed = *speed*

This specifies the “Fast Kinetics” vertical shift speed.

Gain

Gain = *a*

This specifies whether the system has software controllable Gain/Mode settings.
0 = Not software selectable.
1 = Software selectable.

PhotonCountingCCD

PhotonCountingCCD = *a*

This specifies whether the system contains a L3 Vision sensor from Marconi
0 = Standard CCD
1 = L3 Vision sensor

EMCCDRegisterSize

EMCCDRegisterSize = *a*

This specifies the length on the electron multiplying register in L3 Vision CCD

iStar

iStar = *a*

This specifies whether the system is an iStar or a standard ICCD
0 = Standard ICCD
1 = iStar

SlowVerticalSpeedFactor

SlowVerticalSpeedFactor = *a*

This specifies the factor by which the vertical shifted has been slowed. This is used for those CCD's that are not capable at running at 16us. The only possible value is 7.

HELLFunction

HELLFunction = file

The file specified contains the instructions required to perform readout of an iXon CCD. It is specific to each type of CCD.

HELLLoop1

HELLLoop1 = file

The file specified contains generic instructions for readout of an iXon CCD and as such is not specific to a particular CCD.

ADChannels

ADChannels = a{,b}

This line indicates the types of ADChannels available for use and the default selection. a is the default type and is followed by a list of all possible types.

AD2DataHSSpeed

AD2DataHSSpeed = default, min, max
This line specifies the possible horizontal readout speeds. min and max specify the range of readout times available in microseconds.

AD2DumpHSSpeed

AD2DumpHSSpeed = default, min, max

This is similar to AD2DataHSSpeed but specifies the readout speeds available when performing a dump (i.e. discarding) of data from the CCD.

AD2BinHSSpeed

AD2BinHSSpeed = default, min, max

This is similar to AD2DataHSSpeed but specifies the readout speeds available when binning (i.e. summing values from blocks of neighbouring pixels) data from the CCD.

AD2Pipeline

AD2Pipeline = a, b, c: See PipeLine in the controller section

iXon

Ixon = a

Specifies whether the CCD is an iXon camera; if so the line will read 'Ixon=1'. If this line is missing the CCD is not an iXon.

EXAMPLE DETECTOR.INI FILES

DH220

```
[DH220]
Format=1024,1
DummyPixels=0,0,0,0
DataHShiftSpeed=16,1,2,16,32
DataVShiftSpeed=16,16,0,0,0
DummyHShiftSpeed=16,1,2,16,32
DummyVShiftSpeed=16,16,0,0,0
VerticalHorizontalTime=16,16,0,0,0
CodeFile=Instapda.cof
Pixel=25.0,2500.0
Cooling=Single
```

DV420

```
[DV420]
Format=1024,256
DummyPixels=8,8,0,0
DataHShiftSpeed=16,1,2,16,32
DataVShiftSpeed=16,16,0,0,0
DummyHShiftSpeed=16,1,2,16,32
DummyVShiftSpeed=16,16,0,0,0
VerticalHorizontalTime=16,16,0,0,0
CodeFile=Pci_29k.cof
FlexFile = pci_fpga.rbf
```

Pixel=25.0,25.0
Cooling=Vacuum
FKVerticalShiftSpeed=16.0e-6

DV437

[DV437]
Format=512,512
DummyPixels=24,24,16,528
DataHShiftSpeed=16,1,2,16,32
DataVShiftSpeed=16,16,0,0,0
DummyHShiftSpeed=16,1,2,16,32
DummyVShiftSpeed=16,16,0,0,0
VerticalHorizontalTime=16,16,0,0,0
Pixel=13.0,13.0
Cooling=Vacuum
CodeFile=pci_29k.cof
FlexFile=pci_fpga.rbf
Type=1

[CONTROLLER]

This section details the controller card.

ReadOutSpeeds

ReadOutSpeeds = *a,b,c,d-*

Lists the readout speeds available on the specified plug-in card. These values are used in conjunction with the values specified in the head section to generate the final list of available speeds.

PipeLine

PipeLine=a,b,c,d,e,f,g,h

This lists the pipeline depth that must be used the microprocessor to synchronize the reading of the AD with the digitization process. The actual value used is based on a number of factors and is beyond this discussion.

Type

Type=a

This specifies whether the plug-in card is ISA or PCI compatible.

Example:

[CC-010]
ReadOutSpeeds=1,2,16,32
PipeLine=2,1,1,1,0,0,0,0
Type=PCI