USER'S GUIDE TO THE

www.andor.com

Software Release Version 2.72

Manual Version 2.0

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

PAGE

TABLE OF CONTENTS

PAGE

TABLE OF CONTENTS

PAGE

PAGE

PAGE

PAGE

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

The AndorMCD **S**oftware **D**evelopment **K**it (SDK) gives the programmer access to the Andor range of CCD, ICCD and iStar cameras. The key part of the SDK is the 32-bit dynamic link library "ATMCD32D.DLL" which can be used with a wide variety of programming environments including C, C++, Visual Basic and LabVIEW.

The library is compatible with Windows 2000 and XP. 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 it's 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 scan(s), and create the user interface.

The manual is broken into several sections, of which this is the first.

The second section will take you through the software installation process. References are made to the relevant sections of the User Guide (supplied with the Andor system) for hardware installation where appropriate.

The next 6 sections go through the basic operation of the system, for example, acquisition modes, types of triggering and temperature control. To further aid the user there is a comprehensive list of examples included with the SDK that give simple illustrations of operating the Andor system, followed by a basic outline of the examples.

The examples illustrate the use of C, Visual Basic and LabVIEW.

The next section is a complete function reference detailing the function syntax, parameters passed and error codes returned.

SECTION 2 - INSTALLATION

INTRODUCTION

The installation of the **AndorMCD 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. If the main **AndorMCD** or **iStar, iDus, iXon, Newton & Shamrock** software is already installed on the computer, proceed to step **3**.
2. Follow the instructions in the Users guide to AndorMCD (supplied with the system) for installing the hardware. If the AndorMCD software is not being installed then simply ignore section “The Software” of the installation instructions in the above guide.
3. Insert the CD supplied with the SDK, and execute the **SETUP.EXE** program. This will go through the complete software installation process.
4. The installation program will prompt for the destination directory. If this directory does not already exist it will be automatically created.
5. Example programmes will be copied into sub-directories of the installation directory specified above. For the complete list of files copied refer to the “README.TXT” file on the installation disk

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

ATMCD32D.DLL
ATMCD32D.H (needed for C, C++ only)
ATMCD32D.LIB (Borland compatible library, needed for C, C++ only)
ATMCD32M.LIB (Microsoft compatible library, needed for C, C++ only)
ATMCD32D.BAS (needed for Visual Basic only)
ATMCD32D.PAS (needed for Pascal only)
CYUSER.DLL
DETECTOR.INI (except for the iDus, iXon, Newton & Shamrock cameras)

NOTES:

1. **THE FILES ARE ALSO COPIED INTO EACH EXAMPLE DIRECTORY. THIS IS TO ALLOW EACH EXAMPLE TO BE RUN AS A STAND-ALONE PROGRAM.**
2. **ONCE THE INSTALLATION IS COMPLETE RESTART WINDOWS TO ENSURE THAT THE CORRECT DRIVER IS LOADED. IT IS HIGHLY RECOMMENDED THAT THE INSTALLATION IS TESTED, BY RUNNING ONE OF THE INSTALLED EXAMPLES, BEFORE STARTING TO DEVELOP YOUR OWN SOFTWARE.**
3. **DO NOT HAVE MORE THAN ONE EXAMPLE, OR AN EXAMPLE AND THE MAIN ANDORMCD SOFTWARE, RUNNING AT THE SAME TIME.**

The minimum recommended specification is:

- 2.4 GHz Pentium processor or better
- 512 MB of RAM
- 25MB free Hard Disc space
- CD-ROM compatible drive for installation of the software

The PC **must** support **USB 2.0** either with it's built in USB ports or with an additional USB2.0 PCI card. The operating system must be Windows 2000, XP or better.

The graphics card must be capable of at least 800 x 600 pixels resolution at 256 colors. A higher quality card will produce clearer, faster graphics. Use any monitor compatible with your graphics card.

The Andor iDus driver will have to be installed twice as follows:

1. When the iDus is plugged in for the first time
2. When the software is run for the first time. The **New Hardware Detected** function will run and the driver will have to be loaded using the on-screen prompts.

NOTE: AFTER THE DRIVER HAS BEEN INSTALLED ON A PARTICULAR USB 2.0 PC PORT, IT WILL RUN OK ON THAT PORT THEREAFTER. HOWEVER, IF THE DEVICE IS PLUGGED INTO ANOTHER USB 2.0 PORT LATER, THE DRIVER HAS TO BE RE-INSTALLED TO THE NEW PORT.

When developing programs for the iDus camera the following files must be included in the executable directory:

Cyuser.dll
Atmcd32d.dll

The Detector.ini is not needed for the iDus camera as all the Head details are stored in an internal EEPROM. Please remove or rename the detector.ini if it exists in the executable directory. In addition the Atmcd32d.h and Atmcd32d.lib/ Atmcd32m.lib may be included in your own project.

Following the installation of Andor USB SDK you will find all the above files under the **Examples\C** directory of your installation directory. They are included in every example directory.

TROUBLESHOOTING

If you are experiencing **communication problems** with the iDus carry ou the following actions:

- Confirm that the PC being used is **USB 2.0 compatible**
- Check the power to the iDus camera.
- Check the USB cable from the PC to the iDus camera.
- From the Device Manager in the Control Panel ensure that under the **Universal Serial Bus** controller tab there is an **Andor iDus** entry. If the entry does not exist carry out the following actions:

1. Check to see if there is an **Other devices** entry. If there is an entry under that with the title **TECHNOLOGY** then the driver for the iDus camera has not been installed.

2. 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 iDus software was installed**.**

3. Check to see if there is a **USB device**"with an **exclamation mark** beside it. If you cannot account for this device then it is probably the iDus camera and the driver is not installed. Install the driver as escribed previously or right click on the entry and update driver.

4. Close down iDus software, remove the USB cable from either the iDus or the PC and reconnect it again. Run the software to see if the iDus is now detected.

5. If still not connected, remove the USB cable from either the PC or the iDus, power off and on the iDus and reconnect the USB cable again.
6. Run the software to see if the iDus is now detected.

NOTE: IF STILL NOT DETECTED AFTER 6. ABOVE, PLEASE CONTACT YOUR LOCAL ANDOR TECHNICAL SUPPORT STAFF.

iStar

PC REQUIREMENTS

The system requires a PCI-compatible computer. The PCI slot you use must have bus master capability.

The minimum recommended specification is:

- 200 MHz Pentium Processor or better
- 128 Mbytes of RAM
- 32 MB free Hard Disc space
- CD-ROM compatible drive for installation of the software

The operating system must be Windows 2000 or XP.

FUNCTIONS

The following functions are only available for iStar detectors:

- **GetDDGPulse**
- **SetDDGGain**
- **SetDDGGateStep**
- **SetDDGInsertionDelay**
- **SetDDGIntelligate**
- **SetDDGIOC**
- **SetDDGIOCFrequency**
- **GetDDGIOCPulses**
- **SetDDGTimes**
- **SetDDGTriggerMode**
- **SetDDGVariableGateStep**

When creating a program of your own careful note should be taken of the order of commands. The number of pulses present, when integrate on chip is used, is affected by the timings of the experiment. This means the **SetDDGIOCFrequency** function should be called immediately prior to **StartAcquisition**.

iXon

PC REQUIREMENTS

The system requires a PCI-compatible computer. The PCI slot you use must have bus master capability. The minimum recommended specification is:

- 3GHz Pentium Processor or better.
- 1GB of RAM
- Minimum 10,000 RPM Hard drive (RAID 15,000 RPM preferred for extended Kinetic series)
- 32 MB free Hard Disc space.
- Auxiliary internal power connector availability

The operating system must be Windows 2000 or XP.

FUNCTIONS

The following functions are available for iXon, iDus & NEWTON detectors:

- **GetPreAmpGain**
- **GetNumberPreAmpGains**
- **IsPreAmpGainAvailable**
- **SetFanMode**
- **SetFastKineticsEx**
- **SetHighCapacity (only some iXon)**
- **SetPreAmpGain**
- **SetVSAmplitude**

SECTION 3 - READOUT MODES

INTRODUCTION

AndorMCD systems are based on a 2-dimensional detector known as a Charged Coupled Device (CCD). 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**.

AndorMCD has at present 5 different readout modes:

- **Full Vertical Binning**
  Single-Track
  Multi-Track
  Random Multi-Track
  Image/ Full Image.

Figure 1 shows the binning patterns :

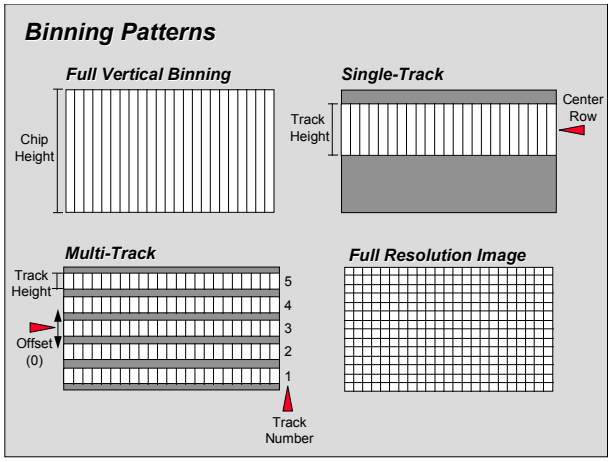


FIGURE 1 - BINNING PATTERNS

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

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)**

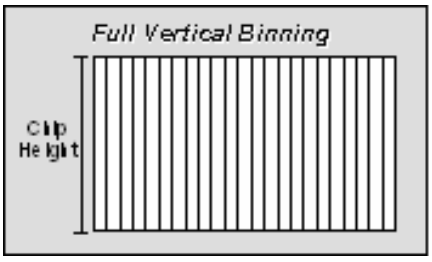


FIGURE 2 - FULL VERTICAL BINNING

Single Track

The next mode we shall look at is **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.

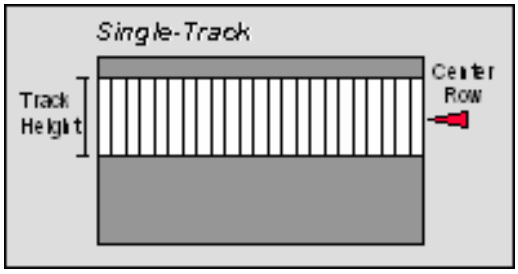


FIGURE 3 - SINGLE-TRACK

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); /* Single Track */

SetSingleTrack(128,20); /* Define track */

NOTE: 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. SEE SHUTTER CONTROL SECTION FOR FURTHER DETAILS.

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.

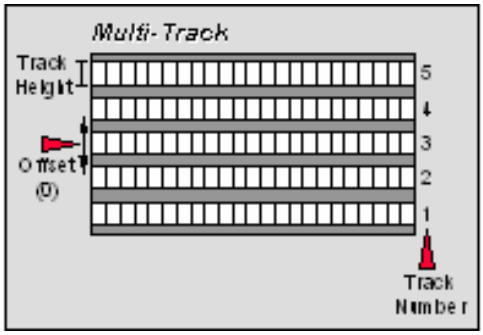


FIGURE 4 - MULTI-TRACK

Multiple tracks 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. See Shutter Control section for further details.

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

SetReadMode(1); /* Multi-Track */

SetMultiTrack(5,20,0,bottom, gap); /*Define tracks*/

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: 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.

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.

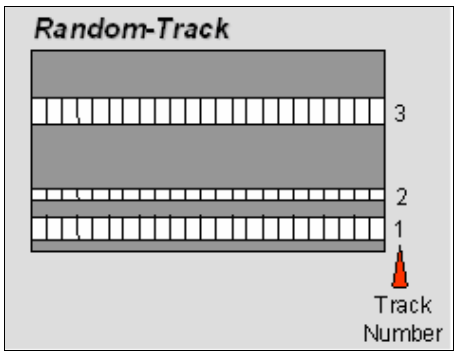


FIGURE 5 - RANDOM TRACK

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); /* Random-Track */
int position[6];
position[0] = 20;  //start of track 1
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); /*Define tracks*/
```

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

- A TRACK OF 1 ROW IN HEIGHT WILL HAVE THE SAME START AND END POSITIONS.
- BEFORE USING RANDOM-TRACK MODE WITH FIBER BUNDLES IT IS OFTEN USEFUL TO ACQUIRE A FULL RESOLUTION IMAGE OF THE OUTPUT.
- HAVING OBSERVED THE VERTICAL POSITIONS OF THE INDIVIDUAL SPECTRA SET THE RANDOM-TRACK MODE ACCORDINGLY.
- IMAGING SPECTROGRAPHS VERTICALLY INVERT INPUT LIGHT (I.E. LIGHT FROM THE TOP FIBER WILL FALL ON THE BOTTOM TRACK ON THE CCD-CHIP.)
- NOT AVAILABLE ON IXON
- RANDOM TRACKS ARE LIMITED TO 20 TRACKS FOR IDUS

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 read out process. See the section on **Shutter Control** for further information.

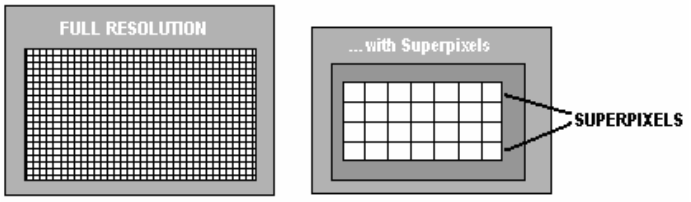


FIGURE 6 – IMAGE MODE

To reduce the file size and quicken the read out time 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 superpixels.

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

SetReadMode(4); /* Image */
SetImage(1,1,1,1024,1,256);/*full resolution image*/

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);/*sub-area image*/

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, ANDOR MCD system is effectively reading out charge from a matrix of superpixels which each measure 4x4. 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 superpixel 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).

SECTION 4 – ACQUISITION MODES

ACQUISITION MODE TYPES

In the previous section the different Read Out Modes (binning patterns) supported by the Andor MCD SDK were discussed. In addition the Andor MCD SDK allows you to control the number and the timing details of acquisitions made using the above binning patterns. To simplify the process of controlling these acquisitions the Andor MCD SDK has divided the acquisition process into 4 different **Acquisition Modes** asthat follows

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

Single Scan is the simplest form of acquisition and **Accumulate** builds on this idea to form a more complex concept whilst **Kinetic Series** in turn builds on Accumulate.

Run Till Abort continually performs scans of the CCD at the fastest possible rate 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.

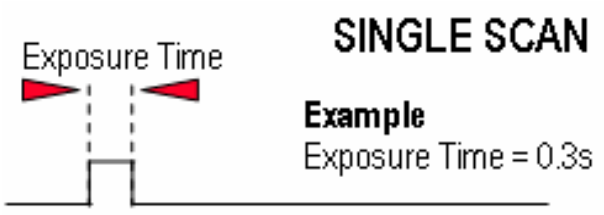
The table below summarizes the information that is needed for each acquisition mode:

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				

NOTE: FOR THE PURPOSE OF THIS DOCUMENT AN ACQUISITION IS TAKEN TO BE THE COMPLETE DATA CAPTURE PROCESS. BY CONTRAST A SCAN IS 1X 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 AndorMCD system. In this mode AndorMCD 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:

SetAcquistionMode(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.

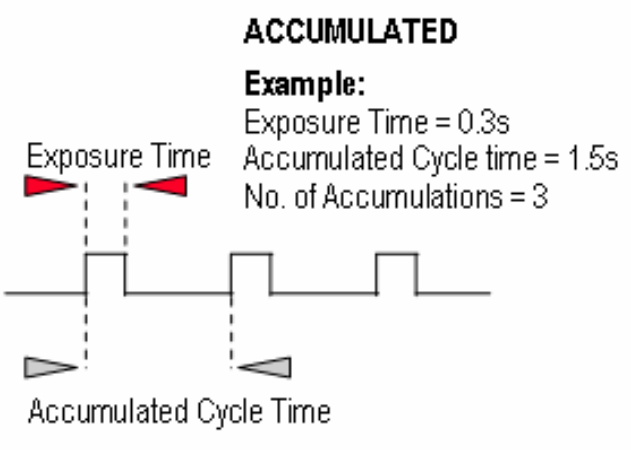
NOTE: DUE TO THE TIME NEEDED TO SHIFT CHARGE INTO THE SHIFT REGISTER, DIGITIZE IT AND OPERATE SHUTTERS, WHERE NECESSARY, THE EXPOSURE TIME CANNOT BE SET TO JUST ANY VALUE.

FOR EXAMPLE, THE MINIMUM EXPOSURE TIME DEPENDS ON MANY FACTORS INCLUDING THE READ OUT 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.

THUS THE ACTUAL CALCULATED EXPOSURE TIME USED BY ANDOR MCD MAY BE OBTAINED VIA THE GETACQUISITIONTIMINGS FUNCTION (THIS FUNCTION SHOULD BE CALLED AFTER THE ACQUISITION DETAILS HAVE BEEN FULLY DEFINED I.E. READ OUT 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 periodicity) between two consecutive scans and also the total number of scans to be added.



To set the acquisition mode to Accumulate call:

SetAcquistionMode(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 - see section **6** on **Trigger Modes**.)

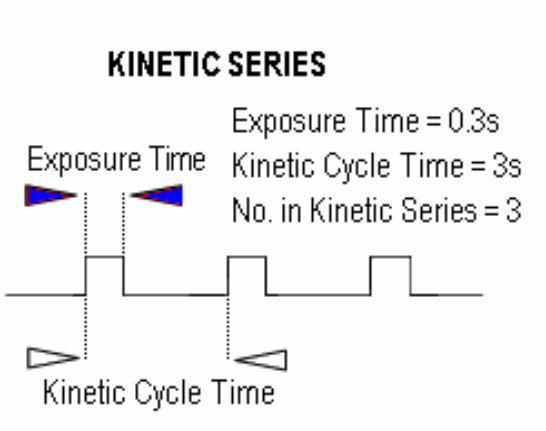
NOTE

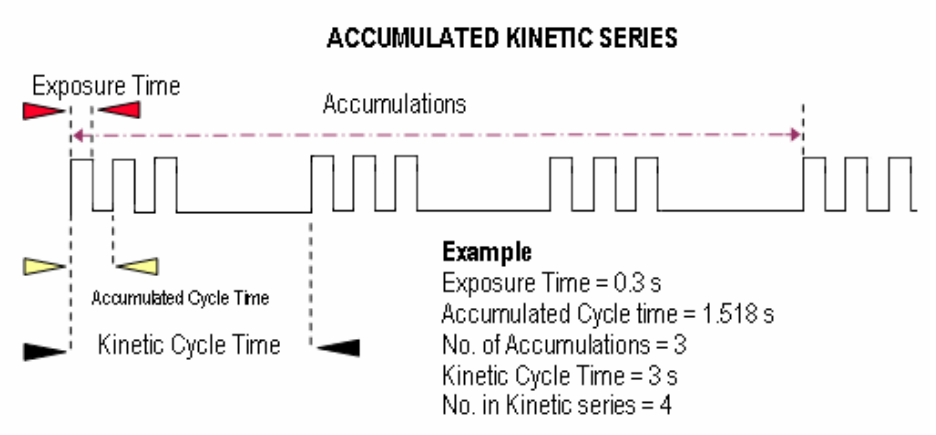
1. IF TOO LOW A VALUE, FOR THE EXPOSURE TIME OR THE CYCLE TIME, IS ENTERED, OR IF IT IS NOT A PERMISSIBLE VALUE, THE DRIVER WILL AUTOMATICALLY CALCULATE THE NEAREST PERMISSIBLE VALUE.
2. THE ACTUAL VALUES USED CAN BE OBTAINED VIA THE GETACQUISITIONTIMINGS FUNCTION (THIS FUNCTION SHOULD BE CALLED AFTER THE ACQUISITION HAS BEEN FULLY DEFINED (I.E. READ OUT 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 MCD BUT IS SYNCHRONIZED TO AN EXTERNALLY GENERATED TRIGGER PULSE.

Kinetic Series

Kinetic Series mode captures a sequence of single scans, or 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 periodicity) 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 ANDORMCD 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- see section on **Trigger Modes**.)

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 **SetKineticCycleTim**e function.

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

NOTE: IF TOO LOW A VALUE, FOR THE EXPOSURE TIME OR EITHER OF THE CYCLE TIMES, IS ENTERED, OR IF IT IS NOT A PERMISSIBLE VALUE, THE DRIVER WILL AUTOMATICALLY CALCULATE THE NEAREST PERMISSIBLE VALUE.

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. READ OUT MODE, TRIGGER MODE ETC. HAVE BEEN SET). IF YOU ARE USING A SHUTTER, PLEASE REFER TO SHUTTER CONTROL IN SECTION 7 FOR FURTHER DETAILS.

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

NOTE: 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 RETREIVE THE OLDEST IMAGE THE GETOLDESTIMAGE FUNCTION CAN BE USED.

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 READ OUT 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.

THUS THE ACTUAL CALCULATED EXPOSURE TIME USED BY ANDORMCD MAY BE OBTAINED VIA THE GETACQUISITIONTIMINGS FUNCTION (THIS FUNCTION SHOULD BE CALLED AFTER THE ACQUISITION DETAILS HAVE BEEN FULLY DEFINED I.E. READ OUT MODE, TRIGGER MODE ETC. HAVE BEEN SET).

Frame Transfer

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

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 read-out 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 read-out 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.

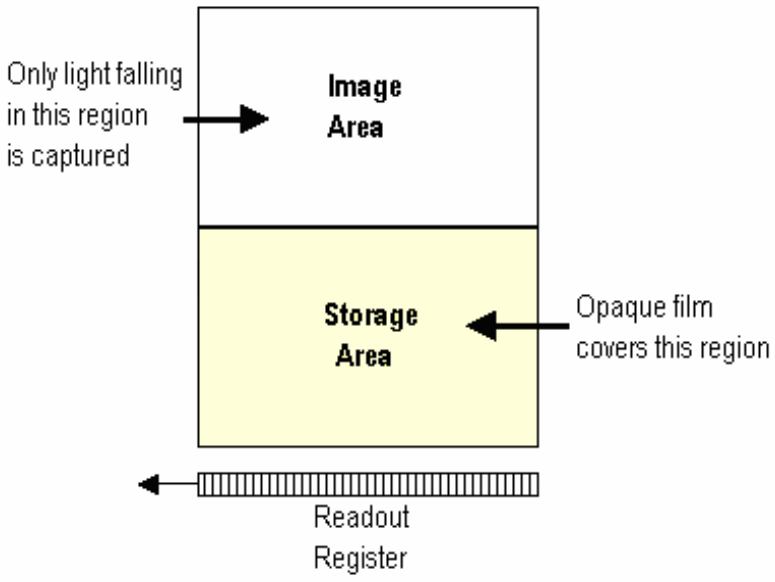


FIGURE 7 – FT CCD

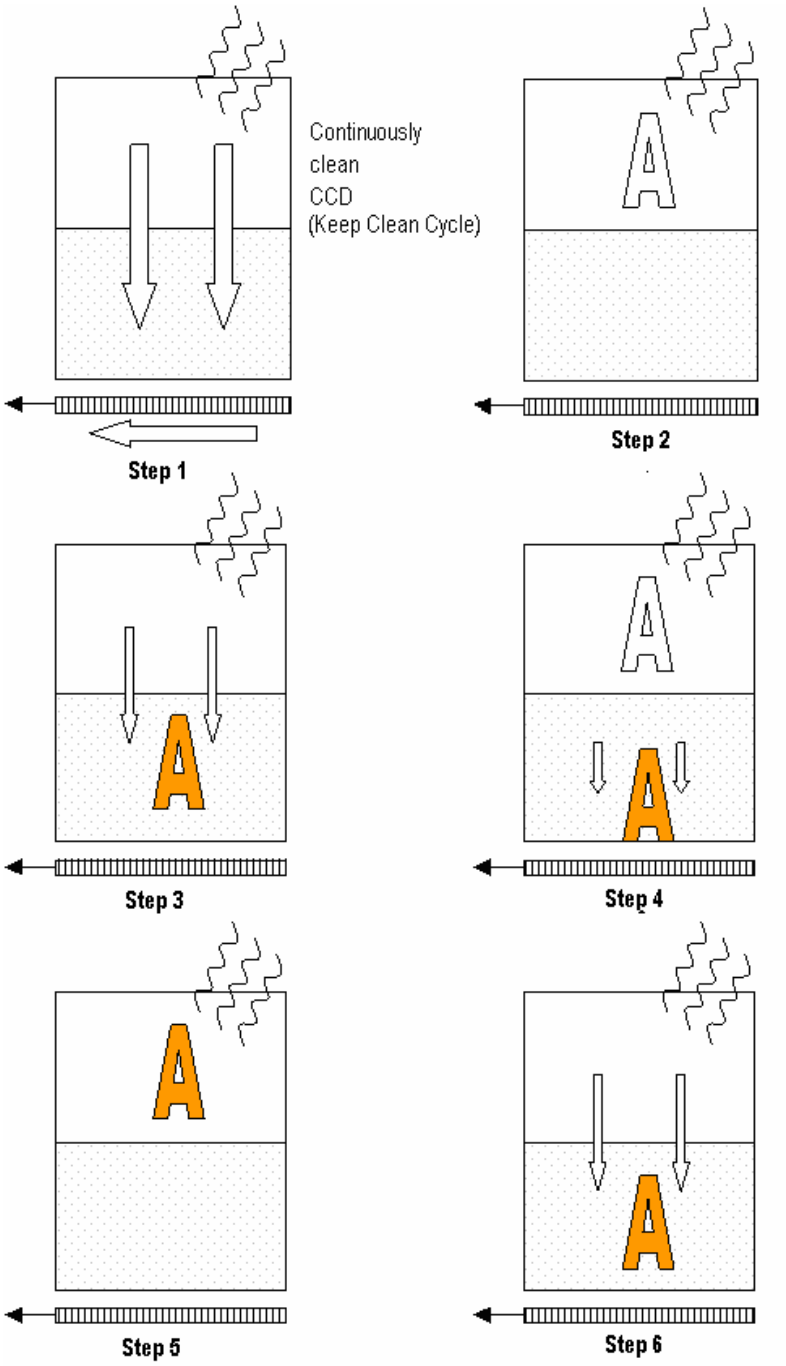


FIGURE 8 - CAPTURE SEQUENCE FOR AN FT CCD

Figure 8 on the previous page takes you through the capture sequence for a FT 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 read-out process starts.

Step 3 - The first phase of stage 3 is to quickly shift the charge, built up in the Image *area,* 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 still being read-out. This read-out phase can take tens of milliseconds to seconds depending on the image size, read-out pattern and read-out speed.

Step 5 - On completion of the read-out, the system will wait until the exposure time has elapsed before starting the next read-out (**Step 6)**.

As the captured image is quickly shifted into the Storage area, a FT 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 READ-OUT 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 SENSOR THE CAMERA CAN ALSO BE OPERATED IN EXTERNAL TRIGGER MODE. IN THIS MODE THERE ARE NO KEEP CLEANS AND THE EXTERNAL TRIGGER STARTS THE "READ OUT" 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.
- 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, 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, a FT CCD acts much like a standard CCD. The capture sequence for this standard mode is illustrated here:

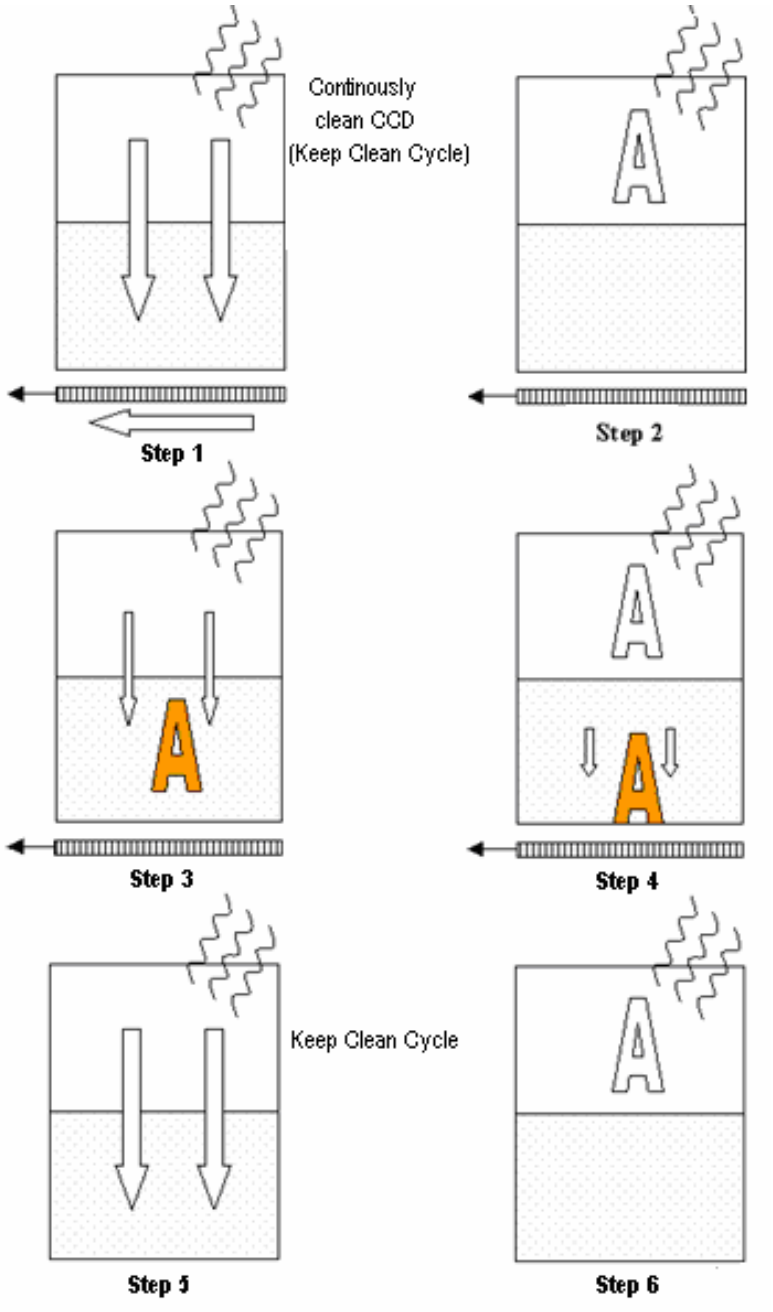


FIGURE 4 – FT CCD NON-FRAME TRANSFER MODE

- **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 read-out 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 read-out, the CCD is again completely cleared, ready to acquire the next image.
- **Step 6** - 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.

NOTE: WHEN USING A FT CCD AS A STANDARD CCD THE EXPOSURE TIME, ACCUMULATION CYCLE TIME AND THE KINETIC CYCLE TIME ARE INDEPENDENT.

THE MINIMUM EXPOSURE TIME IS NOT RELATED TO THE TIME TAKEN TO READOUT 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 read-out 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 16ms exposure enough light has fallen on a pixel to register 10000 counts. The image is then shifted into the Storage area. To read out the image, assuming 1000x1000 pixels, it will take approximately 16 seconds at 16 microseconds per pixel.

This means that during the reading out of the image 10M counts (10000 * 1000) will have been acquired into the pixel described above. As a pixel saturates at approximately 65K counts this means that the pixel will over saturated by a 150 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 16ms, the chance of corrupting data is correspondingly increased.

Changing the read-out 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

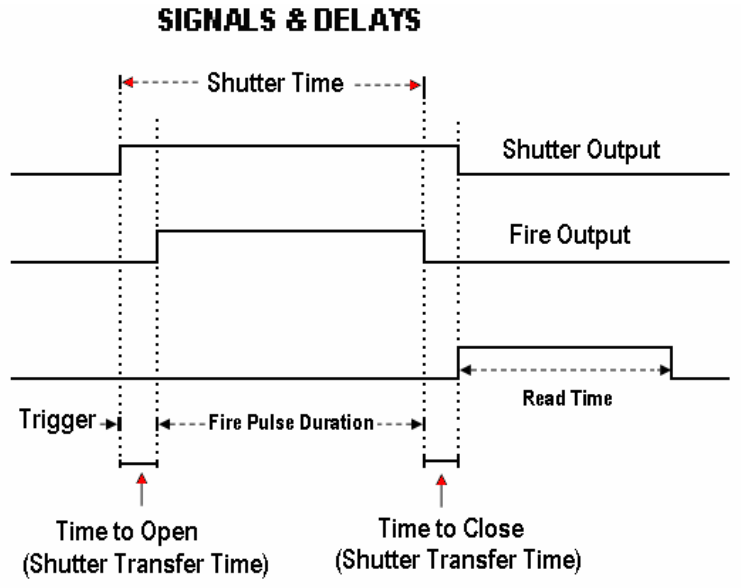
To assist the user in synchronizing data capture with external events the Andor MCD system supports two modes of triggering, **Internal** and **External**. The trigger mode is set via the **SetTriggerMode** function. In the remainder of this section we will examine the two modes in detail and give some 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 MCD system determines when data is actually acquired. Before Andor MCD 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 Andor MCD continually reads out the CCD to help prevent it from being saturated by light falling on it whilst it is not acquiring data).

Andor MCD also generates all the necessary pulses for shuttering and firing external sources. These pulses are accessed via the **Auxiliary Connector**. 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 (e.g. an ordinary room light,) and incoming data, for the purposes of your observation, are steady and unbroken. Thus you can begin acquisitions 'at will'.

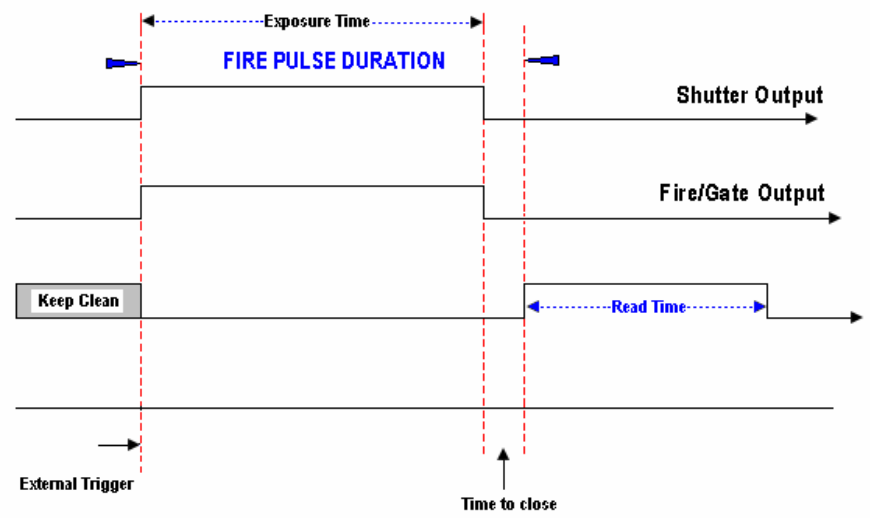
You can use Internal Trigger when you are able to send a trigger signal or **Fire Pulse** to a short-duration, pulsed source (e.g. a laser).

In this case initiating the data acquisition process can also signal the pulsed source to fire. The Fire Pulse is fed to the pulsed source.

External

In **External Trigger Mode** once an acquisition has been started via the **StartAcquisition** function the Andor MCD system 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 is fed via the EXT TRIG socket of the **I/O Box** (available separately, model #IO160) or to **Pin 13** of the **Auxiliary Connector** on the Andor MCD Card, or on the head in the case of iDus / iXon.

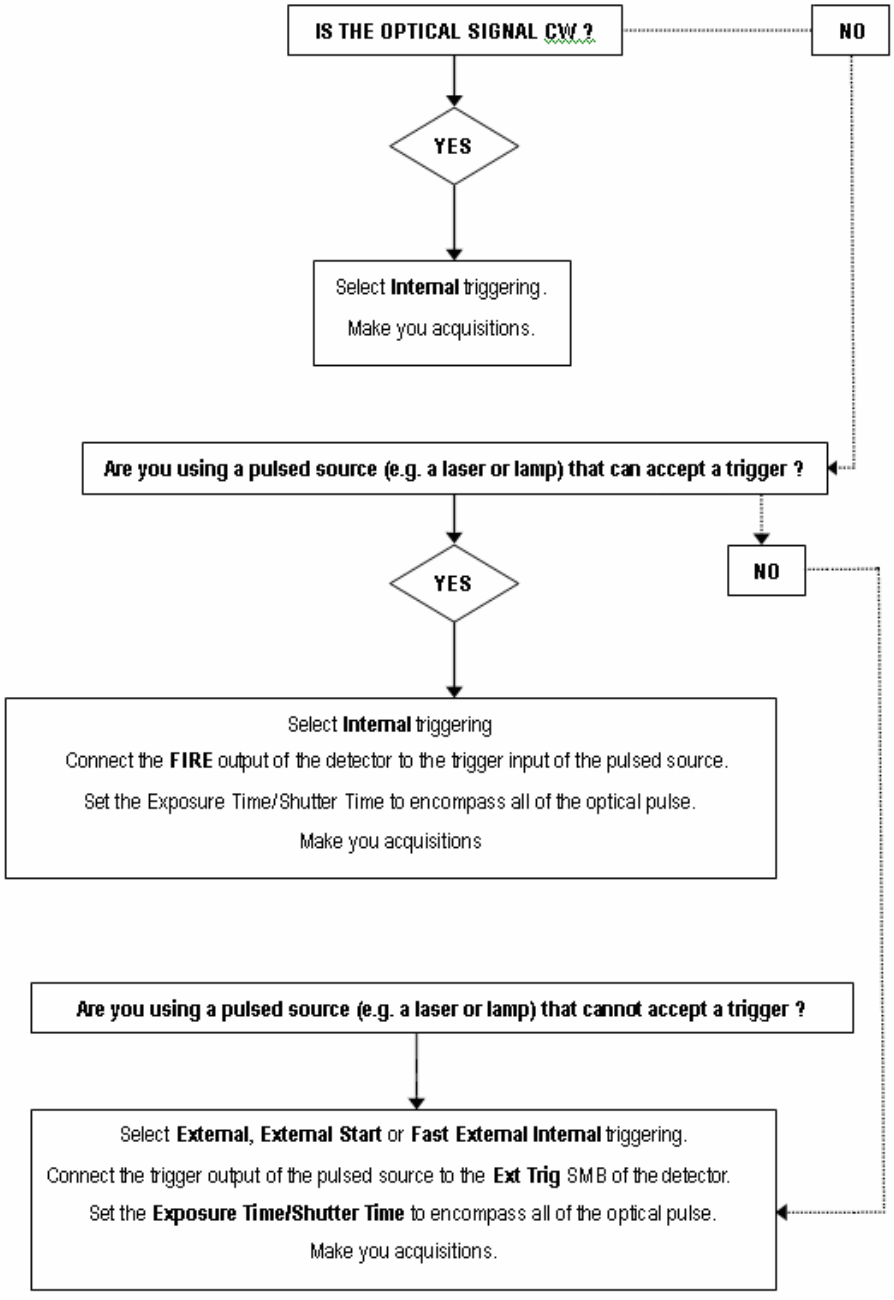
External Trigger Mode is suited to data acquisitions involving a 'pulsed source' (e.g. a laser) where the source does NOT allow a trigger pulse to be sent to it but can generate one. It is possible to increase the frame rate when in external trigger mode by enabling the Fast External Trigger option,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

NOTE: 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 USER MANUAL FOR A FULL DESCRIPTION.

Choosing a Trigger Mode

The following flowchart will help you decide whether you should use Internal or External Trigger Mode:



SECTION 6 – SHIFT SPEEDS

The AndorMCD 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 speeds and their actual values is 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 speeds and their actual values is 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:

```
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 smearing the 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 read out. The AndorMCD system has a dedicated shutter control line, which ensures that the shutter is correctly operated at all times.

The **SetShutter** function provides you with a selection of options, which 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 AndorMCD. The shutter opens and closes under the guidance of AndorMCD and 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. 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: NOT APPLICABLE TO IDUS, ISTAR, IXON & NEWTON MODELS.**

If you set the shutter type to **TTL High** with **SetShutter**, AndorMCD will cause the output voltage from the AndorMCD Card to go 'high' to open the shutter.

If you set the shutter type to **TTL Low** with **SetShutter**, AndorMCD will cause the output voltage from the AndorMCD Card to go 'low' to open the shutter.

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 ALSO SHUTTER MODE AND SHUTTER TRANSFER TIME.

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 function allows 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 actually be set by 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** function.

AndorMCD 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** function.

Figures 5 & 6 on the next page show the timing sequence for both Internal and External triggering modes.

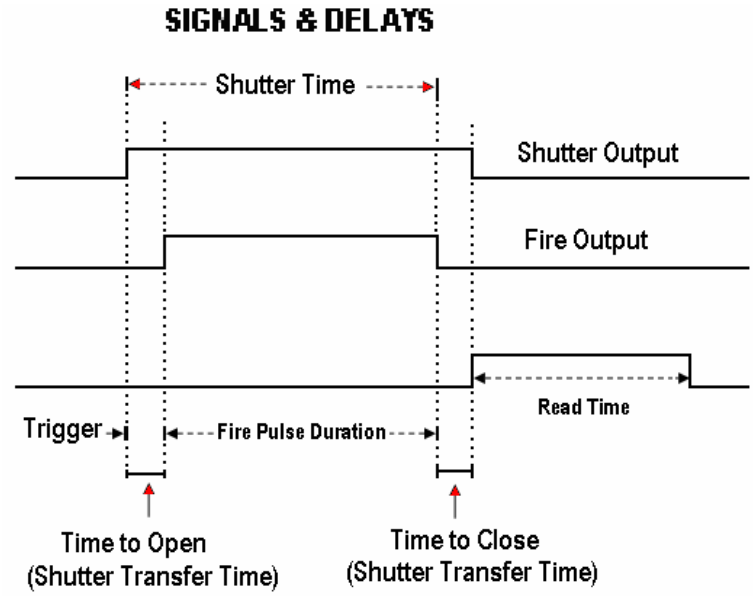


FIGURE 5 - TIMING DIAGRAM FOR SHUTTER AND FIRE PULSES IN INTERNAL TRIGGER MODE

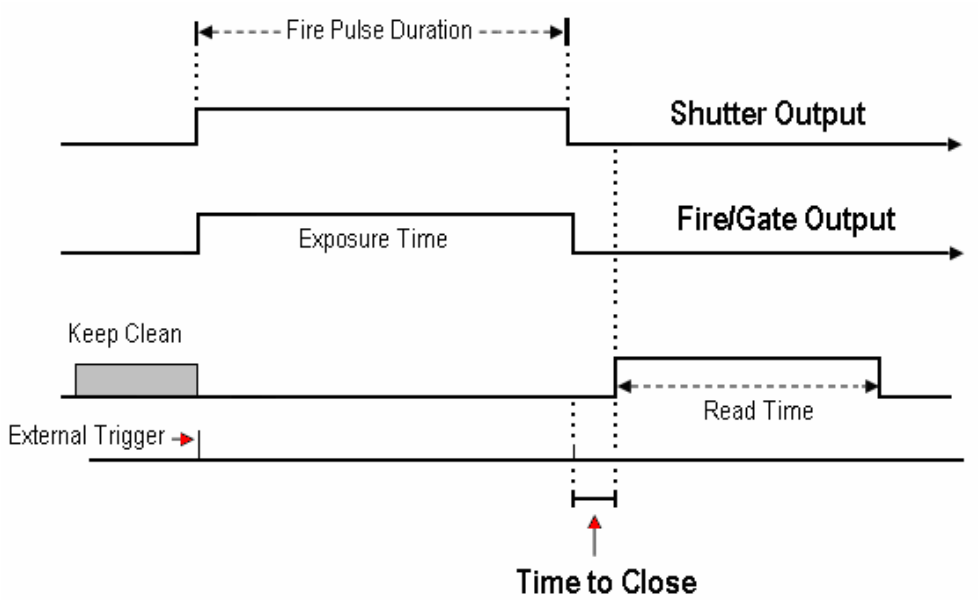


FIGURE 6 - TIMING DIAGRAM FOR SHUTTER AND FIRE PULSES IN EXTERNAL TRIGGER MODE

NOTE: 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 ANDORMCD 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.

IF YOU DO NOT HAVE A SHUTTER CONNECTED, SET THE CLOSING TIME AND OPENING TIME PARAMETERS TO 0. SETTING THESE VALUES TO ANY OTHER VALUE WILL INSERT EXTRA DELAYS INTO CYCLE TIME CALCULATIONS.

SECTION 8 - TEMPERATURE CONTROL

The AndorMCD system 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 AndorMCD allows you to cool and monitor the CCD temperature through software. 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 show 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

NOTE: BECAUSE RAPID COOLING AND HEATING CAN CAUSE THERMAL STRESSES IN THE CCD THE RATE OF COOLING AND HEATING IS REGULATED TO BE LESS THAN 10°C PER MINUTE. ON CLASSIC & ICCD SYSTEMS, THIS REGULATION IS CARRIED OUT BY THE MICROPROCESSOR ON THE PLUG-IN CARD. FOR IDUS, IXON & NEWTON SYSTEMS, IT IS CARRIED OUT "ON-BOARD".

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.

IF THE GETTEMPERATURE FUNCTION RETURNS THE DRV_TEMP_STABILIZED STATUS FLAG THEN THE TEMPERATURE IS WITHIN 3 DEGREES 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 now possible to control more than one Andor camera in an application. Some new functions have been introduced which return the number of available cameras and allow the user to choose which one they currently wish to control*. These functions are as follows:

- **GetAvailableCameras**
- **GetCameraHandle**
- **SetCurrentCamera**
- **GetCurrentCamera**

***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.

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, USB CAMERAS CANNOT SIMULTANEOUSLY ACQUIRE DATA.**
4. **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.**

The example pseudo code on the next page (figure 9) 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 9 - 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 (e.g. retrieving all new images and updating a display while in run till abort mode) or once the acquisition is complete (e.g. retrieving all the images after a kinetic series has completed).

It is also important to determine when a scan has taken place and when using the the GetStatus function when new image data is available. This can be done using the **WaitForAcquisition** function or by using event handles.

When an acquisition is started and the **WaitForAcquisition** function is called it does not return until the camera notifies the computer that new image data is available. The function can be cancelled by calling the **CancelWait** function although this will require the function 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 10 - EXAMPLE OF WaitForAcquisition PSEUDO CODE

The **SetDriverEvent** function can be used in conjunction with event handles. If an event is created using the 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 returns the total number of images the camera has acquired and 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
SetUserEvent(hEvent)

// Start the acquisition
StartAcquisition()

// Wait for the acquisition to complete
WaitForSingleObject()

// Retrieve data
...

// Shut down camera
ShutDown()

// End of program
```

FIGURE 11 - 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 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. **GetImages** should be used along with the **GetNumberNewImages** function.

The **GetNumberNewImages** function returns the indexes of the oldest and newest images currently 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 ca nbe 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 psedo code for this capability is shown in figure 12 on the next page.

```
//
// 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 12 - HORIZONTAL PIXEL SHIFT PSEUDO CODE EXAMPLE

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 readout speeds (which have low readout times) 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 (noise) in your signal. In general, only the very highest readout 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 psedo code for this capability is shown in figure 13 on the next page.

```
//
// 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 13 - VERTICAL PIXEL SHIFT PSEUDO CODE EXAMPLE

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. (More information about **GetCapabilities** is available in **section 11**).

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

SECTION 10 - EXAMPLES

INTRODUCTION

We present here a number of examples of controlling AndorMCD 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 AndorMCD 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 AndorMCD system to acquire data with the appropriate AndorMCD 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 AN EXAMPLE AND THE MAIN ANDORMCD SOFTWARE, RUNNING AT THE SAME TIME.

RUNNING THE EXAMPLES

C

The C examples are supplied as ready to run executable files and with complete source code. The code has been tested with Microsoft VC++ 5.0 and Borland 5.02.

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 Import Library (Borland compatible)

ATMCD32M.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 4.0 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.

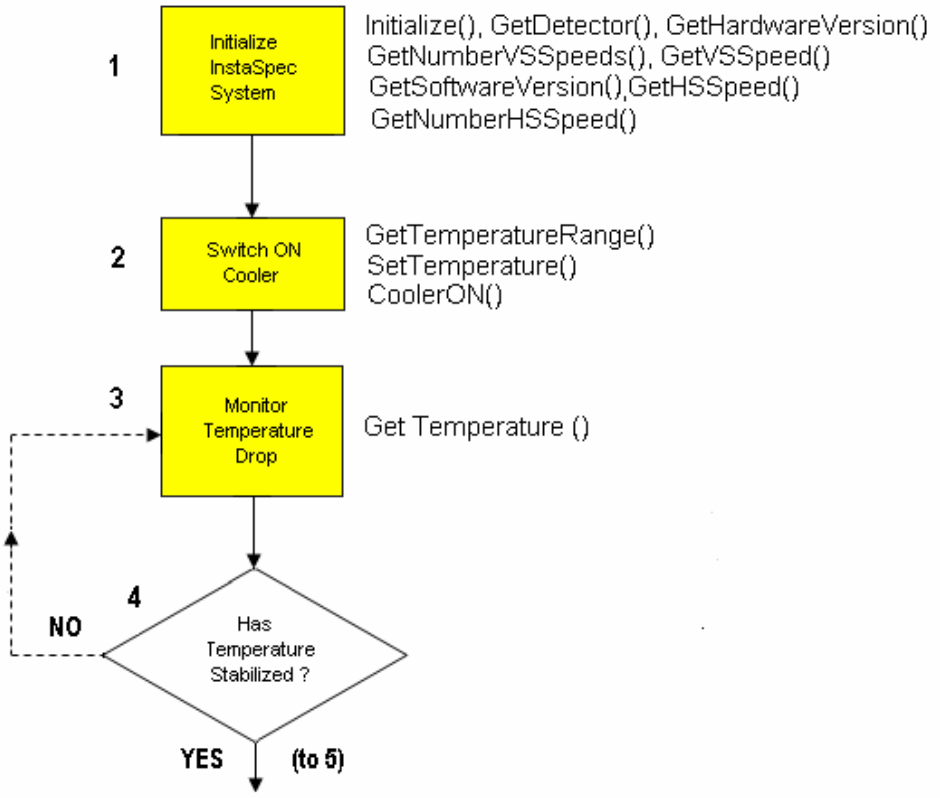
When building you own projects you must include the file **ATMCD32D.BAS**. This file contains the AndorMCD function prototypes for interfacing with the dynamic link library **ATMCD32D.DLL**.

NOTE: TO RUN ANY OF THE EXAMPLES YOU WILL NEED THE FOLLOWING FILES:

ATMCD32D.DLL

DETECTOR.INI: Contains initialization information (not required on iDus, iXon or Newton systems)

FLOW CHART OF THE FUNCTION CALLS NEEDED TO CONTROL THE ANDOR MCD



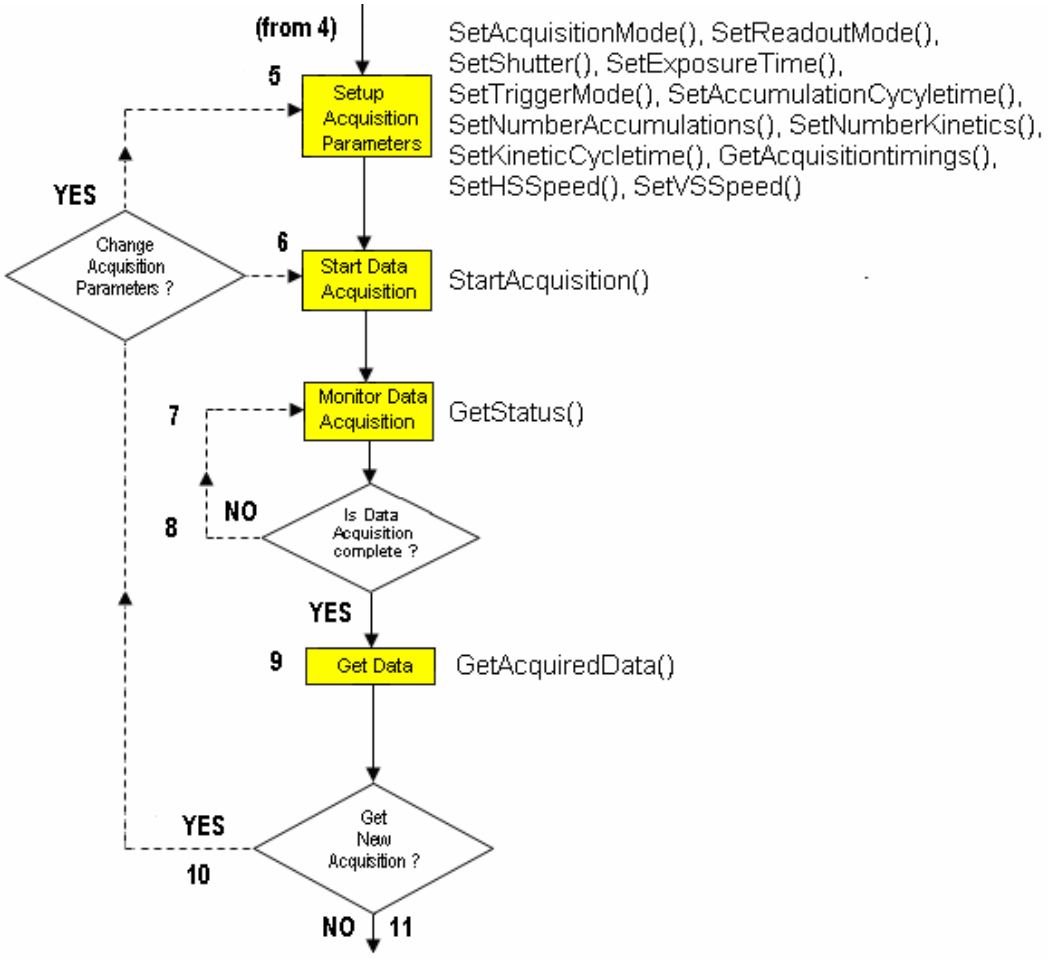
1. The application initializes the AndorMCD system and then obtains information relating to the capabilities of the system.

NOTE: THE ANDORMCD HARDWARE TAKES 2 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.), read out mode (full vertical binning etc.) and the trigger mode (Internal etc.).

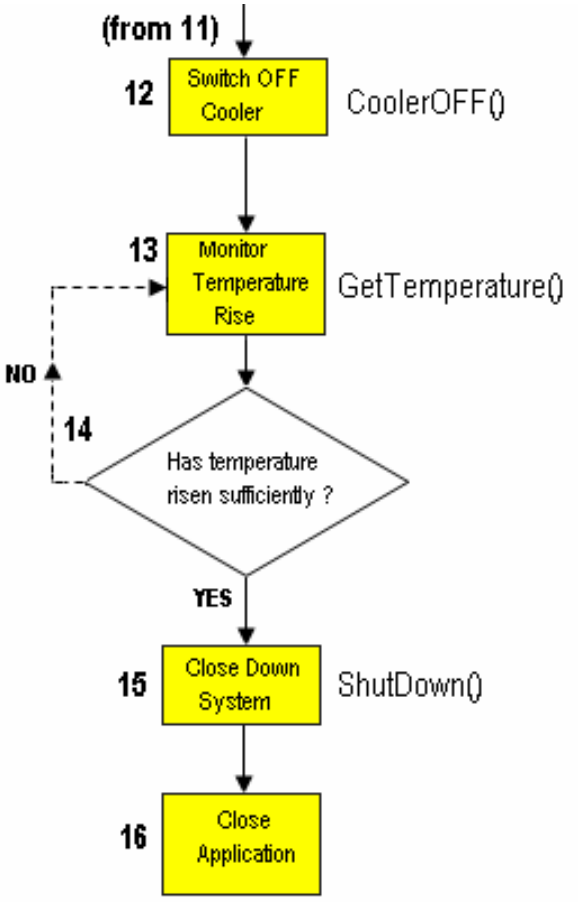
6. You are now ready to start an acquisition.

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 AndorMCD 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. After the temperature has risen above **-20˚C** (Classic & ICCD or **-50˚C** (iDus & iXon), the user may shut down the AndorMCD 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 AndorMCD System. It includes the taking of a single FVB scan for completeness. This example is an expanded version of Example1.

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 signaled by the AndorSDK 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 AndorMCD 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 the third 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

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

NOTE: 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.

When closing down the program via ShutDown you must ensure that the temperature of the detector is above 0ºC, otherwise calling ShutDown while the detector is still cooled will cause the temperature to raise faster than certified.

CoolerON

unsigned int WINAPI CoolerON(void)

Description Switches ON the cooling. 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, but 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.

FreeInternalMemory

unsigned int WINAPI FreeInternalMemory(void)

Description The FreeInternalMemory function will deallocate any memory used internally to store the acquisition 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 GetImage, PrepareAcquisition

GetAcquiredData

unsigned int WINAPI GetAcquiredData(long* array, 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 long* array: 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 is too small.

See also GetStatus, StartAcquisition, GetAcquiredData16

GetAcquiredData16

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

Description 16-bit version of the GetAcquiredData function.

Parameters WORD* array: 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 is too small.

See also GetAcquiredData

NOTE: Filter modes are not available when using this function.

GetAcquisitionProgress

unsigned int WINAPI GetAcquisitionProgress(long* accum, 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. If 0 is returned for both *accum* and *series* then no acquisition is currently running.

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* accum: returns the number of accumulations completed in the current kinetic scan. float* accumulate: 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 read out 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

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

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

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: index to desired camera (set to 0)

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

NOTE: 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.

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. Below is a summary of the capabilities currently returned.

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

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

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

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

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 mono color 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_GAIN
Description: Software Gain through the SetGain function is available.
Bit: 3
State: 1

Capability: AC_SETFUNCTION_EMCCDGAIN

Description: Software Gain through the SetEMCCDGain function is available.

Bit: 4

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

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 instead of RAM 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

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

Return unsigned int

DRV_SUCCESS	Capabilities returned.
DRV_P1INVALID	Invalid caps parameter (i.e. NULL).

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

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.

NOTE: Available in iStar.

GetDDGPulse

unsigned int WINAPI GetDDGPulse(double region, 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 region: 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.

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.
DRV_ACQUIRING	Acquisition in progress.

GetFastestRecommendedVSSpeed

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

Description As your AndorMCD 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 (which have low readout times), 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

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* exposure)

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* exposure: 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 As your AndorMCD 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: Superseded by GetFKVShiftSpeedF.

GetFKVShiftSpeedF

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

Description As your AndorMCD 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

GetHardwareVersion

unsigned int WINAPI GetHardwareVersion(unsigned int* PCB, unsigned int* Flex, unsigned int* dummy1, unsigned int* dummy2, unsigned int* dummy3, unsigned int* dummy4)

Description	This function returns the Hardware version information.	
Parameters	Unsigned int* PCB: Plug-in card version	
	unsigned int* Flex: Flex 10K file version	
	unsigned int* dummy1	
	unsigned int* dummy2	
	unsigned int* dummy3	
	unsigned int* dummy4	
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* model)

Description	This function will retrieve the type of CCD attached to your system.	
Parameters	char* model: A user allocated array of characters for storage of the Head Model.	
Return	unsigned int	
	DRV_SUCCESS	Name returned.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.

GetHorizontalSpeed

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

Description As your AndorMCD 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: Superseded by GetHSSpeed.

GetHSSpeed

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

Description As your AndorMCD 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 (in MHz on idus, iXon & Newton).

Parameters int channel: the AD channel.

int type: output amplification.

Valid values: 0 electron multiplication.

1 conventional.

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 microseconds per pixel shift (in MHz on iXon).

Return unsigned int

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

See also GetNumberHSSpeeds, SetHSSpeed

NOTE: Speed is returned in MHz. If not using an iDus, iXon or Newton type is ignored.

GetImages

unsigned int WINAPI

GetImages(long first, long last, long* array, 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.
long* array: 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 too small.
DRV_NO_NEW_DATA	There is no new data yet.

See also GetImages16, GetNumberNewImages

GetImages16

unsigned int WINAPI
GetImages16(long first, long last, WORD* array, 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* array: 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 too small.
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

GetMostRecentImage

unsigned int WINAPI GetMostRecentImage(long* array, 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* array: 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

GetMostRecentImage16

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

Description 16-bit version of the GetMostRecentImage function.

Parameters WORD* array: 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

GetNewData

unsigned int WINAPI GetNewData(long* array, unsigned long size)

Description 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 long* 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 too small.
DRV_NO_NEW_DATA	There is no new data yet.

See also SetDriverEvent, GetAcquisitionProgress, SetAcquisitionMode, , GetNewData8, GetNewData16

NOTE: DEPRECATED BY FUNCTIONS - GetImages, GetMostRecentImage and GetOldestImage.

GetNewData16

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

Description 16-bit version of the GetNewData function.

Parameters WORD* 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 too small.
DRV_NO_NEW_DATA	There is no new data yet.

NOTE: DEPRECATED BY FUNCTIONS - GetImages, GetMostRecentImage and GetOldestImage.

GetNewData8

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

Description 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* 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 too small.
DRV_NO_NEW_DATA	There is no new data yet.

NOTE: DEPRECATED BY FUNCTIONS - GetImages, GetMostRecentImage and GetOldestImage.

GetNumberADChannels

unsigned int WINAPI GetNumberADChannels(int* number)

Description	As your AndorMCD system may be capable of operating with more than one A-D converter, this function will tell you the number available.	
Parameters	int* number: number of allowed channels	
Return	unsigned int	
	DRV_SUCCESS	Number of channels returned.
See also	SetADChannel	

GetNumberAmp

unsigned int WINAPI GetNumberAmp(int* number)

Description	As your AndorMCD system may be capable of operating with more than one output amplifier, this function will tell you the number available.	
Parameters	int* number: number of allowed channels	
Return	unsigned int	
	DRV_SUCCESS	Number of channels returned.
See also	SetOutputAmplifier	

GetNumberFKVShiftSpeeds

unsigned int WINAPI GetNumberFKVShiftSpeeds(int* number)

Description	As your AndorMCD 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	Speed returned.
	DRV_NOT_INITIALIZED	System not initialized.
	DRV_ACQUIRING	Acquisition in progress.
See also	GetFKVShiftSpeedF, SetFKVShiftSpeed	

GetNumberHorizontalSpeeds

unsigned int WINAPI GetNumberHorizontalSpeeds(int* number)

Description As your AndorMCD 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	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.

See also GetHorizontalSpeed, SetHorizontalSpeed

NOTE: Superseded by GetNumberHSSpeeds.

GetNumberHSSpeeds

unsigned int WINAPI GetNumberHSSpeeds(int channel, int type, int* number)

Description As your AndorMCD 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 type: output amplification.

Valid values:	0	electron multiplication.
	1	conventional.

int* number: number of allowed horizontal speeds

Return unsigned int

DRV_SUCCESS	Speed returned.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid channel.
DRV_P2INVALID	Invalid horizontal read mode

See also GetHSSpeed, SetHSSpeed

NOTE: If not using an iDus, iXon or Newton, type is ignored.

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 no longer can 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

GetNumberPreAmpGains

unsigned int WINAPI GetNumberPreAmpGains(int* number)

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 gains available. The functions GetPreAmpGain and SetPreAmpGain can be used to specify which of these gains is to be used.

Parameters int* number: 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

NOTE: Only for iDus, ixon & Newton. Should return 1 for other models.

GetNumberVerticalSpeeds

unsigned int WINAPI GetNumberVerticalSpeeds(int* number)

Description	As your AndorMCD system maybe 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: Superseded by GetNumberVSSpeeds .

GetNumberVSAmplitudes

unsigned int WINAPI GetNumberVSAmplitudes (int* number)

Description	**This function will normally return 5 for an iXon and 0 for every other camera.**
Parameters	int *number:
Return	unsigned int
	DRV_SUCCESS
	DRV_NOT_INITIALIZED
	DRV_ACQUIRING

GetNumberVSSpeeds

unsigned int WINAPI GetNumberVSSpeeds(int* number)

Description	As your AndorMCD 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	GetVSSpeed, SetVSSpeed, GetFastestRecommendedVSSpeed	

GetOldestImage

unsigned int WINAPI GetOldestImage(long* array, 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 long* array: 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* array, unsigned long size)

Description 16-bit version of the GetOldestImage function.

Parameters WORD* array: 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

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 will be applied if the gain specified by index is selected. 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 required

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

NOTE: Available in iDus, iXon & Newton. . Should return 1 for other models.

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	Size of circular buffer returned.
	DRV_NOT_INITIALIZED	System not initialized.

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	This function will return information on the progress of the current spool operation. The value returned is the number of ".dat" files 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	

GetStatus

unsigned int WINAPI GetStatus(int* status)

Description This function will return the current status of the drivers. 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	Driver status return
DRV_NOT_INITIALIZED	System not initialized

See also SetTemperature, StartAcquisition

NOTE: If the status is DRV_ACCUM_TIME_NOT_MET, DRV_KINETIC_TIME_NOT_MET, DRV_ERROR_ACK or 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.

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.

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

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.

GetVerticalSpeed

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

Description As your AndorMCD system maybe 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: Superseded by GetVSSpeed.

GetVSSpeed

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

Description As your AndorMCD system maybe 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 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 I²C 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	System fully initialized.
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.

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	System fully initialized.
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.

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	System fully initialized.
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.

I2CReset

unsigned int WINAPI I2CReset(void)

Description This function will reset the I^2C data bus.

Parameters

Return unsigned int

DRV_SUCCESS	System fully initialized.
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.

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	System fully initialized.
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.

InAuxPort

unsigned int WINAPI InAuxPort(int port, int* state)

Description This function returns the state of the TTL Auxiliary Input Port on the AndorMCDplug-in card.

Parameters int port: Number of AUX in port on AndorMCD card

Valid Values 1 to 4

int* state: current state of port

0	OFF/LOW
all other	ON/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* directory)

Description This function will initialize the AndorMCD System. As part of the initialization procedure on some cameras (i.e. Classic, Istar and earlier iXion) the **DLL** will need access the following file:

DETECTOR.INI which contains information relating to the detector head, number pixels, readout speeds etc.

Parameters char* directory: Path to the directory containing the files

Return unsigned int

DRV_SUCCESS	System fully initialized.
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.

NOTE: This function must be called first to ensure that the system is fully initialized.

IsPreAmpGainAvailable

unsigned int WINAPI IsPreAmpGainAvailable(int channel, int amplifier, int index, int gain, 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 gain:	PreAmp gain index.
int* status:	0: PreAmpGain not available.
	1: PreAmpGain Available.

Return unsigned int

DRV_SUCCESS	Function found if PreAmpGain is available.
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

OutAuxPort

unsigned int WINAPI OutAuxPort(int port, int state)

Description This function sets the TTL Auxiliary Output port (P) on the AndorMCD plug-in card to either ON/HIGH or OFF/LOW.

Parameters int port: Number of AUX out port on AndorMCD 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 clears 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 clear 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.

See also StartAcquisition, FreeInternalMemory,

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.

See also SaveAsSif

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, which can be read in by the main application. 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 **SaveAsSif SetSifComment**

NOTE: The comment used in Sif files created with this function is discarded once the call completes. ie future calls to SaveAsSif will not use this comment. To set a persistent comment use the SetSifComment function.

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.

See also GetStatus, StartAcquisition , SetSifComment, SaveAsCommentedSif

---

SaveAsTiff

unsigned int WINAPI SaveAsTiff(char* path, char* palette, long position, long type)

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 type 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.

long position: The number in the series, should be 1 for a single scan.

long type: 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

See also

SaveAsSif, SaveAsBmp

SetAccumulationCycleTimeS

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. See section on Acquisition Modes for further details.

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

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:	0,6,7,8	Reserved DO NOT USE
	1	Single Scan
	2	Accumulate
	3	Kinetics
	4	Fast Kinetics
	5	Run till abort
	9	Time Delayed Integration (requires special files)

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.

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. It will be used for subsequent acquisitions.	
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	

SetBaselineClamp

unsigned int WINAPI SetBaselineClamp(int active)

Description	This function turns on and off the baseline clamp functionality within the SDK. With this feature enabled the baseline level of each scan in a kinetic series will be more consistent across the sequence.	
Parameters	int active: 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_P1INVALID	State parameter was not zero or one.

SetCoolerMode

unsigned int WINAPI SetCoolerMode(int mode)

Description This function determines whether the fan is switched off when an application ends.

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.

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 used when acquiring in Random 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

SetDDGGain

unsigned int WINAPI SetDDGGain(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. Gain values between 0 and 255 are permitted.

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 SetGateMode, SetMCPGating

NOTE: Available on iStar.

SetDDGGateStep

unsigned int WINAPI SetDDGGateStep(double step)

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: 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

SetDDGInsertionDelay

unsigned int WINAPI SetDDGInsertionDelay(int delay)

Description This function controls the length of the insertion delay.

Parameters int delay: 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 gating)

Description This function controls the MCP gating. Not available when the fast insertion delay option is selected.

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	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 integrate)

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

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 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 a, double b)

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 (a*exp(b*n))
	2	Logarithmic (a*log(b*n))
	3	Linear (a + b*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 type)

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 type: 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.

SetDMAParameters

(CONTINUED)

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 event)

Description This function passes a Win32 Event handle to the driver via which the driver can inform the main software that something has occurred. For example the driver can "set" the event when an acquisition has completed thus relieving the main code of having to continually pole the driver 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 is during an acquisition is completed.

3) Temperature as stabilized, drifted from stabilization or could not be reached.

Condition 1 can be tested via GetStatus while 2 via GetTemperature.

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 event: 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 GetStatus GetTemperature GetAcquisitionProgress

NOTE , NOT all programming environments allow the use of multiple threads and Win32 events.

SetEMCCDGain

unsigned int WINAPI SetEMCCDGain(int gain)

Description Allows the user to change the amplitude of clock voltages thereby amplifying the signal. Gain values between 0 and 255 are permitted.

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.

NOTE: Only available on EMCCD sensor systems.

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. See section on Acquisition Modes for further details.

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: 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 GetAcquisitionTimings.

SetFanMode

unsigned int WINAPI SetFanMode(int mode)

Description Allows the user to control the mode of the iXon 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 40ºC, 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.

NOTE: Available only on iDus, iXon & Newton.

SetFastKinetics

unsigned int WINAPI SetFastKinetics(int height, int series, float exptime, int mode, int dummy1, int dummy2)

Description This function will set the parameters to be used when taking a fast kinetics acquisition.

Parameters
int height: sub-area height in rows.
int series: number in series.
float exptime: exposure time in seconds.
int mode: binning mode (0 – vertical binning ON, 4 – vertical binning OFF).
int dummy1: Unused parameter, set equal to 1.
int dummy2: Unused parameter, set equal to 1.

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	Currently this value must be set to 1.
DRV_P6INVALID	Currently this value must be set to 1.

See also SetFKVShiftSpeed

SetFastKineticsEx

unsigned int WINAPI SetFastKineticsEx(int height, int series, float exptime, int mode, int hbin, int vbin, int offset)

Description This function will set the parameters to be used when taking a fast kinetics acquisition with an iXon.

Parameters int height: sub-area height in rows.

int series: number in series.

float exptime: 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	Currently this value must be set to 1.
DRV_P6INVALID	Currently this value must be set to 1.
DRV_P7INVALID	Offset not within CCD limits

See also SetFKVShiftSpeed

NOTE: Available for all systems except iDus. On some systems, hbin, vbin = 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 is 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

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	Horizontal speed set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Index is out off range.

See also GetNumberFKVShiftSpeeds, GetFKVShiftSpeedF

SetFrameTransferMode

unsigned int WINAPI SetFrameTransferMode (int mode)

Description This function will set whether an acquisition will readout in Frame Transfer Mode

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.

NOTE: Only available if CCD is a Frame Transfer chip.

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

SetGain

unsigned int WINAPI SetGain(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. Gain values between 0 and 255 are permitted.

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	I^2C command timed out.
DRV_I2CDEVNOTFOUND	I^2C device not present.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Gain value invalid.

See also SetGateMode, SetMCPGating

NOTE: Available on some ICCD models.

SetGate

unsigned int WINAPI SetGate(float delay, float width, float step)

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: 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 mode)

Description Allows the user to control the photocathode gating mode.

Parameters int mode: 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 SetGain, SetMCPGating

NOTE: Available on some ICCD models.

SetHighCapacity

unsigned int WINAPI SetHighCapacity(int state)

Description This function turns on and off the high capacity functionality within the SDK. With this feature 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

0 – Disable High Capacity

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.

NOTE: Available on some iXon models.

SetHorizontalSpeed

unsigned int WINAPI SetHorizontalSpeed(int index)

Description 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: Superseded by SetHSSpeed.

SetHSSpeed

unsigned int WINAPI SetHSSpeed(int type, int index)

Description 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 type: output amplification.

Valid values: 0 electron multiplication.

1 conventional.

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

NOTE: If not using an iXon or Newton, type is ignored.

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

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. See section on Acquisition Modes for further details.

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

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 SetGain, SetGateMode

NOTE: Available on some ICCD models.

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

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	

SetNextAddress

unsigned int WINAPI SetNextAddress(long* data, long lowAdd, long highAdd, long size, long type)

Description This function will set the location of the memory address for the next acquisition. If type is set to virtual (0) a pointer to the virtual address of a buffer is required in data whilst for type physical (1) a physical address is needed. The low 32 bits of the physical address should be passed in lowAdd and the high 32 bits in highAdd.

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

int lowAdd: low 32-bit of 64-bit physical memory address.

int highAdd: high 32-bit of 64-bit physical memory address.

long size: total number of bytes of data.

long type:

0	Virtual address
1	Physical address

Return unsigned int

DRV_SUCCESS	Memory address accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ERROR_MAP	Physical address could not be mapped.
DRV_ERROR_BUFFSIZE	Size less than or equal to 0
DRV_ERROR_MDL	Unable to generate memory descriptor list.
DRV_NOT_SUPPORTED	Function not supported for this operating system

See also SetStorageMode SetUserEvent UnMapPhysicalAddress

NOTE: In order to use the physical address for the buffer, the physical address must be specified as a translated bus address (i.e. bus-relative device memory range has been mapped to system memory space). The physical address is then specified as two 32bit unsigned values in the lowAdd and highAdd parameters. Failure to provide a translated bus address in this format may produce unexpected results.

The code paths for passing in a physical address have not, as yet, been fully tested and may require further tweaking in future.

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

NOTE: For iidus, Ixon & Newton, in Kinetic Series mode this should be set to 1 accumulation.

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

SetOutputAmplifier

unsigned int WINAPI SetOutputAmplifier(int type)

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 type: the type of output amplifier.

0 – Standard EMCCD gain register (default).

1 – Conventional CCD register.

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: Available in iXon & Newton.

SetPhotonCounting

unsigned int WINAPI SetPhotonCounting(int state)

Description This function activates the photon counting option on ICCD's.

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

NOTE: Available on iStar.

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

NOTE: Available on iStar.

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 in iDus , iXon and Newton.

SetRandomTracks

unsigned int WINAPI SetRandomTracks(int number, int* tracks)

Description This function will set the random-track parameters. The position of each track is individually specified as shown below. The horizontal binning for each track is automatically set to 1.

The positions of the tracks are validated to ensure that the tracks are in increasing order and that there is a "gap" between each track.

Example:

Tracks specified as 20 30 31 40 is invalid as the first track ends at row 30 and the second track starts at 31, i.e. NO GAP. A valid series would be 20 30 32 40.

Parameters int number: number tracks

Valid values 1 to number of vertical pixels/2

int* tracks: 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.

See also SetCustomTrackHBin, SetReadMode, StartAcquisition

NOTE: Not available on iXon.

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		

SetShutter

unsigned int WINAPI SetShutter(int type, int mode, int closingtime, int openingtime)

Description This function sets the shutter parameters.

Parameters int type:

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 type.
DRV_P2INVALID	Invalid mode.
DRV_P3INVALID	Invalid time to open.
DRV_P4INVALID	Invalid time to close.

NOTE: The “opening time” and the “closing time” can be different.

SetSifComment

unsigned int WINAPI SetSifComment(char* comment)

Description This function will set the user text that will be added to any sif files created later 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 SaveAsCommentedSif function.

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	

SetSpool

unsigned int WINAPI SetSpool(int active, int method, char* name, int framebuffersize)

Description This function will enable and disable the spooling of acquired data to the hard disk.

- With **spooling method** set to **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. The file names are constructed by taking *name* as the stem to which the series position is appended, together with ".dat". The data is stored in row order starting with the row nearest the readout register. The user must read the data from the hard disk themselves to access it.

The GetAcquiredData function will NOT return the acquired data.

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

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.

char*name: 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

SetStorageMode

unsigned int WINAPI SetStorageMode(int mode)

Description This function will control whether data is transferred to RAM or to a user defined memory address

Parameters int mode: trigger mode

Valid values:

0	RAM (default)
1	Memory address

Return unsigned int

DRV_SUCCESS	Storage mode set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Storage mode invalid.
DRV_NOT_SUPPORTED	Function not supported for this operating system

See also SetUserEvent SetNextAddress

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	Accumulations set.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_ACK	Unable to communicate with card.
DRV_P1INVALID	Temperature invalid.

See also CoolerOFF, CoolerON, GetTemperature, GetTemperatureF, GetTemperatureRange

SetTriggerMode

unsigned int WINAPI SetTriggerMode(int mode)

Description This function will set the trigger mode to either Internal, External or External Start

Parameters int mode: trigger mode

Valid values:

0	Internal
1	External
6	External Start (only valid in Fast Kinetics mode)

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 SetFastExtTrigger

SetUserEvent

unsigned int WINAPI SetUserEvent(HANDLE event)

Description This function passes a Win32 Event handle to the driver via which the driver can inform the main software that an acquisition has completed and data has been transferred to memory. 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 event: Win32 event handle.

Return unsigned int

DRV_SUCCESS	Event handle accepted.
DRV_NOT_INITIALIZED	System not initialized.
DRV_ACQUIRING	Acquisition in progress.
DRV_ERROR_NOHANDLE	Invalid handle.
DRV_NOT_SUPPORTED	Function not supported for this operating system

See also SetStorageMode SetNextAddress

SetVerticalSpeed

unsigned int WINAPI SetVerticalSpeed(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 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: Superseded by SetVSSpeed.

SetVSAmplitude

unsigned int WINAPI SetVSAmplitude(int amplitude)

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 readout speeds are likely to benefit from an increased vertical clock voltage amplitude.

Parameters int amplitude: 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_ACQUIRING	Acquisition in progress.
DRV_P1INVALID	Invalid amplitude parameter.

NOTE: Available in iXon only.

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: The temperature of the detector should be above -20ºC before shutting down the system.

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

UnMapPhysicalAddress

unsigned int WINAPI UnMapPhysicalAddress (void)

Description	If using physical addresses, the user mode application must explicitly use the UnMapPhysicalAddress function to free the address range specified by the physical address before sending another physical address to the device driver.	
Parameters	NONE	
Return	unsigned int	
	DRV_SUCCESS	Memory address accepted.
	DRV_NOT_INITIALIZED	System not initialized.
See also	SetStorageMode SetUserEvent SetNextAddress	

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

Parameters NONE

Return unsigned int

DRV_SUCCESS	Acquisition Event occurred
DRV_NO_NEW_DATA	Non-Acquisition Event occurred.(e.g. CancelWait() called)

See also StartAcquisition, CancelWait

SECTION 12 – ERROR CODES

CODE	NUMBER
DRV_ERROR_CODES	20001
DRV_SUCCESS	20002
DRV_VXDNOTINSTALLED	20003
DRV_ERROR_SCAN	20004
DRV_ERROR_CHECK_SUM	20005
DRV_ERROR_FILELOAD	20006
DRV_UNKNOWN_FUNCTION	20007
DRV_ERROR_VXD_INIT	20008
DRV_ERROR_ADDRESS	20009
DRV_ERROR_PAGELOCK	20010
DRV_ERROR_PAGEUNLOCK	20011
DRV_ERROR_BOARDTEST	20012
DRV_ERROR_ACK	20013
DRV_ERROR_UP_FIFO	20014
DRV_ERROR_PATTERN	20015
DRV_ACQUISITION_ERRORS	20017
DRV_ACQ_BUFFER	20018
DRV_ACQ_DOWNFIFO_FULL	20019
DRV_PROC_UNKONWN_INSTRUCTION	20020
DRV_ILLEGAL_OP_CODE	20021
DRV_KINETIC_TIME_NOT_MET	20022
DRV_ACCUM_TIME_NOT_MET	20023
DRV_NO_NEW_DATA	20024
DRV_SPOOLERROR	20026
DRV_TEMPERATURE_CODES **or** DRV_TEMP_CODES	20033
DRV_TEMPERATURE_OFF **or** DRV_TEMP_OFF	20034
DRV_TEMPERATURE_NOT_STABILIZED **or** DRV_TEMP_NOT_STABILIZED	20035
DRV_TEMPERATURE_STABILIZED **or** DRV_TEMP_STABILIZED	20036
DRV_TEMPERATURE_NOT_REACHED or DRV_TEMP_NOT_REACHED	20037
DRV_TEMPERATURE_OUT_RANGE **or** DRV_TEMP_OUT_RANGE	20038
DRV_TEMPERATURE_NOT_SUPPORTED **or** DRV_TEMP_NOT_SUPPORTED	20039

CODE	NUMBER
DRV_TEMPERATURE_DRIFT **or** DRV_TEMP_DRIFT	20040
DRV_GENERAL_ERRORS	20049
DRV_INVALID_AUX	20050
DRV_COF_NOTLOADED	20051
DRV_FPGAPROG	20052
DRV_FLEXERROR	20053
DRV_GPIBERROR	20054
DRV_DATATYPE	20064
DRV_DRIVER_ERRORS	20065
DRV_P1INVALID	20066
DRV_P2INVALID	20067
DRV_P3INVALID	20068
DRV_P4INVALID	20069
DRV_INIERROR	20070
DRV_COFERROR	20071
DRV_ACQUIRING	20072
DRV_IDLE	20073
DRV_TEMPCYCLE	20074
DRV_NOT_INITIALIZED	20075
DRV_P5INVALID	20076
DRV_P6INVALID	20077
DRV_INVALID_MODE	20078
DRV_INVALID_FILTER	20079
DRV_I2CERRORS	20080
DRV_COFERROR	20071
DRV_I2CDEVNOTFOUND	20081
DRV_I2CTIMEOUT	20082
DRV_P7INVALID	20083
DRV_USBERROR	20089
DRV_IOCERROR	20090
DRV_NOT_SUPPORTED	20991

SECTION 13 – DETECTOR.INI

DETECTOR.INI EXPLAINED

If present, the configuration file called **Detector.ini** is used to configure both the Andor MCD 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* in the section name.

Two 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 read-out register axis. y is the number of pixel perpendicular to the read-out axis.

DataHShiftSpeed

DataHShiftSpeed = 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**

See CCD diagram for further information.

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

SECTION 14 – FUNCTIONS BY USE

DDG: DIGITAL DELAY GENERATOR (iStar ONLY)

GetDDGIOCPulses
GetDDGPulse
SetDDGGain
SetDDGGateStep
SetDDGInsertionDelay
SetDDGIntelligate
SetDDGIOC
SetDDGIOCFrequency
SetDDGTimes
SetDDGTriggerMode
SetDDGVariableGateStep

EXPORT OPTIONS

SaveAsBmp
SaveAsCommentedSif
SaveAsSif
SaveAsEDF
SaveAsTiff

PRE AMPLIFICATION GAIN (iXon AND iDus ONLY)

GetPreAmpGain
GetNumberPreAmpGains
IsPreAmpGainAvailable
SetPreAmpGain

SYSTEM TEMPERATURE

CoolerOff
CoolerOn
GetTemperature
GetTemperatureF
GetTemperatureRange
SetTemperature

SECTION 15 - ADDENDUMS

Version 2.72

Multiple systems control.

GetAvailableCameras
SetCurrentCamera
GetCurrentCamera
GetCameraHandle

Support for new iDus range of cameras

New data retrieval functions along with 16-bit data versions to reduce memory allocation

GetMostRecentImage
GetOldestImage
GetImages
GetMostRecentImage16
GetOldestImage16
GetImages16

Functions to retrieve information on run till abort mode acquisitions

GetSizeOfCircularBuffer
GetNumberNewImages
GetTotalNumberNewImagesAcquired

Function to track spooling

GetSpoolProgress

Function to retrieve recommended vertical shift speed

GetFastestRecommendedVSSpeed

Function to control of pre-amplifier gain – if available

GetNumberPreAmpGains
GetPreAmpGain
SetPreAmpGain
IsPreAmpGainAvailable

Version 2.72 (CONTINUED)

Offset and binning options available in Fast Kinetics mode on iXon

SetFastKineticsEx

Additional iXon controls

SetBaselineClamp

SetHighCapacity

New export options

SaveAsTiff

SaveAsEDF

Option to determine whether cooler remains on after shutdown

SetCoolerMode

Version 2.71

Photon counting functions added

SetPhotonCounting

SetPhotonCountingThreshold

Functions for preallocating and deallocating acquisition memory.

PrepareAcquisition

FreeInternalMemory

Temperature can now be returned as a float

GetTemperatureF

SaveAsSif now saves temperature, gain, vertical shift speed, vertical clock amplitude and DDG info

Functions for adding user text to sif files when saving.

SaveAsCommentedSif

SetSifComment

ARM signal corrected for external trigger

Version 2.70

Features:

- 16-bit data version of GetAcquiredData available to reduce memory allocation i.e. GetAcquiredData16.

- Functions added to replace current system of getting and setting available horizontal and vertical shift speeds. New functions and what they have replaced:

  GetHSSpeed(): supersedes GetHorizontalSpeed()
  GetNumberHSSpeeds(): supersedes GetNumberHorizontalSpeeds()
  GetNumberVSSpeeds(): supersedes GetNumberVerticalSpeeds()
  GetVSSpeed(): supersedes GetVerticalSpeed()
  SetHSSpeed(): supersedes SetHorizontalSpeed()
  SetVSSpeed (): supersedes SetVerticalSpeed( )

- Functions added to handle multiple A-D converters i.e.
  GetNumberADChannels()
  SetADChannels()

- Function to return the physical pixel dimensions. i.e.
  GetPixelSize()

- Function for setting the amplitude of the Vertical Clock Voltage. i.e.
  SetVSAmplitude()

Version 2.62

Features:

- External Trigger for Frame Transfer CCD's now supported.

Bug Fixes:

- **GetHorizontalSpeed** function restored to taking an integer speed parameter for compatibility reasons.

Version 2.61

Features:

- **SaveAsBmp** command now saves individual scans in a kinetic series to separate files.
- Additional examples have been added covering the following topics:
  - ◊ Spooling to disk
  - ◊ Image Binning
  - ◊ Frame Transfer acquisition mode
  - ◊ EMCCD detectors
  - ◊ DDG (iStar)
  - ◊ Acquisition handling with Events
  - ◊ Kinetic Image
- Full Vertical Binning mode now available for EMCCD's
- Intelligate = 1 added to **detector.ini** file to indicate intelligate capability.
- Accumulate option now available in iStar kinetic series.
- **GetHorizontalSpeed** now takes a float parameter to handle speeds faster than 1µs.

Bug Fixes:

- Multitrack now works with horizontal binning.
- Displacement of image on DV465 systems using horizontal binning fixed.
- Kinetic Series acquisitions greater than 64MB can now be acquired under Windows2000 and NT.
- **SaveAsSif** now saves kinetic series acquisitions correctly.