Apogee
INSTRUMENTS

Apogee Instruments COM (ActiveX) API Specification

Supporting Alta® and Ascent® Camera Platforms (Ethernet and USB Interfaces)

Specification Version 1.4_PRELIM

Revision Date: April, 2007

Disclaimer

Apogee Instruments Inc. assumes no liability for the use of the information contained in this document or the software which it describes. The user assumes all risks. There is no warranty of fitness for a particular purpose, either express or implied.

The information contained in this document is assumed to be correct, but in no event shall Apogee Instruments Inc. be held responsible for typographical errors or changes in the software not reflected in this document.

The specifications contained in this document are subject to change without notice.

Support

The Apogee Alta® and Ascent® Camera Control development specification is provided as a courtesy to our customers, and comes without warranty of fitness for any purpose or application, either express or implied. The user assumes all risk for the use of the information contained in this document and the software it describes.

Apogee Instruments, Inc.
1020 Sundown Way Suite 150
Roseville, CA 95661

(916) 218-7450
(916) 218-7451 (Fax)

Revision History

Document Version	First Applicable Driver Version	Detail
1.0	2.0.0	Initial Version
1.1	2.0.42	• API Change: Added Close() method • API Change: Added ExternalShutter property • API Change: Added Apn_Status_ConnectionError as a return value for the ImagingStatus property • API Change: Deprecated Apn_CameraMode_ExternalShutter as a valid option for the CameraMode property • API Update: FanMode property now defaults to Apn_FanMode_Low (previously was Apn_FanMode_Medium) • API Update: Noted that the default value for the IoPortDirection property, after initialization, is 0x0 • Added "Introduction" section • Added "Image Geometry" section • Added "I/O Port Usage" section • Added "Application Notes" section • Revised C++ sample code • Added VB.NET sample code • Added LabVIEW documentation and sample
1.2	2.0.44	• API Update: Noted that the default value for the *SequenceDelay* property, after initialization, is 327us • API Update: Noted that the default value for the *VariableSequenceDelay* property, after initialization, is TRUE • Added "ISerialPort Methods" section • Added "ISerialPort Properties" section • Added serial port sample code (C++ code) • Added sequencing sample code (C++ code)
1.3	3.1.0	• API Update: For the *CameraMode* property, added the mode *Apn_CameraMode_Kinetics*, and deprecated the *Apn_CameraMode_ExternalTrigger* in favor of the new triggering properties • API Update: Deprecated the *GetLine* method and *Line* property. *GetImage* and *Image* should be used in every case, including TDI line downloads. • API Update: Deprecated *NetworkTransferMode*. There is no current plan to add this feature back to Ethernet systems. • API Update: The *DriverVersion* property now is defined to only provide the version of the Apogee.DLL driver. The new property *SystemDriverVersion* should be used to determine the low-level USB driver (AltaUsb.sys) version number. • API Update: Default value for *TDIRate* after initialization is 0.100s • API Update: Default value for *ShutterStrobePeriod* after initialization is 0.001s • API Update: Default value for *ShutterStrobePosition* after initialization is 0.001s • API Update: The *FastSequence* property cannot be used at the same time as the *TriggerNormalEach* property

		• API Change: Added *CameraSerialNumber* property • API Change: Added *ContinuousImaging* property • API Change: Added *DisableFlushCommands* property • API Change: Added DisablePostExposeFlushing property • API Change: Added *DualReadout* property • API Change: Added *FlushBinningV* property • API Change: Added *InterlineCCD* property • API Change: Added *KineticsSectionHeight* property • API Change: Added *KineticsSections* property • API Change: Added *KineticsShiftInterval* property • API Change: Added *OffsetTwelveBit* property • API Change: Added *SequenceBulkDownload* property • API Change: Added *ShutterCloseDelay* property • API Change: Added *SystemDriverVersion* property • API Change: Added *TDIBinningV* property • API Change: Added *TriggerNormalEach* property • API Change: Added *TriggerNormalGroup* property • API Change: Added *TriggerTdiKineticsEach* property • API Change: Added *TriggerTdiKineticsGroup* property • API Change: Added *Usb8051FirmwareRev* property • API Change: Added *UsbDeviceId* property • API Change: Added *UsbProductId* property
1.4	3.1.5	• Updated entire document to make text less specific to the Alta line, and added information regarding Ascent support • API Update: The *ShowIoDialog* method has been updated to support the Ascent I/O port • API Update: The *ShowLedDialog* method now displays "Apogee LED Selection" in the title bar instead of "Apogee Alta LED Selection". LED selection options have been updated for Ascent. • API Update: The *ShowTempDialog* method now displays "Apogee Temperture Control" in the title bar instead of "Apogee Alta Temperature Control". Fan speed in this dialog cannot be changed on Ascent cameras. • API Update: Revised *DataBits* property for Ascent • API Update: Revised *DualReadout* property for Ascent • API Update: Revised *ExternalIoReadout* for Ascent • API Update: Revised *IoPortAssignment* for Ascent • API Update: Revised *IoPortDirection* for Ascent • API Update: Revised *IoPortData* for Ascent • API Update: Revised *FanMode* for Ascent • API Update: Revised *GainTwelveBit* for Ascent • API Update: Revised *OffsetTwelveBit* for Ascent • API Update: Revised *TempHeatsink* for Ascent • API Change: Added *GuideAbort* method • API Change: Added *GuideDecMinus* method • API Change: Added *GuideDecPlus* method • API Change: Added *GuideRAMinus* method • API Change: Added *GuideRAPlus* method • API Change: Added *ConnectionTest* property • API Change: Added *DataAveraging* property • API Change: Added *DigitizationSpeed* property • API Change: Added *FilterPosition* property

		• API Change: Added *FilterPositioningDone* property • API Change: Added *GuideActive* property • API Change: Added *GuideDecMinusDuration* property • API Change: Added *GuideDecPlusDuration* property • API Change: Added *GuideRAMinusDuration* property • API Change: Added *GuideRAPlusDuration* property • API Change: Added *PlatformType* property

NOTE: For Alta camera systems, many of the new features supported in driver version 3.1.0 and higher require camera control firmware version 21 (or higher). If your camera system does not have version 21 or higher of the firmware, please contact Apogee Instruments for an update. Firmware updates are available online at www.ccd.com.

Apogee Software Reference Documentation
Table of Contents

Apogee Software Reference Documentation

1 Introduction

Thank you for your interest in developing applications for the Apogee Alta and Ascent lines of scientific imaging systems!

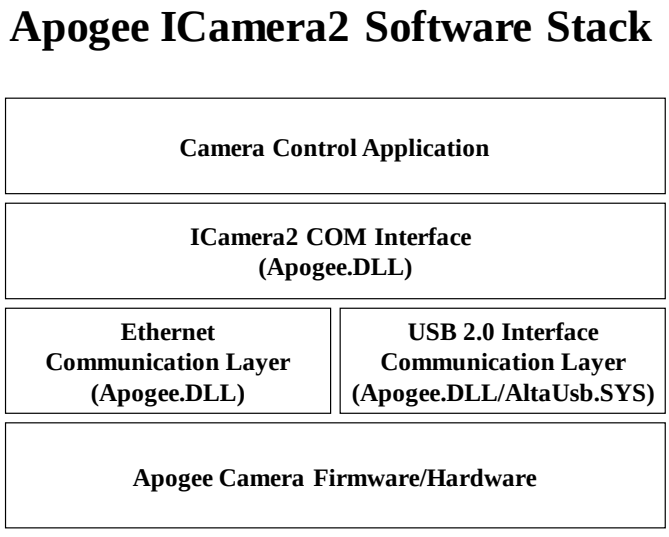
The Apogee camera drivers provide access to all camera functions through a straightforward ActiveX/COM API. ActiveX/COM components are accessible from virtually any Windows programming or scripting language. The driver resides in the file Apogee.dll, which can be installed anywhere on the user's system. Note, though, that the DLL must be registered with the operating system (this is done by software installers automatically, or can be done manually via the command line interface). Please see the installation files for appropriate instruction on hardware and software installation of the Apogee system.

Apogee Alta and Ascent systems are controlled through an interface referred to as the ICamera2 interface. If you have previously developed software for Apogee's AP/KX line of camera systems, you may already be familiar with the previous ICamera interface for control of Apogee cameras. While ICamera code is not forward-compatible with the advanced feature set of the Alta and Ascent lines, developers will find many of the concepts familiar, and porting code from ICamera to ICamera2 should be relatively straightforward.

The ICamera2 interface is composed of various Methods and Properties. Generally speaking, a COM Method is a call made by the application to perform some action, such as taking an exposure. A COM Property is information obtained by the application about the camera system, such as the camera model name.

The ICamera2 interface is designed as a flexible and lightweight layer to access the underlying camera hardware. This approach emphasizes providing the building blocks for application developers, while leaving the process of putting those blocks together to the application writer. It is the simplest interface to fit the widest possible range of applications.

The following diagram shows the various software components and how they fit together:



The remainder of this document will detail the various interfaces supported by the camera systems.

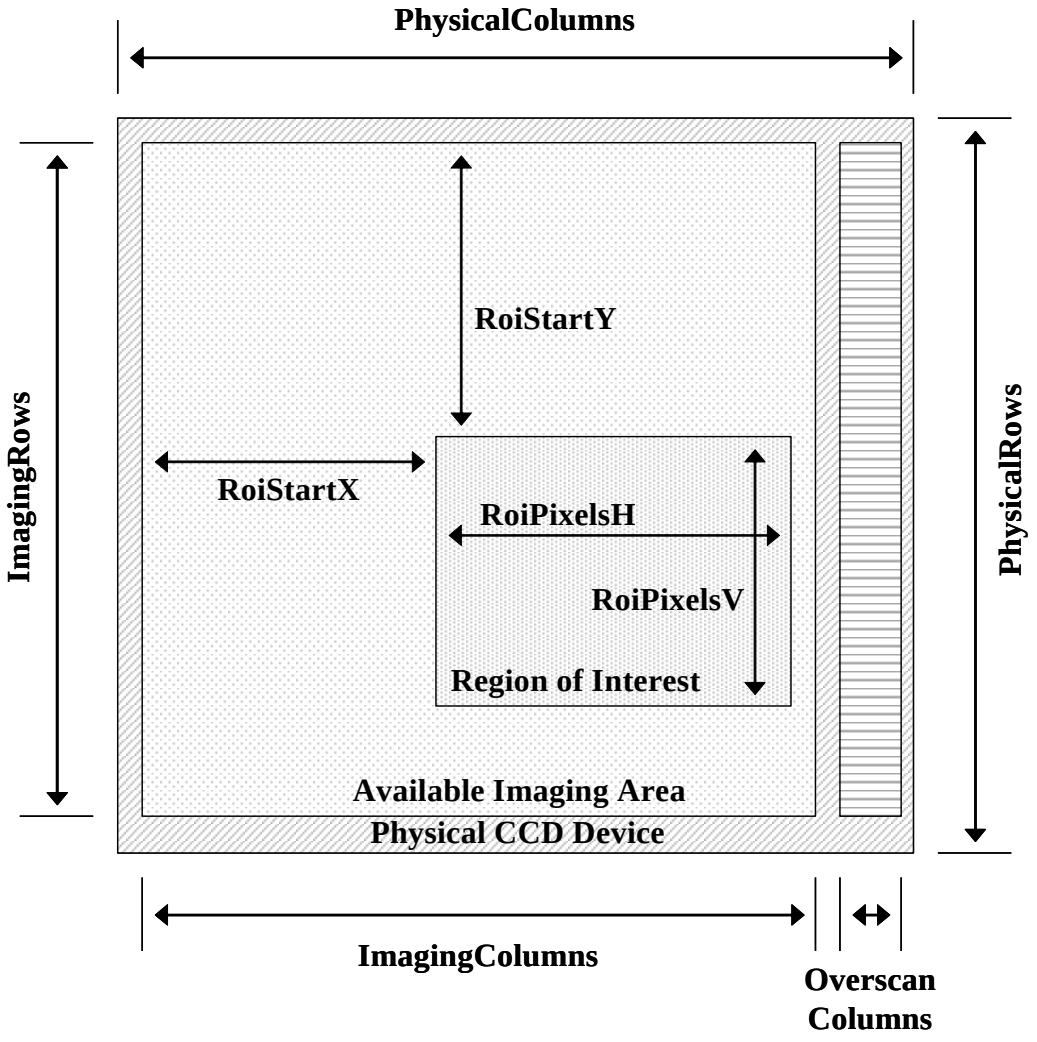
2 Image Geometry

The Apogee Alta and Ascent camera systems require specific geometry parameters in order to properly define a region of interest (ROI) that will contain the digitized image data. The variables that control how the geometry of a particular image is set up are specified in the ActiveX/COM properties later in this document.

The sensor pixels are divided into three regions:

1. Physical CCD Device. This is the actual, complete size of each and every pixel on the CCD sensor. Per manufacturer specs, only a portion of these pixels are available for actually imaging operations.
2. Available Imaging Area. This is the normal, maximum imaging area of the sensor. This area does not include any overscan pixels, since, by definition, overscan pixels are dark reference pixels and not for typical imaging situations.
3. Region of Interest (ROI). This area comprises the actual pixels which will be returned to the application as image data. It may be sized from 1 single pixel to the size of the entire Available Imaging Area, plus the Overscan Columns.

The following diagram may be useful in visualizing the image geometry.



3 ICamera2 Overview by Camera Platform

The following table (organized alphabetically) outlines all of the properties and methods available through the ICamera2 interface, and camera platform support. In general, methods are generally “action oriented” events, while properties usually relate to controlling specific camera settings or features.

3.1 Methods

Method	Description	Alta	Ascent
Close	Close connection to camera	X	X
Expose	Begin the imaging process	X	X
GuideAbort	Stop all guiding/relay operations		X
GuideDecMinus	Activate the Dec- guiding relay		X
GuideDecPlus	Activate the Dec+ guiding relay		X
GuideRAMinus	Activate the RA- guiding relay		X
GuideRAPlus	Activate the RA+ guiding relay		X
GetImage	Downloads image data from the camera	X	X
GetLine	Downloads a single line of image data from the camera	Deprecated	
Init	Connects to the camera and initializes to a known state	X	X
PauseTimer	Pause the camera exposure timer	X	X
ResetSystem	Reset the camera system	X	X
ShowIoDialog	Display the I/O port helper dialog box	X	X
ShowLedDialog	Display the LED control helper dialog box	X	X
ShowTempDialog	Display the temperature control helper dialog box	X	X
StopExposure	Stop an exposure that is already in progress	X	X

3.2 Properties

Property	Description	Alta	Ascent
AvailableMemory	Amount of local memory storage in camera	X	X
CameraInterface	Camera to computer connection interface	X	X
CameraMode	Controls specific type of imaging performed	X	X
CameraModel	Apogee model number of the camera	X	X
CameraRegister	Access specific internal camera registers directly	X	X
CameraSerialNumber	OEM serial number	X	X
Color	Specifies whether the sensor is a color sensor	X	X
ConnectionTest	Verifies connection to the camera	X	X
ContinuousImaging	Special mode of a non-stop series of exposures	X	X
ConvertShortToLong	Converts pixel data from 2 bytes to 4 bytes	X	X
CoolerControl	Camera supports cooling	X	X
CoolerBackoffPoint	Programmed delta if set point cannot be reached	X	X
CoolerDrive	Drive level applied to the cooler	X	X
CoolerEnable	Enables cooler operation	X	X

CoolerRegulated	Camera supports regulated cooling	X	X
CoolerSetPoint	Desired temperature setting in degrees Celsius	X	X
CoolerStatus	Current status of the cooler control unit	X	X
DataAveraging	Average two samples/pixel out of the A/D converter		X
DataBits	Specifies 12 or 16 bit per pixel image resolution	X	
DigitizeOverscan	Enables digitization of the overscan area	X	X
DigitizationSpeed	Programmable speed of the A/D converter		X
DisableFlushCommands	Disables subsequent flush commands	X	X
DisablePostExposeFlushing	Prevents the camera from flushing after exposure	X	X
DisableShutter	Disables control of the camera's internal shutter	X	X
DriverVersion	Specifies the version of Apogee.DLL being used	X	X
DualReadout	Enables two A/D units to simultaneously digitize the sensor		X
ExternalIoReadout	Allows an external signal to begin readout of the sensor	X	
ExternalShutter	Allows an externally controlled shutter to begin an exposure	X	X
FanMode	Selects the desired camera fan speed	X	
FastSequence	Enables fast back to back exposures on interline sensors	X	X
FilterPosition	Selects which filter on the wheel to use		X
FilterPositioningDone	Specifies the desired filter position was achieved		X
FirmwareVersion	Version number of the camera control firmware	X	X
FlushBinningV	Vertical binning used for flushing operations	X	X
ForceShutterOpen	Opens the camera shutter	X	X
GainSixteenBit	Approximate 16bit gain of the camera model	X	X
GainTwelveBit	Sets the 12bit gain of the camera model	X	
GuideActive	Specifies whether the guider relays are active/in use		X
GuideDecMinusDuration	Duration of the pulse on the Dec- guider relay		X
GuideDecPlusDuration	Duration of the pulse on the Dec+ guider relay		X
GuideRAMinusDuration	Duration of the pulse on the RA- guider relay		X
GuideRAPlusDuration	Duration of the pulse on the RA+ guider relay		X
Image	Image data from an exposure, expressed as property	X	X
ImageCount	Number of images in a sequence of images	X	X
ImagingColumns	Number of imaging columns that the camera supports	X	X
ImagingRows	Number of imaging rows that the camera supports	X	X
ImagingStatus	Specifies the current operational state of the camera	X	X
InputVoltage	Voltage level being supplied to the camera	X	X
InterlineCCD	Specifies whether the camera sensor is an interline CCD	X	X
IoPortAssignment	Selects how the I/O port lines will be configured	X	X
IoPortData	Data values set to and from the I/O port lines	X	
IoPortDirection	Selects the direction of the I/O port lines	X	
KineticsSectionHeight	Vertical height for a section in Kinetics Mode	X	X
KineticsSections	Number of sections in Kinetics Mode	X	X
KineticsShiftInterval	Interval between shifting sections in Kinetics Mode	X	X
LedA	Indicates the setting of the first LED light	X	X
LedB	Indicates the setting of the second LED light	X	X
LedMode	Specifies whether the LED lights are enabled for use	X	X
Line	A single line of image data	Deprecated	
MaxBinningH	Maximum horizontal binning supported by the camera	X	X
MaxBinningV	Maximum vertical binning supported by the camera	X	X

MaxExposure	Maximum duration of a single exposure	X	X
MinExposure	Minimum duration of a single exposure	X	X
NetworkTransferMode	Type of transfer for sending data over the network	Deprecated	
OffsetTwelveBit	Sets the 12bit offset of the camera model	X	
OptionBase	Changes the image data array base index from 0 to 1	X	X
OverscanColumns	Number of overscan columns outside of the ROI	X	X
PhysicalColumns	Number of actual, physical columns on the sensor	X	X
PhysicalRows	Number of actual, physical rows on the sensor	X	X
PixelSizeX	Width of one pixel, in microns	X	X
PixelSizeY	Height of one pixel, in microns	X	X
PlatformType	Specifies camera platform type (i.e., Alta or Ascent)	X	X
RoiBinningH	Horizontal binning in the ROI area	X	X
RoiBinningV	Vertical binning in the ROI area	X	X
RoiPixelsH	Width of the ROI area	X	X
RoiPixelsV	Height of the ROI area	X	X
RoiStartX	Starting X coordinate of the ROI area	X	X
RoiStartY	Starting Y coordinate of the ROI area	X	X
Sensor	Name of the sensor used in the camera model	X	X
SensorTypeCCD	Specifies if the sensor in the camera is CCD technology	X	X
SerialASupport[1]	Specifies if the camera supports serial port A	X	X
SerialBSupport[1]	Specifies if the camera supports serial port B	X	X
SequenceBulkDownload	Transfers a series of images as a single image	X	X
SequenceCounter	Specifies the number of images digitized in a sequence	X	X
SequenceDelay	Interval between images in a sequence	X	X
ShutterAmpControl	Disables CCD voltage while shutter strobe is active	X	X
ShutterCloseDelay	Interval between end exposure and begin digitization	X	X
ShutterState	Specifies if the camera shutter is open or closed	X	X
ShutterStrobePeriod	Duration of the shutter strobe pulse	X	X
ShutterStrobePosition	Delay from exposure start until shutter signal is pulsed	X	X
SystemDriverVersion	Version information for AltaUsb.sys	X	X
TDIBinningV	Vertical binning used in a TDI image	X	X
TDICounter	Specifies the number of rows digitized in a TDI image	X	X
TDIRate	Interval rate between rows in a TDI image	X	X
TDIRows	Number of rows in a TDI image	X	X
TempCCD	Current temperature of the CCD sensor	X	X
TempHeatsink	Current temperature of the heatsink	X	
TestLedBrightness	Brightness of the test LED	X	X
TriggerNormalEach	Enables hardware triggering in normal exposures	X	X
TriggerNormalGroup	Enables hardware triggering in normal exposures	X	X
TriggerTdiKineticsEach	Enables hardware triggering in TDI/Kinetics exposures	X	X
TriggerTdiKineticsGroup	Enables hardware triggering in TDI/Kinetics exposures	X	X
Usb8051FirmwareRev	Internal USB firmware version information	X	X
UsbProductId	USB Product ID value	X	X
UsbDeviceId	USB Device ID value	X	X
VariableSequenceDelay	Controls how the specified delay in a sequence is used	X	X

[1] On Ascent, the property is supported for information purposes, though the feature is not. See property documentation for details.

4 ICamera2 Methods

4.1 Init

4.1.1 Format:

```
Init( [in] Apn_Interface Interface,
      [in] long CamIdOne,
      [in] long CamIdTwo,
      [in] long Option )
```

4.1.2 Parameters:

Interface: The interface type requested by the application. Valid values are *Apn_Interface_NET*, for Ethernet cameras, and *Apn_Interface_USB*, for USB 2.0 camera systems.

CamIdOne: The first of three camera identifiers. For camera systems using *Apn_Interface_NET*, this identifier is the camera IP address. The IP address is written in standard little endian byte order, so an address of 192.168.0.3 has the value 0xC0A80003. For camera systems using the *Apn_Interface_USB*, this identifier is used to identifying a particular camera, as enumerated by the operating system.

CamIdTwo: The second of three camera identifiers. For camera systems using the *Apn_Interface_NET*, this identifier is the IP address port number of the camera. For camera systems using the *Apn_Interface_USB*, this identifier is not used and should be set to zero (0x0).

Option: Reserved for future use. In the future, this parameter may be used for passing interface-specific information to the driver during Initialization. Currently, this parameter should be set to zero (0x0).

4.1.3 Description:

The *Init()* method is used for initializing the Alta camera system and loading firmware to the device.

4.2 Close

4.2.1 Format:

```
Close()
```

4.2.2 Parameters:

None.

4.2.3 Description:

The *Close()* method is used to explicitly close a connection that was opened to the camera with the *Init()* method.

An application cannot issue further API calls to the camera system until another *Init()* operation is performed.

4.3 ResetSystem

4.3.1 Format:

```
ResetSystem()
```

4.3.2 Parameters:

None.

4.3.3 Description:

The *ResetSystem()* method resets the camera's internal pixel processing engines, and then starts the system flushing again.

This method may be used by an application to attempt to clear an error condition from the device, instead of re-initializing the complete system. This method is not destructive. Programmed camera settings will remain intact after the method is called. An application using *ResetSystem()* as an attempt to clear an error condition should query status after this method is called to check the current state of the camera.

The *ResetSystem()* method does *not* return the camera to the initial state it was in after the *Init()* method was called. Applications wishing to re-initialize the camera system to known state should call the *Init()* method.

4.4 Expose

4.4.1 Format:

```
Expose( [in] double Duration,
        [in] Boolean Light )
```

4.4.2 Parameters:

Duration: Length of the exposure(s), in seconds. The valid range for this parameter is 0.00000256s to 10994.4s

Light: Determines whether the exposure is a light or dark/bias frame. A light frame requires this parameter to be set to "TRUE", while a dark frame requires this parameter to be "FALSE".

4.4.3 Description:

The *Expose()* method begins the imaging process. The following types of imaging categories are begun with this method:

1) Light (Nominal) Frames
2) Dark and Bias Frames
3) TDI Images
4) Image Sequences
5) Triggered Images

The type of exposure taken is dependent on various state variables, which are properties of the ICamera2 interface—these include *TdiMode* and *TriggerMode*.

4.5 PauseTimer

4.5.1 Format:

PauseTimer(Boolean PauseState)

4.5.2 Parameters:

PauseState: A state variable that controls the pausing of the exposure timer. A value of "TRUE" will issue a command to pause the timer. A value of "FALSE" will issue a command to unpause the timer. Multiple calls with this parameter set consistently to either state (i.e. back-to-back "TRUE" states) have no effect.

4.5.3 Description:

The *PauseTimer()* method pauses the current exposure by closing the shutter and pausing the exposure timer.

There is no limit to the length of time that the exposure timer may be paused.

4.6 StopExposure

4.6.1 Format:

```
StopExposure( Boolean Digitize )
```

4.6.2 Parameters:

Digitize: A state variable that controls whether the stopped exposure data will be digitized or discarded by the application. A value of "TRUE" indicates that the application wishes to download the data in the future. A value of "FALSE" indicates the application will not try to retrieve the image data.

4.6.3 Description:

The *StopExposure()* method halts/stops an exposure already in progress, and the hardware begins digitizing the image.

If *Digitize* is set to "TRUE", then an application should follow a call to the *StopExposure()* method with a call to retrieve the image data (i.e. using *GetImage()*). The application then has the option of discarding the image data entirely, or displaying the data from the shortened exposure.

If *Digitize* is set to "FALSE", then an application should not call *GetImage()* after issuing the *StopExposure()* method.

If *StopExposure()* is called, and there is no exposure in progress, the method has no effect.

4.7 GetImage

4.7.1 Format:

```
GetImage(long pImageBuffer)
```

4.7.2 Parameters:

pImageBuffer: Returns a pointer to 16 bit, unsigned short data located in memory. The image data region should be allocated by the application prior to calling this method.

4.7.3 Description:

The *GetImage()* method returns a pointer to a previously-allocated region of memory (allocated by the calling application) that will be filled with image data.

The application must take care to assure that it properly allocates the image memory region before calling this method.

4.8 GetLine

This method is now considered deprecated. Applications should use the *GetImage* method instead. For a streaming TDI downloads, this means calling *GetImage* for each individual line.

4.8.1 Format:

```
GetLine(long pLineBuffer)
```

4.8.2 Parameters:

pLineBuffer: Returns a pointer to 16 bit, unsigned short data located in memory. The image data region should be allocated by the application prior to calling this method.

4.8.3 Description:

The *GetLine()* method returns a pointer to a previously-allocated region of memory that will be filled with line data.

The application must take care to assure that it properly allocates the image memory region before calling this method.

This method should not be used with the *Apn_Interface_NET* interface type. If it is used with this interface, it will fail.

4.9 GuideAbort

4.9.1 Format:

```
GuideAbort()
```

4.9.2 Parameters:

None.

4.9.3 Description:

The *GuideAbort()* method stops any current guider relay activity on any of the relay pins (Dec-/+ and RA-/+). If an application uses this method to stop the relays, it should use the *GuideActive* property to determine when it is "safe" to use the relays again.

4.10 GuideDecMinus

4.10.1 Format:

```
GuideDecMinus()
```

4.10.2 Parameters:

None.

4.10.3 Description:

The *GuideDecMinus()* method activates the Dec- guider relay, for the period of time as specified by the *GuideDecMinusDuration* property. Using this method again before the relay is free will have no impact on the camera.

4.11 GuideDecPlus

4.11.1 Format:

```
GuideDecPlus()
```

4.11.2 Parameters:

None.

4.11.3 Description:

The *GuideDecPlus()* method activates the Dec+ guider relay, for the period of time as specified by the *GuideDecPlusDuration* property. Using this method again before the relay is free will have no impact on the camera.

4.12 GuideRAMinus

4.12.1 Format:

```
GuideRAMinus()
```

4.12.2 Parameters:

None.

4.12.3 Description:

The *GuideRAMinus()* method activates the RA- guider relay, for the period of time as specified by the *GuideRAMinusDuration* property. Using this method again before the relay is free will have no impact on the camera.

4.13 GuideRAPlus

4.13.1 Format:

```
GuideRAPlus()
```

4.13.2 Parameters:

None.

4.13.3 Description:

The *GuideRAPlus()* method activates the RA+ guider relay, for the period of time as specified by the *GuideRAPlusDuration* property. Using this method again before the relay is free will have no impact on the camera.

5 ICamera2 Helper-Dialog Methods

The ICamera2 interface also includes three methods to assist application writers in getting their applications up and running as quickly as possible. These methods invoke modal dialog boxes for encapsulating some of the Alta and Ascent functionality. This allows application writers to concentrate on features specific to their software, instead of creating dialog boxes to display camera features. Of course, many application writers will choose not to use these generic dialog boxes, and any functionality in these dialogs can also be queried through the ICamera2 properties.

5.1 ShowIoDialog

5.1.1 Format:

```
ShowIoDialog()
```

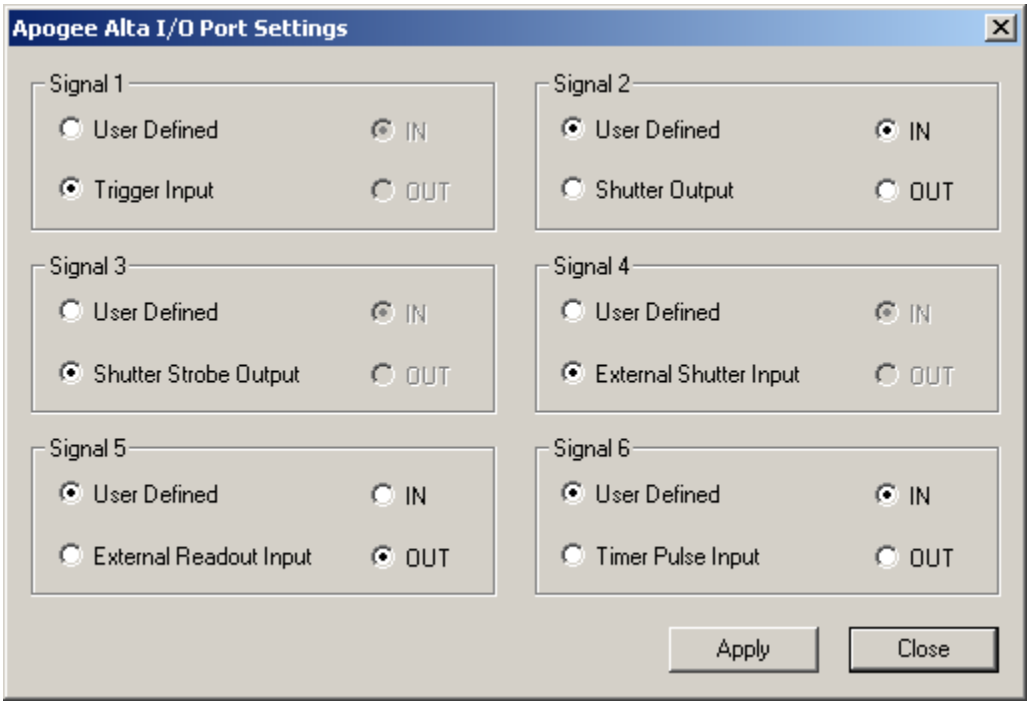
5.1.2 Parameters:

None.

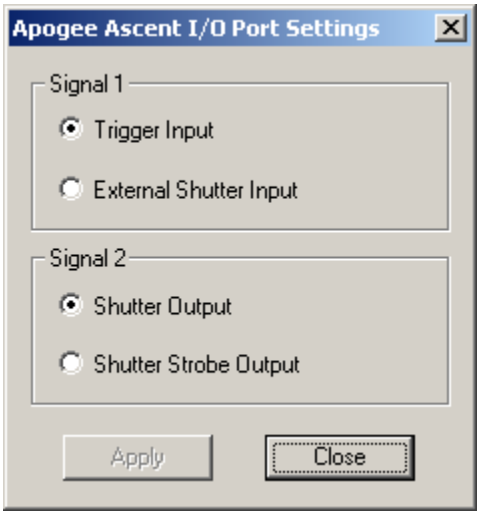
5.1.3 Description:

The *ShowIoDialog()* method can be used to display an I/O selection dialog to the user. The dialog box is a modal dialog. The *ShowIoDialog()* method is not required to access the camera I/O features—please see the various properties relating to camera I/O for that information. This method is intended as a convenience for application writers who do not wish to write their own dialog box to encapsulate this functionality. Depending on the camera platform (Alta versus Ascent) one of two possible dialog boxes will be displayed by the driver.

The following graphic shows the I/O selection dialog for Alta cameras:



The following graphic shows the I/O selection dialog for Ascent cameras:



5.2 ShowLedDialog

5.2.1 Format:

```
ShowLedDialog()
```

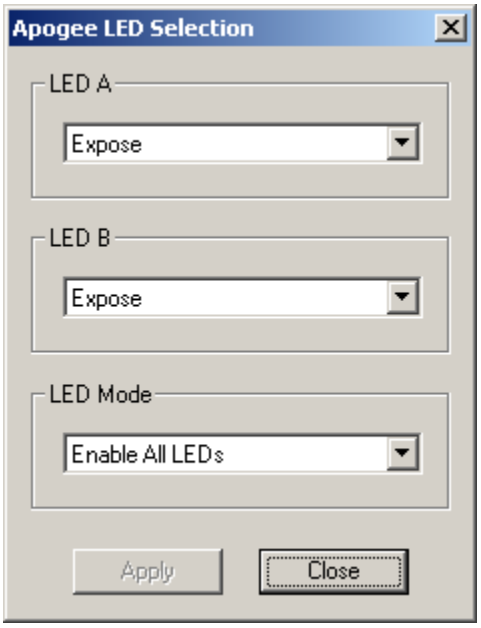
5.2.2 Parameters:

None.

5.2.3 Description:

The *ShowLedDialog()* method can be used to display an LED selection dialog to the user. The dialog box is a modal dialog. The *ShowLedDialog()* method is not required to access the camera LED features—please see the various properties relating to camera LED control for that information. This method is intended as a convenience for application writers who do not wish to write their own dialog box to encapsulate this functionality.

The following graphic shows the LED selection dialog:



5.3 ShowTempDialog

5.3.1 Format:

ShowTempDialog()

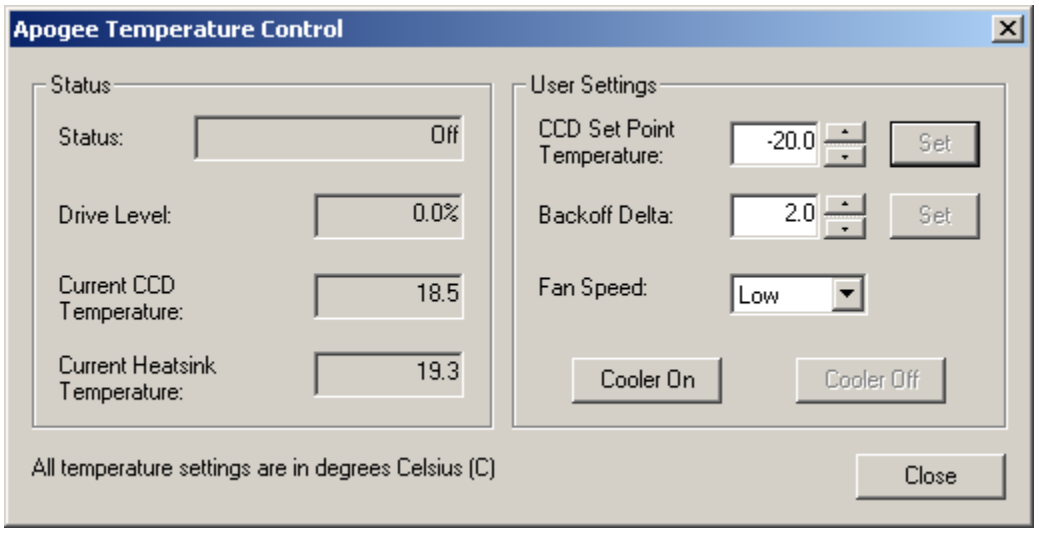
5.3.2 Parameters:

None.

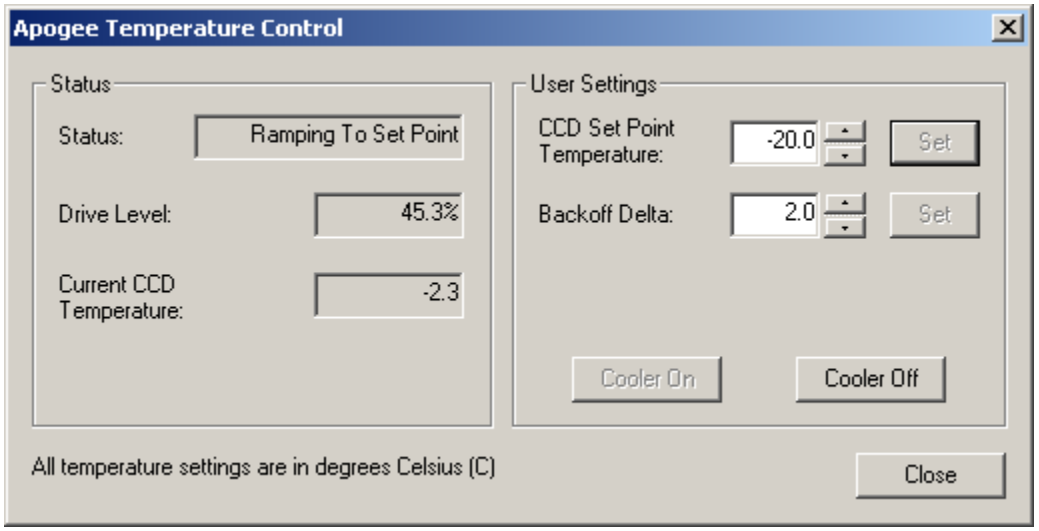
5.3.3 Description:

The *ShowTempDialog()* method can be used to display a temperature control dialog to the user. The dialog box is a modal dialog. The *ShowTempDialog()* method is not required to access the camera temperature control features—please see the various properties relating to camera I/O for that information. This method is intended as a convenience for application writers who do not wish to write their own dialog box to encapsulate this functionality. Depending on the camera platform (Alta versus Ascent) one of two possible dialog boxes will be displayed by the driver.

The following graphic shows the temperature control dialog for Alta cameras:



The following graphic shows the temperature control dialog for Ascent cameras:



6 ICamera2 Properties

The following table lists the ICamera2 properties available to applications. The properties are arranged by functional use within the camera (i.e., functionality related to the cooler is in one place, binning another, et cetera).

Camera Settings			
Variable	R/W	Data Type	Notes
AvailableMemory	RO	Long	Returns the amount of available memory for storing images in terms of kilobytes (KB).
CameraInterface	RO	Apn_Interface	Returns the interface type supported by the camera. Valid values are listed below.
			0x0 (Apn_Interface_NET): Ethernet interface.
			0x1 (Apn_Interface_USB): USB 2.0 interface.
CameraMode	R/W	Apn_CameraMode	Returns/Sets the operational mode of the camera. The default value for this variable after initialization is Apn_CameraMode_Normal.
			0x0 (Apn_CameraMode_Normal): Specifies nominal camera operation for exposure control. Single exposures, or sequences of exposures, are may be initiated by software or hardware control. Applications should note that the ContinuousImaging property is only available in this mode.
			0x1 (Apn_CameraMode_TDI): Specifies camera operation using time delayed integration (drift scan) mode. Used in conjunction with TDIRows, TDIRate and TDIBinningV. The actual TDI exposure is started with the Expose method, but the "Duration" parameter of Expose is not used. This mode cannot be used with interline sensors.
			0x2 (Apn_CameraMode_Test): Specifies that the camera operation should be defined using simulated data for image parameters.
			0x3 (Apn_CameraMode_ExternalTrigger): Specifies camera operation using an external trigger to control the exposure. While maintained for backward compatibility, this mode should be considered deprecated. Applications should use the new TriggerNormal* and TriggerTdiKinetics* properties to enable and use external hardware triggering.
			0x4 (Apn_CameraMode_ExternalShutter): Specifies camera operation using an external shutter to control the exposure. This mode is deprecated. Applications should use the ExternalShutter property instead. Should an application request to use this mode, the driver will change the CameraMode property to be Apn_CameraMode_Normal.
			0x5 (Apn_CameraMode_Kinetics): Specifies camera operation for Kinetics Mode. In this mode, the user will optically mask all but a portion of the CCD. This remaining section is exposed, shifted by some number of rows, and then exposed again. The process continues until the entire CCD surface is exposed. This mode cannot be used with interline sensors.
CameraModel	RO	String	Returns a camera model identifier for the device.
CameraSerialNumber	RO	String	Returns a special OEM-specific serial number—not the serial number assigned by Apogee Instruments for the camera

			system. This property is intended for custom OEM application use only and the programming of specific serial numbers is solely reserved for OEM customers of Apogee Instruments. Cameras without an OEM serial number will display “N/A” or “Unknown” when this property is used.
ConnectionTest	RO	Boolean	Tests the connection between the camera and the user’s computer. Several short test patterns and written and then read back from the camera. If this data is written and read back successfully, the test is considered successful.
DataBits	R/W	Apn_Resolution	Digitization Resolution. Valid values are listed below. The default value for this variable after initialization is Apn_Resolution_SixteenBit. This property cannot be changed for Alta Ethernet systems, or for Ascent cameras.
			0x0 (Apn_Resolution_SixteenBit): Selects resolution of 16 bits per pixel.
			0x1 (Apn_Resolution_TwelveBit): Selects resolution of 12 bits per pixel.
DigitizationSpeed	R/W	Long	Returns/Sets the A/D digitization speed of the camera. This feature is not supported on Alta camera systems. Valid settings on Ascent are listed below. Applications should program the integer index value to select the appropriate speed. The default value after initialization can vary by camera model.
			0: 10MHz
			1: 5MHz
			2: 2.5MHz
			3: 1.25MHz
			4: 0.612MHz
			5: 0.306MHz
			6: 0.153MHz
			7: 0.076MHz
DriverVersion	RO	String	Version number of the camera driver. This is the version of the Apogee.DLL. A return value of <=0 indicates that the driver stack version could not be recognized, and should be treated as an error code by the application. Note that the use of this property does not require a connection to the camera system.
DataAveraging	R/W	Boolean	Returns/Sets the ability to internally average the pixel data within the camera before it is sent back to the user. Pixel data from the sensor is digitized twice and then averaged. This can be used as an additional method of noise reduction. The default value of this variable after initialization is FALSE.
DualReadout	R/W	Boolean	Returns/Sets dual A/D digitization capabilities with the camera system. If a specific camera system does not support dual digitization, this property has no effect. The Alta camera platform does not support this property. The default value of this variable after initialization is FALSE.
FirmwareVersion	RO	Long	Version number of the camera control firmware.
ImagingStatus	RO	Apn_Status	Returns the current imaging state of the camera. Error conditions are noted by negative numbers. The Apn_Status_Idle is a unique state that the camera should never be in once initialization has occurred (the normal camera state is for the camera to be flushing).
			-3: Apn_Status_ConnectionError - Error. An internal error was generated while attempting to communicate with the camera.

			This error may occur when a connection to the camera is attempted and failed, or if the driver detects a failure while communicating with the camera system (for example, if a USB connector is suddenly unplugged). -2: Apn_Status_DataError - Error. An internal error was generated by the camera during image readout and the internal FIFO was hung. Using the Reset() or Init() methods may return the camera to a known, good state. -1: Apn_Status_PatternError - Error. An internal error was generated by the camera during pixel processing. Using the Reset() or Init() methods may return the camera to a known, good state. 0: Apn_Status_Idle - Idle. The camera system is completely idle. Flushing operations have not been started. Applications should typically never see this state after the Init() method has been called. 1: Apn_Status_Exposing - Exposing. An exposure is in progress. 2: Apn_Status_ImagingActive - Imaging Active. The camera is reading out an image, or waiting for an image to begin. While an image is actually being exposed, the status returned will be Apn_Status_Exposing. 3: Apn_Status_ImageReady - Image Ready. The camera has completed an exposure and digitized the image data. Applications should poll this flag before retrieving the image data. Once the image data has been read, the camera will return (in the nominal case) to the Apn_Status_Flushing state. 4: Apn_Status_Flushing - Flushing. The camera system is flushing the sensor. No other operations are in effect. 5: Apn_Status_WaitingOnTrigger - Waiting on Trigger. The camera is waiting for a trigger event to start an exposure.
InputVoltage	RO	Double	Returns the operating input voltage to the camera.
MaxExposure	RO	Double	Returns the maximum exposure duration. This is a hard value. Exposure times sent to the “Expose” method, which are greater than MaxExposure, will be truncated to the value specified by MaxExposure.
MinExposure	RO	Double	Returns the *suggested* minimum exposure duration, based on the camera model’s sensor type, shutter size, et cetera. As this is a suggested duration, the actual exposure time sent to the “Expose” method may be less than the value specified in MinExposure.
NetworkTransferMode	R/W	Apn_NetworkMode	This mode is disabled and considered deprecated. Apogee Instruments has no current or future plans to implement UDP downloading on Ethernet camera systems. Changing this variable has no effect. Used only with Ethernet camera systems. Valid values are listed below. This variable should only be changed when the interface of the camera is Ethernet. Modifying this variable while controlling a USB camera has no effect. The default value of this variable after initialization is Apn_NetworkMode_Tcp. 0x0 (Apn_NetworkMode_Tcp): Selects transfer of the image data via TCP/IP. While the slowest of the transfer types, it can be used over the Internet itself, and is highly reliable. 0x1 (Apn_NetworkMode_Udp): A faster transfer type for use within lightly-loaded local area networks (LANs).

PlatformType	RO	Apn_Platform	Returns the overall platform type of the camera model. This property is valid after initialization/connection with *Init()*. If the Apn_Platform_Unknown identifier is detected, either initialization has not occurred or the platform cannot be determined. This property will return one of the following values:
			0x0 (Apn_Platform_Unknown): Unknown platform type, or initialization/connection with camera not established.
			0x1 (Apn_Platform_Alta): Alta camera platform.
			0x2 (Apn_Platform_Ascent): Ascent camera platform.
SerialASupport	RO	Boolean	Returns whether the camera supports Serial Port A. Note that Ascent cameras do not have serial ports, and so will return FALSE if queried by this property.
SerialBSupport	RO	Boolean	Returns whether the camera supports Serial Port B. Note that Ascent cameras do not have serial ports, and so will return FALSE if queried by this property.
SystemDriverVersion	RO	String	Returns the version number of the AltaUsb.sys camera driver. If the system driver cannot be determined on a USB system, the property will return “Unknown”. This property will always return “N/A” on an Ethernet system.
Usb8051FirmwareRev	RO	String	Returns a revision code for the internal USB firmware within the camera head. This property is mainly designed for Apogee Instruments’ own internal diagnostic tests. This property will always return “N/A” on an Ethernet system.
UsbProductId	RO	Long	Returns the USB Product ID (PID) associated with the camera system. This property is mainly designed for Apogee Instruments’ own internal diagnostic tests.
UsbDeviceId	RO	Long	Returns the USB Device ID (DID) associated with the camera system. This property is mainly designed for Apogee Instruments’ own internal diagnostic tests.
Flush Settings			
Variable	R/W	Data Type	Notes
DisableFlushCommands	R/W	Boolean	Enables/Disables any flushing command sent by the driver (Apogee.DLL) to be recognized or unrecognized by the camera control firmware. This property may be used with DisablePostExposeFlushing to completely stop all flushing operations within the camera. The default value of this variable after initialization is FALSE. WARNING: This is a highly specialized property designed for very unique experiments. Applications and users will not normally need to modify this variable from the default value.
DisablePostExposeFlushing	R/W	Boolean	Enables/Disables the camera control firmware to/from immediately beginning an internal flush cycle after an exposure. This property may be used with DisableFlushCommands to completely stop all flushing operations within the camera. The default value of this variable after initialization is FALSE. WARNING: This is a highly specialized property designed for very unique experiments. Applications and users will not normally need to modify this variable from the default value.
Shutter Settings			
Variable	R/W	Data Type	Notes
DisableShutter	R/W	Boolean	TRUE forces shutter closed and disabled during an exposure; FALSE allows normal operation. Overridden by the value of ForceShutterOpen. The default value of this variable after

			initialization is FALSE.
ExternalIoReadout	R/W	Boolean	When TRUE, the readout of the camera is no longer started by the external shutter. Instead, a separate I/O pin is used to start the readout. The default value of this variable after initialization is FALSE. Note that Ascent cameras do not support starting readout using the I/O port, so writes using this property have no effect on those systems. Ascent cameras will always return FALSE.
ExternalShutter	R/W	Boolean	When TRUE, allows an external shutter to control the start of an exposure. Note that even when using this property, an application should still call the Expose method in order to set up the internal state of the camera correctly. The default value of this variable after initialization is FALSE.
ForceShutterOpen	R/W	Boolean	TRUE forces shutter to open; FALSE allows normal shutter operation (if shutter was previously opened with this command, FALSE will then close the shutter). This property overrides the DisableShutter property. The default value of this variable after initialization is FALSE.
ShutterAmpControl	R/W	Boolean	TRUE disables the CCD voltage while the shutter strobe is high. The default value of this variable after initialization is FALSE.
ShutterCloseDelay	R/W	Double	Returns/Sets the amount of time between the close of the shutter and the beginning of the sensor readout. The default value of this variable after initialization is camera dependent. NOTE: This is a specialized property designed for unique experiments. Applications and users will not normally need to modify this variable from the default value.
ShutterState	RO	Boolean	Returns TRUE if shutter is open; FALSE if closed.
ShutterStrobePeriod	R/W	Double	Sets the period of the shutter strobe appearing on a pin at the experiment interface. The minimum valid value is 45ns and maximum value is 2.6ms (40ns/bit resolution). The default value of this variable after initialization is 0.001s (1ms).
ShutterStrobePosition	R/W	Double	Sets the delay from the time the exposure begins to the time the rising edge of the shutter strobe period appears on a pin at the experiment interface. The minimum valid value is 3.31us and the maximum value is 167ms (2.56us/bit resolution). The default value of this variable after initialization is 0.001s (1ms).
LED Settings			
Variable	R/W	Data Type	Notes
LedMode	R/W	Apn_LedMode	Format of the status LED lights. Must be one of following (Default is Apn_LedMode_EnableAll):
			0x0 (Apn_LedMode_DisableAll): Disable all LEDs
			0x1 (Apn_LedMode_DisableWhileExpose): Disable LEDs during exposure only
			0x2 (Apn_LedMode_EnableAll): Enable LEDs at all times
LedA	R/W	Apn_LedState	Indicates the usage of LED A, which is user-defined by the table below. The default value of this variable after initialization is Apn_LedState_Expose.
			0x0 (Apn_LedState_Expose): Expose
			0x1 (Apn_LedState_ImageActive): Image Active
			0x2 (Apn_LedState_Flushing): Flushing
			0x3 (Apn_LedState_ExtTriggerWaiting): Waiting for external

			trigger
			0x4 (Apn_LedState_ExtTriggerReceived): External Trigger Received
			0x5 (Apn_LedState_ExtShutterInput): External Shutter Input
			0x6 (Apn_LedState_ExtStartReadout): External Start Readout. Note that if this is selected on Ascent systems (where the external start readout feature is not supported), the LED will simply fail to ever illuminate.
			0x7 (Apn_LedState_AtTemp): At Temperature
LedB	R/W	Apn_LedState	Indicates the usage of LED B, as defined by the user. (See table for LedA, above.) The default value of this variable after initialization is Apn_LedState_Expose.
TestLedBrightness	R/W	Double	Controls the brightness/intensity level of the test LED light within the cap of the camera head. Expressed as a percentage from 0% to 100%. The default value of this variable after initialization is 0%.
I/O Port Settings			
Variable	R/W	Data Type	Notes
IoPortAssignment	R/W	Long	Defines the signal usage for the I/O port. On Alta systems, the valid range is for the 6 LSBs, 0x0 to 0x3F. On Ascent systems, the valid range is for the 2 LSBs, 0x0 to 0x3. The default value for this variable after initialization is 0x0. The following shows how the I/O pins/signals are assigned on the two platforms:
			Bit 0 (I/O Signal 1): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used as a trigger input. *Ascent Definition:* A value of zero (0) indicates that this pin will be used as a trigger input. A value of one (1) indicates that this pin will be used as an external shutter input.
			Bit 1 (I/O Signal 2): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used as a shutter output. *Ascent Definition:* A value of zero (0) indicates that this pin will be used as a shutter output. A value of one (1) indicates that this pin will be used as a shutter strobe output.
			Bit 2 (I/O Signal 3): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used as a shutter strobe output. *Ascent Definition:* Permanently set for RA- (Guider)
			Bit 3 (I/O Signal 4): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used as an external shutter input. *Ascent Definition:* Permanently set for RA+ (Guider)
			Bit 4 (I/O Signal 5): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used for starting readout via an external signal.

			Ascent Definition: Permanently set for Dec+ (Guider)
			Bit 5 (I/O Signal 6): *Alta Definition:* A value of zero (0) indicates that the I/O bit is user defined according to the specified IoPortDirection. A value of one (1) indicates that this I/O will be used for an input timer pulse. *Ascent Definition:* Permanently set for Dec- (Guider)
IoPortDirection	R/W	Long	Defines I/O port signal selection. Valid range is for the 6 LSBs, 0x0 to 0x3F. This property defines user-selected I/O port definitions. The I/O signals must have been marked specifically as user defined by the IoPortAssignment property. The default value for this variable after initialization is 0x0. Because of the different I/O port usage on Ascent, this property is not used on that camera platform. On Ascent systems, writes using this property have no effect, and reads always return the value 0x0.
			Bit 0: I/O Signal 1 (0=IN and 1=OUT)
			Bit 1: I/O Signal 2 (0=IN and 1=OUT)
			Bit 2: I/O Signal 3 (0=IN and 1=OUT)
			Bit 3: I/O Signal 4 (0=IN and 1=OUT)
			Bit 4: I/O Signal 5 (0=IN and 1=OUT)
			Bit 5: I/O Signal 6 (0=IN and 1=OUT)
IoPortData	R/W	Long	Data sent to/from the I/O port. Dependent on the I/O port assignment and direction (IoPortAssignment/IoPortDirection). Applications are responsible for toggling bits, i.e., if Bit 2 of the I/O port is specified as an OUT signal, and a 0x1 is written to this bit, it will remain 0x1 until 0x0 is written to the same bit. Valid range of this property is for the 6 LSBs, 0x0 to 0x3F. Because of the different I/O port usage on Ascent, this property is not used on that camera platform. On Ascent systems, writes using this property have no effect, and reads always return the value 0x0.
Filter Wheel Settings			
Variable	R/W	Data Type	Notes
FilterPosition	R/W	Long	Returns/Sets the filter position. The valid range is from 0-7.
FilterPositioningDone	RO	Boolean	Returns TRUE when the filter position specified by the FilterPosition property is reached/ready. Returns FALSE if the selected filter position is not yet in place.
Cooler/Fan Settings			
Variable	R/W	Data Type	Notes
CoolerControl	RO	Boolean	Returns TRUE if the camera supports cooling, FALSE if no cooling is available.
CoolerRegulated	RO	Boolean	Returns TRUE if the camera supports regulated cooling, FALSE if regulated cooling is not available.
CoolerEnable	R/W	Boolean	Returns/Sets the Cooler operation. A value of TRUE will enable Cooler operation, and FALSE will turn the cooler off.
CoolerStatus	RO	Apn_CoolerStatus	Returns the current cooler status
			0x0 (Apn_CoolerStatus_Off): Off (At or near Ambient). No drive applied to the Cooler.
			0x1 (Apn_CoolerStatus_RampingToSetPoint): Ramp to the Set Point specified by the CoolerSetPoint property.
			0x2 (Apn_CoolerStatus_AtSetPoint): At Set Point specified by the CoolerSetPoint property.

			0x3 (Apn_CoolerStatus_Revision): Controller generated temp revision. If the temperature Set Point is revised, the system will continue to return this status code until the next read of the CoolerSetPoint property.
CoolerSetPoint	R/W	Double	Returns/Sets the desired temperature in degrees Celsius. If the Set Point cannot be reached, the Cooler will determine a new Set Point based on the Backoff Point, and change the status to Apn_CoolerStatus_Revision. An application should reread this property to see the new Set Point that the system is using. Once the application rereads this property, the status of Apn_CoolerStatus_Revision will be cleared.
CoolerBackoffPoint	R/W	Double	Returns/Sets the Backoff temperature of the cooler subsystem. The Backoff Point is given in degrees Celsius. If the cooler is unable to reach the Set Point, the Backoff Point is number of degrees up from the lowest point reached. Used to prevent the cooler from being constant driven with max power to an unreachable temperature. The default value of this variable after initialization can vary depending on camera model, but is typically set at 2.0 degrees Celsius.
CoolerDrive	RO	Double	Drive level applied to the temp controller. Expressed as a percentage from 0% to 100%.
FanMode	R/W	Apn_FanMode	Returns/Sets the current fan speed. The default value of this variable after initialization is Apn_FanMode_Low. Because there is no programmable fan speed on Ascent, this property is not used on that camera platform. On Ascent systems, writes using this property have no effect, and reads always return the value 0x0 (Apn_FanMode_Off).
			0x0 (Apn_FanMode_Off): Off
			0x1 (Apn_FanMode_Low): Low
			0x2 (Apn_FanMode_Medium): Medium
			0x3 (Apn_FanMode_High): High
TempCCD	RO	Double	Returns the current CCD temperature in degrees Celsius.
TempHeatsink	RO	Double	Returns the current Heatsink temperature in degrees Celsius. The Ascent camera platform does not support reading the heatsink temperature, and this property will return -255 (an obviously incorrect value) on those systems.
Geometry Settings			
Variable	R/W	Data Type	Notes
PhysicalColumns, PhysicalRows	RO	Long	Returns the total number of physical columns or rows on the CCD. These variables depend upon the particular geometry of the sensor used within the camera.
ImagingColumns, ImagingRows	RO	Long	Returns the imaging area size in terms of unbinned pixels. These variables depend upon the particular geometry of the sensor used within the camera.
OverscanColumns	RO	Long	Returns the number of overscan columns in terms of unbinned pixels. This variable depends upon the particular sensor used within the camera.
DigitizeOverscan	R/W	Boolean	Determines whether the overscan region will ignored or digitized. Only valid when RoiBinningH is set to 1. The default value for this variable after initialization is FALSE.
RoiPixelsH, RoiPixelsV	R/W	Long	Returns/Sets the image/subframe size in terms of binned pixels. The variables are indexed from one (1). When DigitizeOverscan is FALSE, the valid range for RoiPixelsH is

			from 1 to ImagingColumns, and when DigitizeOverscan is TRUE, the valid range for RoiPixelsH is from 1 to ImagingColumns+OverscanColumns. The valid range for RoiPixelsV is from 1 to ImagingRows. The default value of RoiPixelsH after initialization is ImagingColumns, and the default value of RoiPixelsV after initialization is ImagingRows.
RoiStartX, RoiStartY	R/W	Long	Returns/Sets the subframe start position in terms of unbinned pixels. The variables are indexed from zero (0). When DigitizeOverscan is FALSE, the valid range for StartX is from 0 to ImagingColumns-1, and when DigitizeOverscan is TRUE, the valid range for StartX is from 0 to ImagingColumns+OverscanColumns-1. The valid range for StartY is from 0 to ImagingRows-1. The default value of both variables after initialization is 0.
Binning Parameters			
Variable	R/W	Data Type	Notes
FlushBinningV	R/W	Long	Returns/Sets the vertical binning value used during flushing operations. The valid range for this property is between 1 and the corresponding value of MaxBinningV. The default value after camera initialization is sensor-specific. NOTE: This is a specialized property designed for unique experiments. Applications and users will not normally need to modify this variable from the default value.
MaxBinningH, MaxBinningV	RO	Long	Returns the maximum horizontal and vertical binning factors of the device.
RoiBinningH, RoiBinningV	R/W	Long	Returns/Sets the horizontal and vertical binning parameters for an exposure. The valid range for these properties is between 1 and the corresponding value of MaxBinningH (for RoiBinningH) or MaxBinningV (for RoiBinningV). The default value for both variables after initialization is 1. Note that changing the binning values requires the application to recalculate the RoiPixelsH and RoiPixelsV values, which are in terms of binning pixel counts.
TDI Parameters			
Variable	R/W	Data Type	Notes
TDIBinningV	R/W	Long	The vertical binning of a TDI image. The valid range for this variable is between 1 and the corresponding value of MaxBinningV. The default value for this variable after initialization is 1. Modifying this property also changes the value of the KineticsSectionHeight variable.
TDICounter	RO	Long	Dynamically incrementing count during a TDI image. The final value of TDICounter equals TDIRows. Valid range is between 1 and 65535.
TDIRate	R/W	Double	Incremental rate between TDI rows. Range is from 5.12us to 336ms. The default value for this variable after initialization is 0.100s. Modifying this property also changes the value of the KineticsShiftInterval variable.
TDIRows	R/W	Long	Total number of rows in the TDI image. Range is between 1 and 65535. The default value for this variable after initialization is 1. Modifying this property also changes the value of the KineticsSections variable.
Kinetics Parameters			
Variable	R/W	Data Type	Notes

KineticsSectionHeight	R/W	Long	The vertical height for a Kinetics Mode section. Modifying this property also changes the value of the TDIBinningV variable. The default value for this variable after initialization is 1.
KineticsSections	R/W	Long	The number of sections in a Kinetics Mode image. Modifying this property also changes the value of the TDIRows variable. The default value for this variable after initialization is 1.
KineticsShiftInterval	R/W	Double	Incremental rate between Kinetics Mode sections. Range is from 5.12us to 336ms. Modifying this property also changes the value of the TDIRate variable. The default value for this variable after initialization is 0.100s.
Triggering Parameters			
Variable	R/W	Data Type	Notes
TriggerNormalEach	R/W	Boolean	Enables/Disables the use of an external hardware trigger when using Apn_CameraMode_Normal in either a camera sequence (using ImageCount) or if ContinuousImaging is being used. When TRUE, every image in the series, *after the first image*, is triggered with a hardware trigger. Should be used in conjunction with TriggerNormalGroup if the application wants to use a hardware trigger on *every* image of a series. The default value for this variable after initialization is FALSE. Note that this property cannot be used in conjunction with the FastSequence property, since progressive scan is defined as having the least possible time between exposures. However, the FastSequence property may be used with a single trigger to start a series of images (using TriggerNormalGroup).
TriggerNormalGroup	R/W	Boolean	Enables/Disables the use of an external hardware trigger when using Apn_CameraMode_Normal in either a camera sequence (using ImageCount) or if ContinuousImaging is being used. If this property is TRUE, and TriggerNormalEach is FALSE, the external trigger will be used to start an entire series of consecutive images. If this property is TRUE, and TriggerNormalEach is TRUE, then an external hardware trigger will be required for *every* image in a series. The default value for this variable after initialization is FALSE.
TriggerTdiKineticsEach	R/W	Boolean	Enables/Disables the use of an external hardware trigger using Apn_CameraMode_TDI or Apn_CameraMode_Kinetics. If this property is TRUE, every section or slice of a Kinetics Mode image, *after the first slice*, is triggered with a hardware trigger. Should be used in conjunction with TriggerTdiKineticsGroup if the application wants to use a hardware trigger on *every* slice within the image. The default value for this variable after initialization is FALSE.
TriggerTdiKineticsGroup	R/W	Boolean	Enables/Disables the use of an external hardware trigger using Apn_CameraMode_TDI or Apn_CameraMode_Kinetics. If this property is TRUE, and TriggerTdiKineticsEach is FALSE, the external trigger will be used to start an entire kinetics mode image that has already been set up. If this property is TRUE, and TriggerTdiKineticsEach is TRUE, then an external hardware trigger will be required for *every* slice of the kinetics image. The default value for this variable after initialization is FALSE.
Guider/Relay Settings			
Variable	R/W	Data Type	Notes
GuideActive	RO	Boolean	Returns TRUE if any of the guider relays is currently active, and FALSE otherwise.

GuideDecMinusDuration	R/W	Double	Returns/Sets the duration for the Dec- guider relay, specified in units of seconds. The default value for this variable after initialization is 0.
GuideDecPlusDuration	R/W	Double	Returns/Sets the duration for the Dec+ guider relay, specified in units of seconds. The default value for this variable after initialization is 0.
GuideRAMinusDuration	R/W	Double	Returns/Sets the duration for the RA- guider relay, specified in units of seconds. The default value for this variable after initialization is 0.
GuideRAPlusDuration	R/W	Double	Returns/Sets the duration for the RA+ guider relay, specified in units of seconds. The default value for this variable after initialization is 0.
Sequence Parameters			
Variable	R/W	Data Type	Notes
ContinuousImaging	R/W	Boolean	Enables/Disables a continuous series of exposures until stopped. The default value for this variable after initialization is FALSE.
FastSequence	R/W	Boolean	Enables/Disables very fast back to back exposures. Interline CCDs only. (Also referred to as Ratio Mode.) The default value for this variable after initialization is FALSE. Note that this property cannot be used in conjunction with the TriggerNormalEach property, since progressive scan is defined as having the least possible time between exposures. However, the FastSequence property may be used with a single trigger to start a series of images (using TriggerNormalGroup).
ImageCount	R/W	Long	Number of images in an image sequence. For single exposures, this property is simply set to 1. Valid range is between 1 and 65535. The default value of this variable after initialization is 1.
SequenceBulkDownload	R/W	Boolean	Enables/Disables how image data will be retrieved from the camera during a sequence. Ethernet cameras treat this property as Read Only (RO) and the return value is always TRUE. For USB camera systems, this variable is used to determine whether the returned data will be downloaded in bulk, or streamed as it becomes available. By definition, SequenceBulkDownload must be FALSE when the ContinuousImaging property is enabled, since setting this variable TRUE assumes that the number of images in a series is known in advance of starting the exposure. The default value for this variable after initialization is TRUE.
SequenceCounter	RO	Long	Dynamically incrementing count during an image sequence. The final value of SequenceCounter should equal ImageCount. Valid range is between 0 and 65535.
SequenceDelay	R/W	Double	Time delay between images of the sequence. Range is from 327us to 21.42s. Dependent on VariableSequenceDelay. The default value of this variable after initialization is 327us.
VariableSequenceDelay	R/W	Boolean	If TRUE, SequenceDelay is from end of last readout to binning of next image. If FALSE, SequenceDelay is a constant time interval from the beginning of the last exposure to the beginning of the next exposure. The default value of this variable after initialization is TRUE.
CCD Settings			
Variable	R/W	Data Type	Notes

Color	RO	Boolean	Returns TRUE is CCD sensor has color dyes, and FALSE otherwise
GainSixteenBit	RO	Double	Returns the 16bit gain in e-/ADU units. The 16bit gain is for informational purposes only. It is not a programmable value. It should be noted that 16bit gain values will have slight deviation from camera model to camera model. The gain number given here is a generic approximation, based on the sensor within a particular camera model. Applications or users who wish to use the camera gain in some meaningful way, should measure the gain for their particular system, or use the value provided by Apogee Instruments in the camera data sheet.
GainTwelveBit	R/W	Long	A programmable value to select the actual gain being used by the camera. The valid range of this property is from 0-1023. Applications or user who wish to use this property to change the camera gain, should experiment and test different values in order to determine the gain for their particular camera system. The default value for this property after initialization is camera dependent. This property will frequently be used with the OffsetTwelveBit property. Because there is no 12bit digitization on Ascent, this property is not used on that camera platform. On Ascent systems, writes using this property have no effect, and reads always return the value 0x0.
InterlineCCD	RO	Boolean	Returns TRUE if the sensor is an Interline CCD, FALSE otherwise.
OffsetTwelveBit	R/W	Long	A programmable value to select the actual offset being used by the camera. The valid range of this property is from 0-1023. Applications or users who wish to use this property to change the camera offset, should experiment and test different values in order to determine the desired offset for their particular camera system. The default value for this property after initialization is camera dependent. This property will frequently be used with the GainTwelveBit property. Because there is no 12bit digitization on Ascent, this property is not used on that camera platform. On Ascent systems, writes using this property have no effect, and reads always return the value 0x0.
PixelSizeX	RO	Double	Returns the size (width) of the sensor's pixels in micrometers.
PixelSizeY	RO	Double	Returns the size (height) of the sensor's pixels in micrometers.
Sensor	RO	String	Returns the sensor model installed in the camera (I.e. "KAF401E")
SensorTypeCCD	RO	Boolean	Returns TRUE if the sensor is a CCD; FALSE if CMOS
Other			
Variable	R/W	Data Type	Notes
CameraRegister[*Index*]	R/W	Long	Reads or writes data to the camera register specified by *Index*. Applications should rarely (if ever) require use of this property. Also note that not every camera register can be read.
Image	RO	Variant	Returns a 2D SAFEARRAY, of type LONG (4 bytes per element) or INTEGER (2 bytes per element), which contains the image data. The type of data (LONG or INTEGER) returned is controlled by the associated property of ConvertShortToLong.
Line	RO	Variant	Returns a 1D SAFEARRAY of type LONG (4 bytes per element) or INTEGER (2 bytes per element) which contains the image data. The type of data (LONG or INTEGER) returned is controlled by the associated property of ConvertShortToLong.

			This property is considered deprecated. Applications should use the Image property instead. For streaming TDI downloads, this will require calling Image for each line.
ConvertShortToLong	R/W	Boolean	If TRUE, converts unsigned short (2 bytes per element) image data to LONG (4 bytes per element) when using the Image and Line properties. The default value of this variable after initialization is FALSE.
OptionBase	R/W	Boolean	Returns/Sets the array base index for the Image and Line properties. TRUE sets the base index to 1; FALSE sets the base index to 0. The default value of this variable after initialization is FALSE.

7 ICamDiscover

ICamDiscover provides a simple dialog box within the driver, to assist in the user's camera selection. It is a generic component designed to be quickly inserted into an application, and providing most of the arguments to the ICamera2 Init() method.

ICamDiscover contains only a single method, ShowDialog(). This method causes a modal dialog box to appear. The following description applies to this single method of the ICamDiscover interface.

Format:

```
ShowDialog( Boolean Interactive )
```

Parameters:

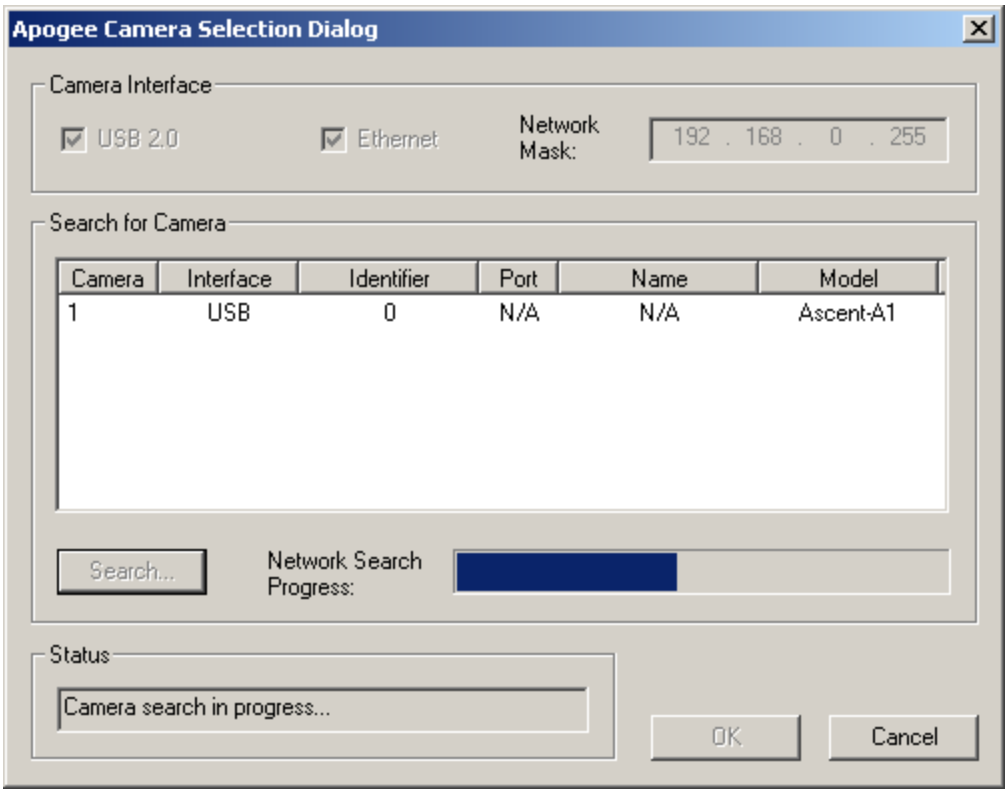
Interactive: A state variable that controls whether the displayed dialog box is interactive or not. A value of "TRUE" indicates that the dialog should be interactive, and the user will be able to select one of the cameras found during the discovery process. A value of "FALSE" indicates that the dialog box will be used only to locate cameras. Applications will almost always set this variable to "TRUE", and use the user's selection to call the ICamera2 Init() method.

Description:

The *ShowDialog()* method displays a dialog box that allows the user to query cameras either directly attached to the computer (USB2) or locally on a network (Ethernet).

The dialog has properties that may be queried or configured to provide a more "custom" look and feel to the dialog box. See the list of ICamDiscover properties for more information.

The current design of this dialog box is shown below. Note that this is a screenshot image of an interactive dialog box, and was invoked using ShowDialog(TRUE).



The “Search” window shows the cameras located. The user selects which interface type should be included in the search (USB 2.0 or Ethernet). If an Ethernet search is request, the user is also queried for a network mask. The default mask is 192.168.0.255, meaning that any host with an address of 192.168.0.*X* will be located.

Properties define the default state of the dialog. The USB 2.0 and Ethernet check boxes may be checked or unchecked before the dialog is display. In addition, the network mask may be changed as well.

ICamDiscover Settings			
Variable	R/W	Data Type	Notes
DlgTitleBarText	R/W	String	Sets the Title Bar of the ICamDiscover dialog box. Default value is “Apogee Alta Camera Selection Dialog”.
DlgCheckEthernet	R/W	Boolean	If TRUE, sets the default state of the Ethernet check box to be enabled (checked). FALSE disables. Setting or unsetting the Ethernet check box will also enable or disable the IP Address field for the network mask. The default value is FALSE.
DlgCheckUsb	R/W	Boolean	If TRUE, sets the default state of the USB 2.0 check box to be enabled (checked). FALSE disables. The default value is FALSE.
DlgNetworkMask	R/W	Long	Returns or sets the value of the network mask field. The default value is 0xC0A800FF, which corresponds to 192.168.0.255.
DlgShowEthernet	R/W	Boolean	If TRUE, the Ethernet check box is displayed in the dialog box. A setting of FALSE will hide the Ethernet check box,

			and remove it from the user interface. This could be used if a particular application is specifically written for a particular type of camera interface. The default value is TRUE.
DlgShowUsb	R/W	Boolean	If TRUE, the USB check box is displayed in the dialog box. A setting of FALSE will hide the USB check box, and remove it from the user interface. This could be used if a particular application is specifically written for a particular type of camera interface. The default value is TRUE.
ValidSelection	RO	Boolean	If TRUE, the user has pressed the “OK” button, and selected one of the cameras in the search box. If TRUE, the values of the Selected* properties (SelectedInterface, SelectedCamIdOne, SelectedCamIdTwo) are valid. On creation of the interface, the default value is FALSE.
SelectedInterface	RO	Apn_InterfaceType	If a valid selection was made, returns the interface type as either Apn_Interface_NET or Apn_Interface_USB. Default value is Apn_Interface_NONE.
SelectedCamIdOne	RO	Long	If a valid selection was made, returns the first camera identifier. For cameras of Apn_Interface_NET, this value is the IP Address of the camera. For Apn_Interface_USB, this value is the order in which the camera was enumerated by the operating system, out of the number of cameras that were detected by the operating system.
SelectedCamIdTwo	RO	Long	If a valid selection was made, returns the second camera identifier. For cameras of Apn_Interface_NET, this value is the Port number of the camera. For Apn_Interface_USB, this value is zero (0x0).
SelectedModel	RO	String	If a valid selection was made, returns a string with the model name of the camera. If the user did not make a valid selection, this property will contain the string “No Model”.

8 I/O Port Usage

8.1 Overview

Alta and Ascent camera systems provide an 8 pin MiniDIN connector, enabling various hardware signals to be controlled by the device. On Alta systems, six of the eight pins are programmable, and of the remaining two pins, one is ground and one is a +12 volt line. Each of the six pins may be programmed to be either a specific fixed-function I/O pin, or else a general purpose and user-defined I/O pin. On Ascent systems, two pins are programmable, and the rest are dedicated to fixed-function relays.

8.2 Hardware Description

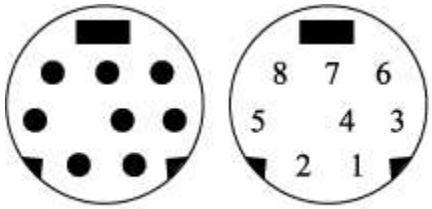
The programmable pins of the I/O port are 3.3V, LVTTL signals.

IMPORTANT: Incoming trigger signals are not de-bounced internally within the camera system. Users should insure a clean signal is sent to the device.

The pin out is shown in the following illustration. The numbers correspond to the I/O pin numbers, defined in the table following the pin-out.

Alta I/O Port:

Pin	Description
1	LVTTL Signal 1
2	LVTTL Signal 2
3	LVTTL Signal 3
4	LVTTL Signal 4
5	LVTTL Signal 5
6	LVTTL Signal 6
7	+12 volt power from the camera head
8	Ground



miniDIN8

Ascent I/O Port:

Pin	Description
1	LVTTL Signal 1
2	LVTTL Signal 2
3	RA- (Guider relay)
4	RA+ (Guider relay)
5	Dec+ (Guider relay)
6	Dec- (Guider relay)
7	Guider Common Ground
8	Ground

8.3 Alta I/O Port Operation

Each LVTTL signal of the I/O Port has two modes of operation. The signal may be used as a general purpose and user-defined I/O pin, or it may be configured to perform a predefined/fixed-function operation. After initialization, the I/O Port defaults to having the signals set as user-defined.

The ICamera2 property, *IoPortAssignment*, is used to control whether each signal is set to the user-defined or predetermined setting. For user-defined signals, the *IoPortDirection* property determines whether the signal is an input or output. Please refer to the documentation for these properties for additional details regarding their operation.

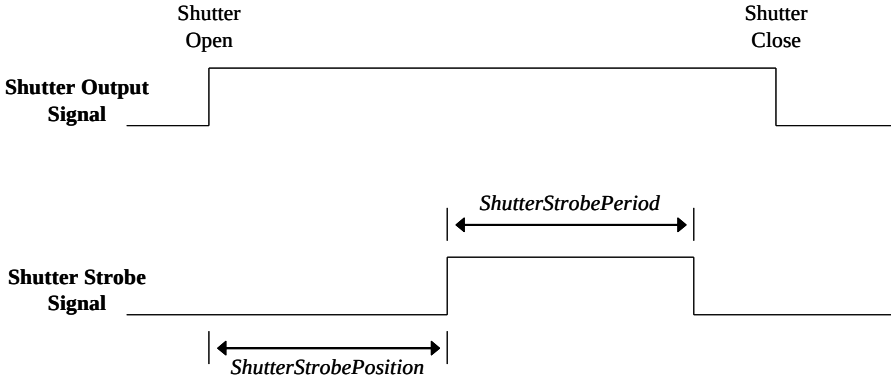
The fixed function descriptions of each pin are as follows:

Pin 1 – **Trigger Input** – Used to initiate triggered exposures (both single exposures and sequences) or TDI row read operations. When the appropriate camera properties are invoked, the ICamera2 interface object will automatically enable this pin to be used as an input for an external trigger signal. Applications should still call the *Expose* method to put the camera system into a state where it is waiting for the trigger to arrive. Triggered exposures use the duration parameter that is specified in the Expose method to program the camera's exposure timer. Internally, the Alta camera will automatically continue flushing the sensor until the triggered exposure begins.

Pin 2 – **Shutter Output** – Enables an output signal that goes high while the shutter is open. This signal could be used by other hardware in an experiment that needs to respond or act when the shutter is open.

Pin 3 – **Shutter Strobe Output** – Enables a programmable pulse, or strobe, to be output on the pin. The duration/period and position of this strobe value is controlled by the *ShutterStrobePeriod* and *ShutterStrobePosition* properties in the ICamera2 interface. The duration/period of the strobe can be anywhere from 45ns to 2.6ms, in increments of 40ns. The position of the shutter strobe after an exposure begins can be between 3.31us and 167ms, in increments of 2.56us.

The following illustration shows both a shutter output signal (Pin 2) as well as a programmable shutter strobe (Pin 3), and their relation to one another:



Pin 4 – **External Shutter Input** – Allows for external control of camera exposures. When using the "External Shutter Input" signal, the exposure duration is entirely controlled by the input to this pin. This differs from a triggered exposure, where the camera's internal exposure timer is used to control the duration of the image. Note however, that applications should still call the *Expose* method in order to properly set up the camera's internal state variables. This signal is used in conjunction with the *ExternalShutter* and *ExternalIoReadout* properties, as well as the "External Readout Start" pin (Pin 5) to provide two unique ways of controlling externally started exposures. Until the trigger is received, applications will get a status of *Apn_Status_WaitingOnTrigger* when querying the *ImagingStatus* property.

In the first mode, the *ExternalShutter* property is set to TRUE, and the rising edge of the "External Shutter Input" signal will halt flushing, open the shutter, and begin the exposure timer. A falling edge of this signal will close the shutter and begin the readout/digitization process.

In the second mode, both the *ExternalShutter* and the *ExternalIoReadout* properties are set to TRUE, and the "External Readout Start" (Pin 5) is used to provide additional flexibility. In this second mode, a rising edge of the "External Shutter Input" pin will halt flushing, open the shutter, and begin the exposure timer. However, when the falling edge of the signal is detected, the shutter closes but readout of the sensor does *not* begin. Readout is begun by using the "External Readout Start" signal. This second mode allows the shutter to be closed as many times as desired during an externally controlled exposure.

The following C++ code shows how to use the camera's external shutter capability:

```
// Query the camera for a full frame image
long ImgXSize = AltaCamera->ImagingColumns;
long ImgYSize = AltaCamera->ImagingRows;

// Allocate memory and calculate a byte count
unsigned short *pBuffer          = new unsigned short[ ImgXSize * ImgYSize ];
unsigned long   ImgSizeBytes     = ImgXSize * ImgYSize * 2;

// External operations
AltaCamera->ExternalShutter          = true;
AltaCamera->ExternalIoReadout        = false;
AltaCamera->IoPortAssignment         = 0x08;

// Even though the exposure time will not be used, still call Expose
AltaCamera->Expose( 0.001, true );

// Check camera status to make sure image data is ready
while ( AltaCamera->ImagingStatus != Apn_Status_ImageReady );

// Get the image data from the camera
AltaCamera->GetImage( (long)pBuffer );
```

Pin 5 – **External Readout Start** – This pin is used in conjunction with the "External Shutter Input" to begin external readout/digitization of an exposure. Use of this fixed function pin requires that the *ExternalIoReadout* property has also been set to TRUE. See the description above for additional details regarding the behavior of this signal.

Developers and users should note that when this mode is enabled, the imaging sensor retains the image data until signaled. Dark current/noise will build until the user begins the digitization process.

Pin 6 – **Timer Pause Input** – The signal can be used to pause the exposure timer for a particular image. When the Alta camera detects that this input signal is high, the shutter will close and the timer will be halted. When the pause signal transitions back to low, the timer is restarted. This signal can be used for blanking events during an exposure.

8.4 Ascent I/O Port Operation

Each of the first two LVTTL signals of the I/O Port has two modes of operation. The ICamera2 property, *IoPortAssignment*, is used to control Pins 1 and 2. After initialization, the I/O Port defaults to being set up for a trigger input on Pin 1 and a shutter output on Pin 2.

Please refer to the documentation of the *IoPortAssignment* property for additional details regarding its operation.

The functional descriptions of each pin are as follows:

Pin 1 – Trigger Input/External Shutter In

Trigger Input: Used to initiate triggered exposures (both single exposures and sequences) or TDI row read operations. When the appropriate camera properties are invoked, the ICamera2 interface object will automatically enable this pin to be used as an input for an external trigger signal. Applications should still call the *Expose* method to put the camera system into a state where it is waiting for the trigger to arrive. Triggered exposures use the duration parameter that is specified in the Expose method to program the camera's exposure timer. Internally, the camera will automatically continue flushing the sensor until the triggered exposure begins.

External Shutter Input: Allows for external control of camera exposures. When using the "External Shutter Input" signal, the exposure duration is entirely controlled by the input to this pin. This differs from a triggered exposure, where the camera's internal exposure timer is used to control the duration of the image. Note however, that applications should still call the *Expose* method in order to properly set up the camera's internal state variables. This signal is used in conjunction with the *ExternalShutter* property to control externally started exposures. Until the trigger is received, applications will get a status of *Apn_Status_WaitingOnTrigger* when querying the *ImagingStatus* property. When the *ExternalShutter* property is set to TRUE, the rising edge of the "External Shutter Input" signal will halt flushing, open the shutter, and begin the exposure timer. A falling edge of this signal will close the shutter and begin the readout/digitization process.

The following C++ code shows how to use the camera's external shutter capability:

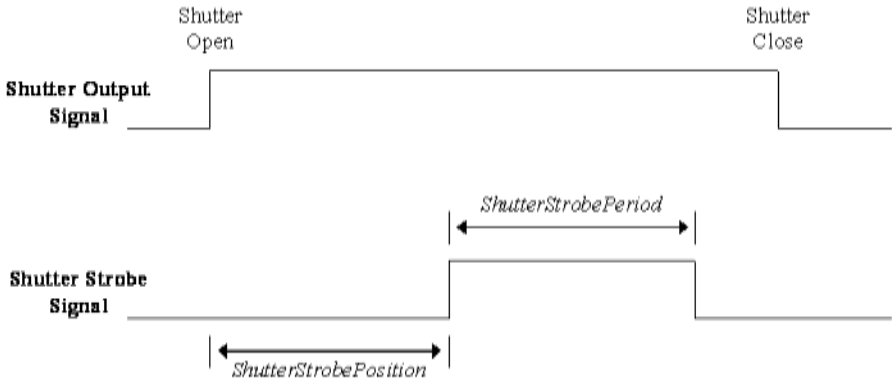
```
// Query the camera for a full frame image
long ImgXSize = AscentCamera->ImagingColumns;
long ImgYSize = AscentCamera->ImagingRows;

// Allocate memory and calculate a byte count
unsigned short *pBuffer          = new unsigned short[ ImgXSize * ImgYSize ];
unsigned long   ImgSizeBytes     = ImgXSize * ImgYSize * 2;

// External operations
AscentCamera->ExternalShutter          = true;
AscentCamera->IoPortAssignment         = 0x01;

// Even though the exposure time will not be used, still call Expose
AscentCamera->Expose( 0.001, true );

// Check camera status to make sure image data is ready
while ( AscentCamera->ImagingStatus != Apn_Status_ImageReady );

// Get the image data from the camera
AscentCamera->GetImage( (long)pBuffer );
```

Pin 2 – **Shutter Output/Shutter Strobe Output**

Shutter Output: Enables an output signal that goes high while the shutter is open. This signal could be used by other hardware in an experiment that needs to respond or act when the shutter is open.

Shutter Strobe Output: Enables a programmable pulse, or strobe, to be output on the pin. The duration/period and position of this strobe value is controlled by the *ShutterStrobePeriod* and *ShutterStrobePosition* properties in the ICamera2 interface. The duration/period of the strobe can be anywhere from 45ns to 2.6ms, in increments of 40ns. The position of the shutter strobe after an exposure begins can be between 3.31us and 167ms, in increments of 2.56us.

The following illustration shows a shutter output signal as well as a programmable shutter strobe, and their relationship to one another:



Pin 3 – **RA- Guider Relay** – Relay line for guiding operations. Used in conjunction with the *GuideRAMinus()* method and the *GuideRAMinusDuration* property.

Pin 4 – **RA+ Guider Relay** – Relay line for guiding operations. Used in conjunction with the *GuideRAPlus()* method and the *GuideRAPlusDuration* property.

Pin 5 – **Dec+ Guider Relay** – Relay line for guiding operations. Used in conjunction with the *GuideDecPlus()* method and the *GuideDecPlusDuration* property.

Pin 6 – **Dec- Guider Relay** – Relay line for guiding operations. Used in conjunction with the *GuideDecMinus()* method and the *GuideDecMinusDuration* property.

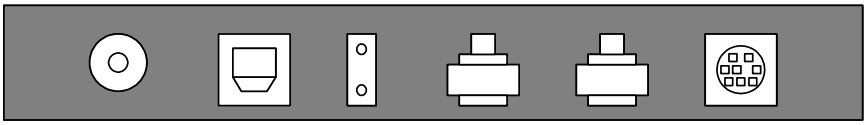
9 Serial Port Usage (Alta Systems)

9.1 Overview

Alta camera systems provide two serial port connectors, enabling RS232 serial devices to be controlled by the camera. This section details the Send, Receive, and Ground signals for serial ports A and B. Control of the serial ports is by the various properties of the *ISerialPort* programming interface.

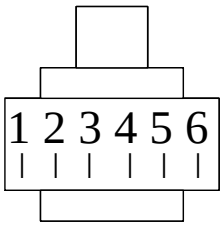
9.2 Hardware Description

The diagram below shows the positions of the two serial ports:



Power **USB** **LEDs** **Serial B** **Serial A** **I/O Port**

The following is the pin-out for each serial connector (there is no difference on the connector pin-out between USB and Ethernet cameras):



For each connector, the signals are arranged as follows:

Pin	Serial A	Serial B
2	Ground	Ground
4	Receive	Receive
5	Transmit	Transmit

10 ISerialPort Methods (Alta Systems)

10.1 OpenPort

10.1.1 Format:

```
Init( [in] Apn_Interface Interface,
      [in] long CamIdOne,
      [in] long CamIdTwo,
      [in] short SerialId )
```

10.1.2 Parameters:

Interface: The interface type requested by the application. Valid values are *Apn_Interface_NET*, for Ethernet cameras, and *Apn_Interface_USB*, for USB 2.0 camera systems.

CamIdOne: The first of three camera identifiers. For camera systems using *Apn_Interface_NET*, this identifier is the camera IP address. The IP address is written in standard little endian byte order, so an address of 192.168.0.3 has the value 0xC0A80003. For camera systems using the *Apn_Interface_USB*, this identifier is used to identifying a particular camera, as enumerated by the operating system.

CamIdTwo: The second of three camera identifiers. For camera systems using the *Apn_Interface_NET*, this identifier is the IP address port number of the camera serial port. For camera systems using the *Apn_Interface_USB*, this identifier is not used and should be set to zero (0x0).

SerialId: The identifier of the particular serial port to open on the camera system. Serial Port A is denoted as 0x0, and Serial Port B is denoted as 0x1.

10.1.3 Description:

The *OpenPort()* method is used for initializing a particular serial port of the Alta camera.

10.2 ClosePort

10.2.1 Format:

```
ClosePort()
```

10.2.2 Parameters:

None.

10.2.3 Description:

The *ClosePort()* method is used to explicitly close a connection to a serial port that was opened with the *OpenPort()* method.

An application cannot issue further API calls to the camera system until another *OpenPort()* operation is performed.

11 ISerialPort Properties (Alta Systems)

Serial Port Settings			
Variable	R/W	Data Type	Notes
BaudRate	R/W	Long	Returns/Sets the baud rate of the serial port. Valid baud rate values are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. The default baud rate after initialization is 9600.
BytesRead	RO	Short	Returns the number of bytes read from the serial port after the previous read operation from the port (i.e., using the *SerialData* property). After initialization of the port, or after a failed read from the port, a value of zero (0) is returned.
FlowControl	R/W	Apn_SerialFlowControl	Returns/Sets the flow control specified for the port. Valid values are listed below. The default value for this variable after initialization of the port is Apn_SerialFlowControl_Off.
			-1 (Apn_SerialFlowControl_Unknown): The flow control state of the port cannot be determined. Almost always implies a communication failure with the camera system. This value cannot be written; it only results from a failed flow control read.
			0x0 (Apn_SerialFlowControl_Off): The port does not/should not use flow control.
			0x1 (Apn_SerialFlowControl_On): The port does/should use flow control.
Parity	R/W	Apn_SerialParity	Returns/Sets the parity specified for the port. Valid values are listed below. The default value for this variable after initialization is Apn_SerialParity_None.
			-1 (Apn_SerialParity_Unknown): The parity state of the port cannot be determined. Almost always implies a commununication failure with the camera system. This value cannot be written; it only results from a failed parity read.
			0x0 (Apn_SerialParity_None): The port does not/should not use parity.
			0x1 (Apn_SerialParity_Odd): The port does/should use odd parity.
			0x2 (Apn_SerialParity_Even): The port does/should use even parity.
SerialData	R/W	String	Returns/Sets a character string of data from/to the serial port.

12 Sequences

12.1 Overview

The camera systems provide the capability to capture internal sequences of images. An internal sequence is defined as two or more images captured by the camera, using a single *Expose* call. In other words, the *ImageCount* property must be greater than or equal to two.

Sequences of images are useful when an application knows the user would like to require a certain number of images in advance, and the application also wants to strictly minimize the delay between successive images. For application and programming simplicity, Apogee Instruments only recommends using sequences when both of these conditions must be met. When the camera's internal sequencing engine is used, all imaging parameters, including exposure time, must be kept the same.

There are two types of sequencing operations possible. The first type is called a "Bulk Sequence". A bulk sequence is defined as a sequence of images that are all placed into local memory before being downloaded in one block by the calling application. In this mode, the application is responsible for allocating memory for the entire sequence of images (as if concatenated together), and then must separate each image after downloading. The second type is called a "Streaming Sequence". In this type of sequence, the camera system starts the series of images with a single *Expose* command. However, the calling application checks the camera status regularly to determine when the next image will be ready for download.

Applications should take care to ensure that data is paced back to the user's computer regularly (for streaming sequences), otherwise the application should make sure that the series of images will not overflow the camera's local memory buffer. If the local memory buffer is filled, the camera will stop digitization of subsequent images until there is room in the buffer. Note, however, that an application should have little problem in streaming data back to the user's computer and staying ahead of the internal digitization process. This warning is mainly provided for specific types of use and applications where, for whatever reason, the calling software is not able to download image data for an extended period of time.

Ethernet camera systems may only use the bulk sequence operation.

12.2 Bulk Sequence Control and Usage

The following is meant to provide a simple example of bulk sequences using C++. As previously discussed, a bulk sequence is defined as a sequence of images that are all placed into local memory before being downloaded in one block by the calling application. In this mode, the application is responsible for allocating memory for the entire sequence of images (as if concatenated together), and then must separate each image after downloading. The example code demonstrates how to set up a bulk sequence, divides the sequence data into separate frames, and saves each frame as a separate file to disk.

```
// Query user for number of images in the sequence
printf( "Number of images in the sequence:  " );
scanf( "%d", &NumImages );
printf( "Preparing sequence of %d images.\n", NumImages );

// Set the image count
ApogeeCamera->ImageCount = NumImages;
```

```
// Query the camera for a full frame image
long ImgXSize = ApogeeCamera->ImagingColumns;
long ImgYSize = ApogeeCamera->ImagingRows;

// Set sequence download variable
ApogeeCamera->SequenceBulkDownload = true;

// Variables for the image buffer and another for iterating through the
buffer
unsigned short* pBuffer;
unsigned short* pBufferIterator;

// Allocate memory and calculate a byte count.  For bulk
// sequences, the buffer should be sized to include all images
pBuffer           = new unsigned short[ ImgXSize * ImgYSize * NumImages];
ImgSizeBytes      = ImgXSize * ImgYSize * 2;

// Do a 0.001s dark frame (bias)
printf( "Starting camera exposure...\n" );
ApogeeCamera->Expose( 0.001, false );

// Check camera status to make sure image data is ready
while ( ApogeeCamera->ImagingStatus != Apn_Status_ImageReady );

// Get the image data from the camera
printf( "Retrieving image data from camera...\n" );
ApogeeCamera->GetImage( (long)pBuffer );

pBufferIterator = pBuffer;

// Write the test images to different output file (overwrite if it already
exists)
// In this process, we are dividing the single image buffer into the separate
// images that comprise the bulk data set.
for ( i=1; i<=NumImages; i++ )
{
      sprintf( szFilename, "BulkImage%d.raw", i );

      filePtr = fopen( szFilename, "wb" );

      if ( filePtr == NULL )
      {
            printf( "ERROR:  Failed to open file for writing output data." );
      }
      else
      {
            printf( "Wrote image to output file \"%s...\"\n", szFilename );

            fwrite( pBufferIterator, sizeof(unsigned short),
(ImgSizeBytes/2), filePtr );
            fclose( filePtr );

            if ( i < NumImages )
            {
                  // Only change the pointer for the first n-1 images
                  pBufferIterator += (ImgSizeBytes/2);
```

```
            }
      }
}
```

12.3 Streaming Sequence Control and Usage

The following is meant to provide a simple example of streaming sequences using C++. As previously discussed, a streaming sequence is defined as a sequence of images that are continuously sent back to the application while other images in the sequence are still being exposed. Applications do not normally need to worry about keeping up with the sequence, as images can be buffered into the camera's local memory before being downloaded by the application. In this mode, the application is responsible for allocating memory for at least one image, and then downloading the series of images. The example code demonstrates how to set up a streaming sequence, and saves each downloaded frame as a separate file to disk.

```
// Query user for number of images in the sequence
printf( "Number of images in the sequence:  " );
scanf( "%d", &NumImages );
printf( "Preparing sequence of %d images.\n", NumImages );

// Set the image count
ApogeeCamera->ImageCount = NumImages;

// Query the camera for a full frame image
long ImgXSize = ApogeeCamera->ImagingColumns;
long ImgYSize = ApogeeCamera->ImagingRows;

// Toggle the sequence download variable
ApogeeCamera->SequenceBulkDownload = false;

// Variable for the image buffer
unsigned short* pBuffer;

// Create a buffer for one image, which will be reused for
// each image in the sequence
pBuffer           = new unsigned short[ ImgXSize * ImgYSize];
ImgSizeBytes      = ImgXSize * ImgYSize * 2;

// Do a sequence of 0.001s dark frames (bias frames)
printf( "Starting camera exposure...\n" );
ApogeeCamera->Expose( 0.001, false );

for ( i=1; i<=NumImages; i++ )
{
      while ( ApogeeCamera->SequenceCounter != i )
      {
            // printf( "Waiting for ApogeeCamera->SequenceCounter to
increment\n" );
      }

      // Get the image data from the camera
      printf( "Retrieving image data from camera (Image #%d)...\n",
ApogeeCamera->SequenceCounter );
      ApogeeCamera->GetImage( (long)pBuffer );
```

```
      sprintf( szFilename, "StreamedImage%d.raw", i );

      filePtr = fopen( szFilename, "wb" );

      if ( filePtr == NULL )
      {
            printf( "ERROR:  Failed to open file for writing output data." );
      }
      else
      {
            printf( "Wrote image data to output file \"%s...\"\n", szFilename
);

            fwrite( pBuffer, sizeof(unsigned short), (ImgSizeBytes/2),
filePtr );
            fclose( filePtr );
      }
}
```

13 Continuous Imaging

13.1 Overview

Continuous Imaging is a special form of the camera's internal sequencing engine. When using this type of internal sequencing, the camera may be set up so that it does not simply capture a specific number of frames with one *Expose* call. Rather, the camera will continuously take one image after another until a *StopExposure* command is issued. This type of operation may only be used as a form of streaming sequences.

Sequences of images are useful when an application constantly wants to cycle frames through the camera system, and the application also wants to strictly minimize the delay between successive images. For application and programming simplicity, Apogee Instruments only recommends using continuous imaging when both of these conditions must be met. When the camera's internal sequencing engine is used, all imaging parameters, including exposure time, must be kept the same. Continuous imaging is a specific mode, requiring the application to regularly request and track the data stream from the camera. Some applications may find less complexity in simply constructing a loop that takes single frames one at a time until told to exit the loop.

Ethernet camera systems cannot perform streaming sequences, and therefore cannot use the continuous imaging feature.

Continuous imaging is only available when the camera mode is *Apn_CameraMode_Normal*.

13.2 Control and Usage

Continuous imaging requires some extra overhead on the part of the calling application. Once an application begins the continuous imaging process with an *Expose* call, the application must continuously check for the next image frame to be downloaded. The application must provide itself some process to send the camera a *StopExposure* command when requested.

Frames will be buffered in the camera's internal memory if the application does not request them fast enough. There is no provision for skipping frames in the memory buffer. The internal camera firmware manages the local memory buffer as a circular buffer, so the application should generally make sure to keep up on the number of frames available for download. However, the local memory within the device should be enough storage to handle the frames. Applications will also need to manage the rolling value of the *SequenceCounter* property which rolls over after counting from 1 to 65535.

Because control of continuous imaging requires event based processing, or a multithreaded application, a concise sample is not presented here. But the usage is straightforward:

1. Enable the *ContinuousImaging* property
2. Call the *Expose* method
3. Loop as if doing an infinite streaming sequence, looking at the *SequenceCounter* property before requesting data
4. Issue *GetImage* calls when the data is ready
5. Return to the loop processing until a *StopExposure* command is requested by the user

14 Hardware Triggering

14.1 Overview

The Alta and Ascent camera systems allow for the use of an external, hardware trigger/signal to begin an exposure. The trigger signal arrives through the camera I/O port—the pins and use of which are defined in another section of this document. This section provides additional detail on the properties for enabling or disabling different types of exposure triggers.

Previous versions of the driver and firmware used the *CameraMode* property to control hardware triggering, by setting this property to either *Apn_CameraMode_ExternalShutter* or *Apn_CameraMode_ExternalTrigger*. Trigger operations are now controlled by properties that are set when using the camera in a specific mode. The following short table shows the trigger properties and the corresponding camera modes for which they are valid.

Property	**Normal**	**TDI**	**Kinetics**
ExternalShutter	Yes	No	No
ExternalIoReadout	Yes	No	No
TriggerNormalEach	Yes	No	No
TriggerNormalGroup	Yes	No	No
TriggerTdiKineticsEach	No	Yes	Yes
TriggerTdiKineticsGroup	No	Yes	Yes

The *ExternalShutter* property is straightforward. When used, this signal (which is assigned a different I/O pin than the usual trigger start signal) controls the length of the exposure. It may be used in conjunction with the *ExternalIoReadout* property, to control when digitization begins. These two properties are designed to be used with single exposures.

The *Each*/*Group* trigger properties are designed to give the greatest flexibility and number of options to users, for each corresponding camera mode.

14.2 Normal Mode Triggers

The following chart details how the *Each*/*Group* properties are interpreted in *Apn_CameraMode_Normal*, when *ImageCount* equals one (single exposure) and when *ImageCount* is greater than one (using the camera's internal sequence engine).

	ImageCount = 1	ImageCount > 1
TriggerNormalEach = FALSE TriggerNormalGroup = FALSE	Software initiated single exposure. No hardware trigger enabled.	Software initiated sequenced exposure. No hardware trigger enabled.
TriggerNormalEach = FALSE TriggerNormalGroup = TRUE	Hardware trigger is used to begin the single exposure.	Hardware trigger is used to begin the sequenced exposure. One trigger kicks off the entire series of images.

TriggerNormalEach = TRUE TriggerNormalGroup = FALSE	Not a valid/usable option, and will have no impact. Because *ImageCount* is one, the camera control firmware should ignore the *Each* setting.	The first image of the sequence is begun by software control. Each subsequent image in the sequence will be initiated when its corresponding hardware trigger arrives.
TriggerNormalEach = TRUE TriggerNormalGroup = TRUE	Hardware trigger is used to begin the single exposure. Because *ImageCount* is one, the camera control firmware should ignore the *Each* setting.	The first image, as well as all subsequent images, of the sequence will be initiated by a corresponding hardware trigger.

14.3 TDI Triggers

The following chart details how the *Each*/*Group* properties are interpreted in *Apn_CameraMode_TDI.* TDI operation presumes multiple rows, and, in effect, is very similar to a sequence of normal images.

TriggerTdiKineticsEach = FALSE TriggerTdiKineticsGroup = FALSE	Software initiated TDI image. No hardware trigger enabled.
TriggerTdiKineticsEach = FALSE TriggerTdiKineticsGroup = TRUE	A single hardware trigger is used to begin the entire TDI image.
TriggerTdiKineticsEach = TRUE TriggerTdiKineticsGroup = FALSE	The first row of the TDI image is begun by software control. Each subsequent row in the TDI image will be initiated when its corresponding hardware trigger arrives.
TriggerTdiKineticsEach = TRUE TriggerTdiKineticsGroup = TRUE	The first row, as well as all subsequent rows, of the TDI image will be initiated by a corresponding hardware trigger.

14.4 Kinetics Triggers

The following chart details how the *Each*/*Group* properties are interpreted in *Apn_CameraMode_Kinetics.*

TriggerTdiKineticsEach = FALSE TriggerTdiKineticsGroup = FALSE	Software initiated Kinetics image. No hardware trigger enabled.
TriggerTdiKineticsEach = FALSE TriggerTdiKineticsGroup = TRUE	A single hardware trigger is used to begin the entire Kinetics imaging process.
TriggerTdiKineticsEach = TRUE TriggerTdiKineticsGroup = FALSE	The first row of the TDI image is begun by software control. Each subsequent row in the TDI image will be initiated when its corresponding hardware trigger arrives.
TriggerTdiKineticsEach = TRUE TriggerTdiKineticsGroup = TRUE	The first row, as well as all subsequent rows, of the TDI image will be initiated by a corresponding hardware trigger.

14.5 Control and Usage

The following chart details how the *Each*/*Group* properties are interpreted in *Apn_CameraMode_Kinetics.*

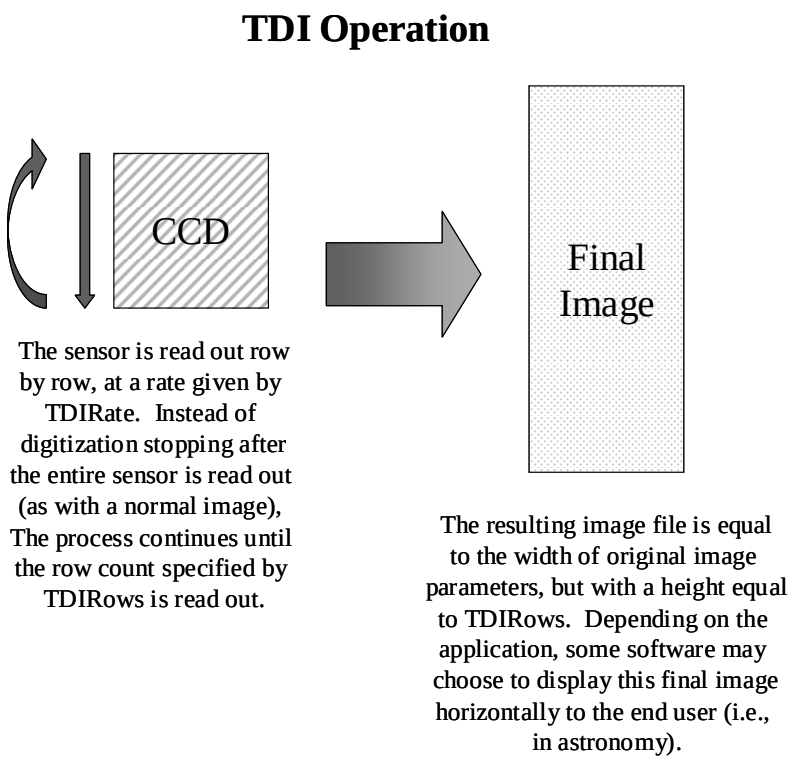
```
////////////////////////////////////////////////////////////////////
// Single Hardware Trigger Example
////////////////////////////////////////////////////////////////////

// First we'll do a single triggered exposure.  This requires the
// "TriggerNormalGroup" property to be enabled
ApogeeCamera->TriggerNormalGroup = true;

// We will make our exposure a 0.1s light frame.  Even though this
// is a triggered exposure, we still need the Expose() method to be
// called to set up our exposure
printf( "Starting a single triggered exposure of 0.1s...\n" );
ApogeeCamera->Expose( 0.1, true );

// Tell the user something informative...
printf( "Waiting on trigger...\n" );

// Check camera status to make sure image data is ready
while ( ApogeeCamera->ImagingStatus != Apn_Status_ImageReady );

// Get the image data from the camera
printf( "Retrieving image data from camera...\n" );
ApogeeCamera->GetImage( (long)pBuffer );

// We're going to set this to "true" again in the next example, but
// for good form, we'll return our state to non-triggered images
ApogeeCamera->TriggerNormalGroup = false;

////////////////////////////////////////////////////////////////////
// Sequenced (Internal) Hardware Trigger Example
////////////////////////////////////////////////////////////////////

// NOTE: The following example uses the camera engine's internal
// capability to do sequences of images.  An application could
// also easily do a loop of single triggered images in order to
// achieve the same result, but driven by software/the application.

// Do a sequence of triggered exposures.  This requires both the
// "TriggerNormalGroup" and "TriggerNormalEach" properties to be
// enabled.  TriggerNormalGroup will enable a trigger for the first
// image.  TriggerNormalEach will enable a trigger for each
// subsequent image in the sequence.
ApogeeCamera->TriggerNormalGroup    = true;
ApogeeCamera->TriggerNormalEach     = true;

// Toggle the sequence download variable
ApogeeCamera->SequenceBulkDownload = false;

// Set the image count
```

```
NumImages = 5;

// Set the image count
ApogeeCamera->ImageCount = NumImages;

// For visual clarification, enable an LED light to see when the camera is
// waiting on a trigger to arrive.  Enable the other LED to see when the
// camera is flushing.
ApogeeCamera->LedMode   = Apn_LedMode_EnableAll;
ApogeeCamera->LedA      = Apn_LedState_ExtTriggerWaiting;
ApogeeCamera->LedB      = Apn_LedState_Flushing;

// As with single exposures, we must start the trigger process with the
// Expose method.  We only need to call Expose once to kick off the entire
// sequence process.
ApogeeCamera->Expose( 0.1, true );

for ( i=1; i<=NumImages; i++ )
{
      printf( "Waiting on trigger for image #%d...\n", i );

      // For sequences of images, the correct usage is to use the
      // SequenceCounter property in order to correctly determine when
      // an image is ready for download.
      while ( ApogeeCamera->SequenceCounter != i );

      // Get the image data from the camera
      printf( "Retrieving image data from camera (Image #%d)...\n", i );
      ApogeeCamera->GetImage( (long)pBuffer );
}

// Return our state to non-triggered images
ApogeeCamera->TriggerNormalGroup = false;
ApogeeCamera->TriggerNormalEach  = false;

// Reset our image count to 1
ApogeeCamera->ImageCount = 1;
```

15 Time Delayed Integration (TDI) Mode

15.1 Overview

Time Delayed Integration (TDI) Mode is a special mode of camera operation used to create single images that are larger than the sensor area of the CCD itself. The camera is programmed to read out and digitize each row at a particular rate, up to some number of pre-programmed rows. The following diagram illustrates the process:



Because of their unique operation, interline sensors cannot use TDI mode.

15.2 Control and Usage

To control the camera in TDI mode, an application should set the *CameraMode* variable to *Apn_CameraMode_TDI*. The *TDIRate* property is used to control the rate at which rows are digitized in the camera. The *TDIRows* variable is the total number of rows in the final TDI image. The *Expose* method is used for beginning the TDI process, but note that the *Duration* parameter is not used. Applications should consider TDI to be a type of sequence, with the corresponding option to download the image data in bulk or streaming formats. For streaming downloads, software should use the *TDICounter* property to determine when a new row is ready to be retrieved by the calling application. When performing a bulk download of the image data, the application should treat the TDI image as a single image that will be ready for download based on the *Apn_Status_ImageReady* flag.

TDI may be used in conjunction with hardware triggering.

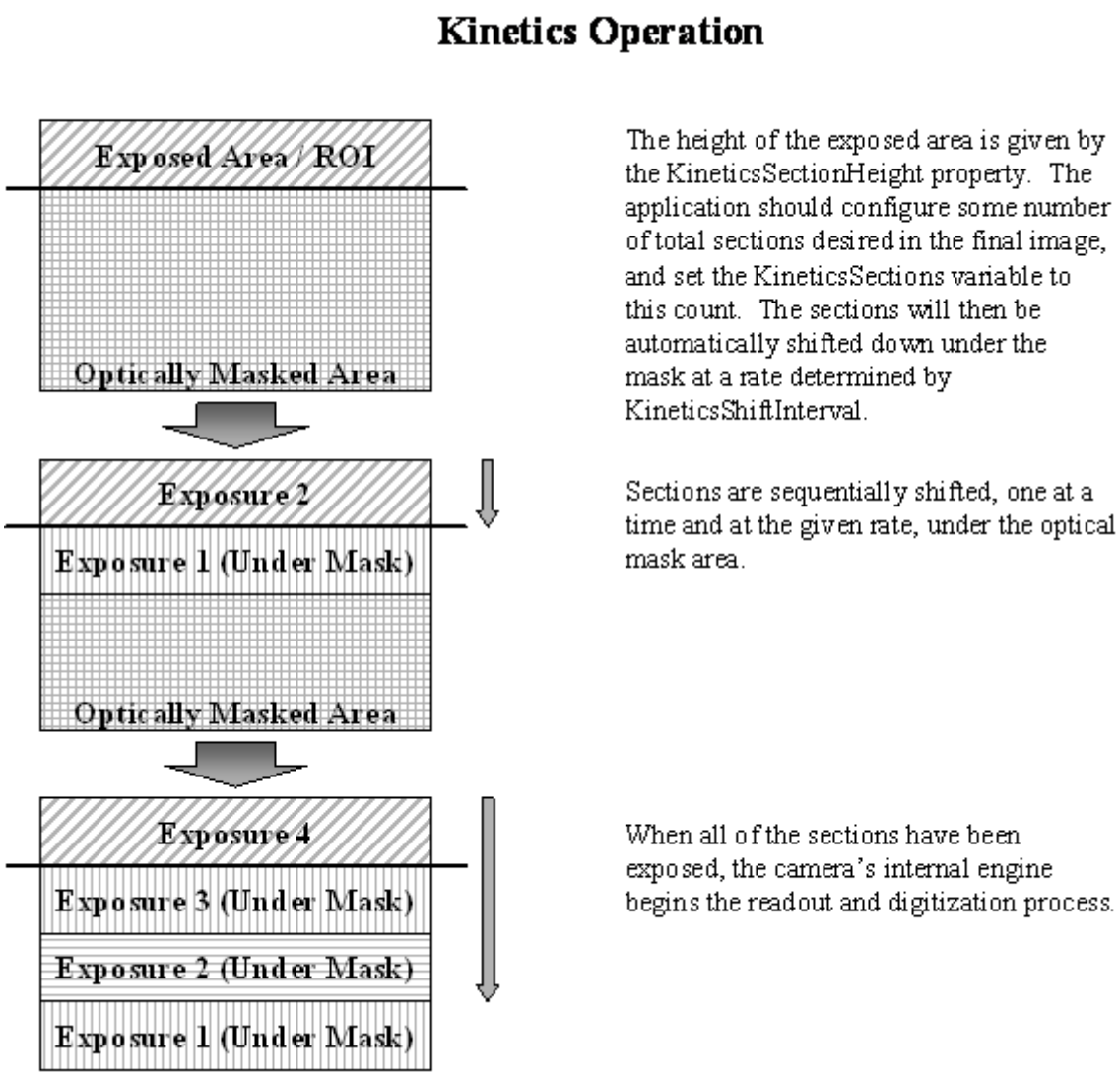
The following C++ code shows the basic operation of TDI as a streaming sequence:

```
// Set the TDI row count
NumTdiRows = 3000;
ApogeeCamera->TDIRows = NumTdiRows;

// Set the TDI rate
ApogeeCamera->TDIRate = 0.3;

// Toggle the camera mode for TDI
ApogeeCamera->CameraMode = Apn_CameraMode_TDI;

// Toggle the sequence download variable
ApogeeCamera->SequenceBulkDownload = false;

// Set the image size
// The height is 1 since SequenceBulkDownload is false
long ImgXSize = ApogeeCamera->ImagingColumns;
long ImgYSize = 1;

// Create a buffer for one line of data
pBuffer = new unsigned short[ ImgXSize * ImgYSize];

// Start the exposure
ApogeeCamera->Expose( 0.001, true );

// Get the row data from the camera
for ( i=1; i<=NumTdiRows; i++ )
{
      // wait for next TDI row
      while ( ApogeeCamera->TDICounter != i );

      // Get the line data from the camera
      ApogeeCamera->GetImage( (long)pBuffer );

      // Do something with the row of data just downloaded
}
```

16 Kinetics Mode

16.1 Overview

Kinetics Mode is a special mode of camera operation used to take extremely fast, successive images of small vertical region of the sensor. In this mode, the user will optically mask off most of the CCD sensor, allowing for some small portion to be exposed during the course of the experiment. The exposed section of the sensor is illuminated, and then shifted by some number of rows (a "section height"). The sensor is then exposed again to light, and then shifted repeatedly, until the user has exposed the entire surface of the CCD sensor. One the appropriate number of "sections" are exposed, the sensor is read out and the data is digitized. The following diagram illustrates this process:



Because of their unique operation, interline sensors cannot use Kinetics mode.

16.2 Control and Usage

To control the camera in Kinetics Mode, an application should set the *CameraMode* variable to *Apn_CameraMode_Kinetics*. Each section of a Kinetics mode image has a specified vertical height, defined by the value of *KineticsSectionHeight*. The *KineticsShiftInterval* property is used to control the rate at which each section is shifted in the camera. The *KineticsSections* variable is the total number of sections in the final image. The *Expose* method is used for beginning the Kinetics process, but note that the *Duration* parameter is not used. From the standpoint of the application, a Kinetics image is a single image, and applications should query the *Apn_Status_ImageReady* flag.

Kinetics Mode may be used in conjunction with hardware triggering. However, when using hardware triggering with Kinetics Mode, the software should take care to disable other options on the I/O port.

Kinetics Mode may be used with the camera's internal sequencing capabilities.

The following C++ code shows the basic operation of Kinetics Mode:

```
// Set the camera mode to Kinetics
ApogeeCamera->CameraMode = Apn_CameraMode_Kinetics;

// 4 sections
NumSections = 4;

// Set our section variables
ApogeeCamera->KineticsSections            = NumSections;
ApogeeCamera->KineticsSectionHeight = ImgYSize / NumSections;

// Set the section rate...using 0.2s arbitrarily
ApogeeCamera->KineticsShiftInterval = 0.2;

// Begin the exposure process
ApogeeCamera->Expose( 0.001, true );

// Check camera status to make sure image data is ready
while ( ApogeeCamera->ImagingStatus != Apn_Status_ImageReady );

// Get the image data from the camera
ApogeeCamera->GetImage( (long)pBuffer );

// Do something with the image data just downloaded
```

17 Examples

17.1 ICamera2 Using C++

The following is meant to provide a simple example of using the ICamera2 and ICamDiscover objects within the Microsoft Visual C++ environment. The example code creates the ICamera2 and ICamDiscover objects, attempts to locate a usable camera, and then takes a dark frame image with a 0.001s duration. The example test image is then retrieved. After this is done, the camera I/O port is configured so that I/O pins 4 and 5 are set up to be user defined I/O and pin 2 is set up as a shutter output signal. Finally, all of the created objects are released.

```
#include <stdio.h>

// Import the type library to create an easy to use wrapper class
#import "Apogee.DLL" no_namespace


void main()
{
      ICamera2Ptr       ApogeeCamera;           // Camera interface
      ICamDiscoverPtr   Discover;         // Discovery interface
      HRESULT           hr;               // Return code
      FILE*             filePtr;          // File pointer


      CoInitialize( NULL );               // Initialize COM library

      // Create the ICamera2 object
      hr = ApogeeCamera.CreateInstance( __uuidof( Camera2 ) );
      if ( SUCCEEDED(hr) )
      {
            printf( "Successfully created the ICamera2 object\n" );
      }
      else
      {
            printf( "Failed to create the ICamera2 object\n" );
            CoUninitialize();       // Close the COM library
            return;
      }

      // Create the ICamDiscover object
      hr = Discover.CreateInstance( __uuidof( CamDiscover ) );
      if ( SUCCEEDED(hr) )
      {
            printf( "Successfully created the ICamDiscover object\n" );
      }
      else
      {
            printf( "Failed to create the ICamDiscover object\n" );
            ApogeeCamera = NULL;    // Release ICamera2 COM object
            CoUninitialize();       // Close the COM library
            return;
```

```
}

// Set the checkboxes to default to searching both USB and
// ethernet interfaces for all types of Alta/Ascent cameras
Discover->DlgCheckEthernet    = true;
Discover->DlgCheckUsb         = true;

// Display the dialog box for finding a camera
Discover->ShowDialog( true );

// If a camera was not selected, then release objects and exit
if ( !Discover->ValidSelection )
{
      printf( "No valid camera selection made\n" );
      Discover          = NULL;     // Release ICamDiscover COM object
      ApogeeCamera      = NULL;     // Release ICamera2 COM object
      CoUninitialize();             // Close the COM library
      return;
}

// Initialize camera using the ICamDiscover properties
hr = ApogeeCamera->Init( Discover->SelectedInterface,
                         Discover->SelectedCamIdOne,
                         Discover->SelectedCamIdTwo,
                         0x0 );

if ( SUCCEEDED(hr) )
{
      printf( "Connection to camera succeeded.\n" );
}
else
{
      printf( "Failed to connect to camera" );
      Discover          = NULL;     // Release Discover COM object
      ApogeeCamera      = NULL;     // Release ICamera2 COM object
      CoUninitialize();             // Close the COM library
      return;
}

// Query the camera for a full frame image
long ImgXSize = ApogeeCamera->ImagingColumns;
long ImgYSize = ApogeeCamera->ImagingRows;

// Allocate memory
unsigned short *pBuffer = new unsigned short[ ImgXSize * ImgYSize ];

// Calculate counts
unsigned long ImgSizeBytes = ImgXSize * ImgYSize * 2;
unsigned long PixelCount   = ImgXSize * ImgYSize;

// Display the camera model
_bstr_t szCamModel( ApogeeCamera->CameraModel );
printf( "Camera Model:  %s\n", (char*)szCamModel );

// Display the driver version
_bstr_t szDriverVer( ApogeeCamera->DriverVersion );
printf( "Driver Version:  %s\n", (char*)szDriverVer );
```

```
	// Do a 0.001s dark frame (bias)
	printf( "Starting camera exposure...\n" );
	ApogeeCamera->Expose( 0.001, false );

	// Check camera status to make sure image data is ready
	while ( ApogeeCamera->ImagingStatus != Apn_Status_ImageReady );

	// Get the image data from the camera
	printf( "Retrieving image data from camera...\n" );
	ApogeeCamera->GetImage( (long)pBuffer );

	// Write test image to an output file (overwrite if it already exists)
	filePtr = fopen( "ImageData.bin", "wb" );

	if ( filePtr == NULL )
	{
		printf( "ERROR:  Failed to open file for writing output data." );
	}
	else
	{
		printf( "Wrote image data to file \"ImageData.bin...\"\n" );
		fwrite( pBuffer, sizeof(unsigned short), PixelCount, filePtr );
		fclose( filePtr );
	}

	// Delete the memory buffer for storing the image
	delete [] pBuffer;

	// Show how to configure the I/O Port registers

	// Default setting is for the I/O Port to be completely user defined.
	// Setting the IoPortAssignment to 0x2 will then select only Pin 2
	// (Bit 1) to be configured for the pre-defined Shutter Output state
	// (note that Bit 0 corresponds to Pin 1)
	ApogeeCamera->IoPortAssignment = 0x2;

	// We want Pins 4 and 5 to be configured as outputs, so this requires
	// us to set Bits 3 and 4 of the IoPortDirection variable (note that
	// Bit 0 corresponds to Pin 1)
	ApogeeCamera->IoPortDirection = 0x18;

	// The I/O Port is now configured for the application to use.

	// Release our allocated objects.  Alternatively, we could call the
	// ApogeeCamera->Close() method, but that isn't necessary
	// in C++, as setting the object to NULL will close down the object.
	Discover          = NULL;           // Release ICamDiscover COM object
	ApogeeCamera      = NULL;           // Release ICamera2 COM object

	CoUninitialize();       // Close the COM library
}
```

17.2 ICamera2 Using VB.NET

The following is meant to provide a simple example of using the ICamera2 and ICamDiscover objects within the Microsoft Visual Basic .NET environment. The example code creates the ICamera2 and ICamDiscover objects, attempts to locate a usable camera, and then takes a dark frame image with a 0.001s duration. The example test image is then retrieved, and written to a file.

```
Module Module1

    Sub Main()

        Dim FindDlg As APOGEELib.CamDiscover
        Dim ApogeeCamera As APOGEELib.Camera2
        Dim ImageData As Array
        Dim FileNum As Integer

        FindDlg = New APOGEELib.CamDiscover()
        ApogeeCamera = New APOGEELib.Camera2()
        FileNum = FreeFile()

        FindDlg.DlgCheckEthernet = True
        FindDlg.DlgCheckUsb = True

        FindDlg.ShowDialog(True)

        If FindDlg.ValidSelection Then

            ApogeeCamera.Init(FindDlg.SelectedInterface,
FindDlg.SelectedCamIdOne, FindDlg.SelectedCamIdTwo, 0)

            ApogeeCamera.Expose(0.001, False)

            Do
            Loop Until ApogeeCamera.ImagingStatus =
APOGEELib.Apn_Status.Apn_Status_ImageReady

            ImageData = ApogeeCamera.Image

            FileOpen(FileNum, "Image.raw", OpenMode.Binary, OpenAccess.Write)
            FilePut(FileNum, ImageData)
            FileClose(FileNum)

        End If

    End Sub

End Module
```

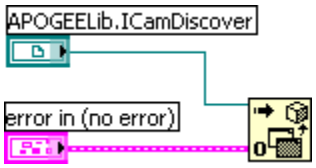
17.3 ICamera2 Using LabVIEW

The Apogee ActiveX/COM DLL can be used within LabVIEW, a graphical programming environment from National Instruments. LabVIEW allows the user to control the camera system through the DLL. At this time, Apogee does not provide an instrument driver for LabVIEW beyond the Apogee ActiveX/COM DLL.

The easiest way to invoke the ActiveX/COM capabilities within LabVIEW is to use LabVIEW as an Automation Client. In this mode, LabVIEW acts as a client, and requests information from the Apogee DLL, which is the automation server.

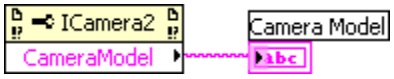
In order to use the Apogee DLL from within LabVIEW, refer to your LabVIEW documentation to create an Automation Open Reference. This will allow the ActiveX/COM DLL to be opened. The Automation Reference requires the user to select an ActiveX class in order to operate properly. Choose the option to "Select ActiveX Class" and look at the list of available ActiveX components on the computer. Note that it is not unusual for many components to be registered. Select the component labeled "Apogee Camera Control Library." If the "Apogee Camera Control Library" is not present or shown as an ActiveX Class, then the Apogee.DLL has not been installed properly. Please see your installation instructions for proper installation before continuing. Once the reference has been opened, LabVIEW will refer to it in a shortened form, i.e. APOGEELib.ICamera2. For camera discovery, the name will appear as APOGEELib.ICamDiscover.

The partial diagram below shows the Automation Open Reference for an ActiveX control, using the APOGEELib.ICamDiscover object.



Once the Automation Reference has been opened with the Apogee ActiveX camera control, the various Properties and Methods of the object will be available from the Automation Property Nodes and Automation Invoke Nodes. These nodes also require an associated ActiveX Class, which should be set to the ICamera2 or ICamDiscover object. Once this is done, select the appropriate Method or Property to use, and connect the node to other LabVIEW components as appropriate.

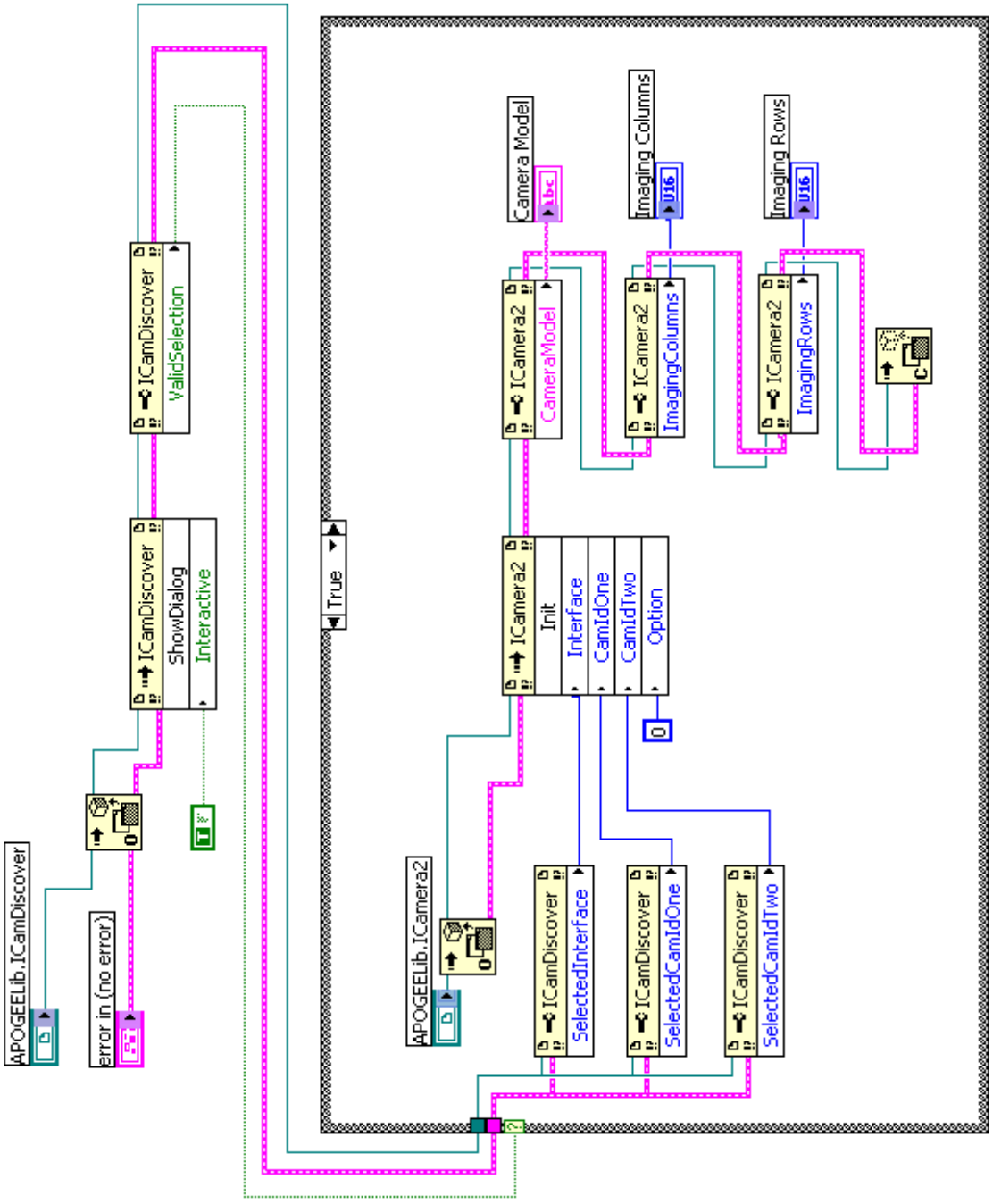
The partial diagram below shows a Property Node (CameraModel).



When finished with the Apogee ActiveX Control, make sure to complete operation with an Automation Close Reference.

The diagram on the next page is a very simple LabVIEW virtual instrument, which opens an Automation Reference to control the ICamDiscover interface, and queries the user for a camera selection. The sample then opens another Automation Reference to the ICamera2 interface, initializes the camera with the Init method, and then uses the ICamera2 interface to display the camera model, as well as the number of rows and columns available for imaging.

For more information regarding LabVIEW usage, as well as specifics of how to use LabVIEW as an Automation Client, please reference the documentation provided by National Instruments.


APOGEELib.ICamDiscover
error in (no error)
ICamDiscover
ShowDialog
Interactive
ICamDiscover
ValidSelection
True
APOGEELib.ICamera2
ICamera2
Init
Interface
CamIdOne
CamIdTwo
Option
0
ICamDiscover
SelectedInterface
ICamDiscover
SelectedCamIdOne
ICamDiscover
SelectedCamIdTwo
Camera Model
ICamera2
CameraModel
Imaging Columns
U16
ICamera2
ImagingColumns
Imaging Rows
U16
ICamera2
ImagingRows

17.4 ISerialPort Using C++

The following is meant to provide a simple example of using the ISerialPort object within the Microsoft Visual C++ environment. The example code creates the object, attempts to locate a usable Alta camera, and then performs a simple write/read sequence from the port, as well as changing the baud rate. Note that any delay between a serial port write and a subsequent read is the responsibility of the calling application to implement, depend on the serial port hardware being used. A Sleep() statement is used in the example below.

```
#include <stdio.h>

// Import the type library to create an easy to use wrapper class
#import "apogee.dll" no_namespace


void main()
{
      ISerialPortPtr    AltaSerialPort;   // Camera interface
      ICamDiscoverPtr   Discover;         // Discovery interface
      HRESULT           hr;               // Return code

      CoInitialize( NULL );               // Initialize COM library

      // Create the ActiveX objects from the universally unique identifier
      hr = AltaSerialPort.CreateInstance( __uuidof( SerialPort ) );
      if ( FAILED( hr ) )
      {
            printf("Failed to create the ISerialPort object\n");
            return;
      }
      else
      {
            printf("Successfully created the ISerialPort object\n");
      }

      hr = Discover.CreateInstance( __uuidof( CamDiscover ) );
      if ( FAILED( hr ) )
      {
            printf("Failed to create the ICamDiscover object\n");
            return;
      }
      else
      {
            printf("Successfully created the ICamDiscover object\n");
      }

      Discover->ShowDialog( true );

      if ( Discover->ValidSelection )
      {
            hr = AltaSerialPort->OpenPort( Discover->SelectedInterface,
                                           Discover->SelectedCamIdOne,
                                           0x0, 0x0 );
      }
```

```
      else
      {
            printf( "No Valid Selection made\n" );
            return;
      }

      if ( FAILED(hr) )
      {
            printf( "Failed to connect to camera serial port\n" );
      }
      else
      {
            printf( "Connection Successful.\n" );

            _bstr_t WriteBuffer;
            _bstr_t ReadBuffer;
            BSTR    TempReadBuffer;
            short   ReadCount;
            ULONG   BaudRate;

            WriteBuffer = ":GB#";

            AltaSerialPort->SerialData = WriteBuffer.copy();

            printf( "Buffer written to port:  %s\n", (char*)WriteBuffer );

            Sleep( 200 );

            ReadBuffer = _bstr_t( AltaSerialPort->SerialData );

            printf( "Buffer read from port:  %s\n", (char*)ReadBuffer );
            printf( "Buffer length:  %u\n", AltaSerialPort->BytesRead );

            // Baud rate test
            BaudRate = AltaSerialPort->BaudRate;
            printf( "Baud Rate:  %u\n", BaudRate );
            AltaSerialPort->BaudRate = 19200;
            BaudRate = AltaSerialPort->BaudRate;
            printf( "Baud Rate:  %u\n", BaudRate );

            hr = AltaSerialPort->ClosePort( );

            if ( FAILED(hr) )
            {
                  printf( "Failed to close the serial port\n" );
            }
            else
            {
                  printf( "Serial port connection closed\n" );
            }
      }

      Discover          = NULL;     // Release the objects
      AltaSerialPort    = NULL;

      CoUninitialize();             // Close the COM library
}
```

18 Application Notes

The following notes comprise a short Frequently Asked Questions (FAQ) that Apogee Instruments has received by various developers over time. These notes should help explain some of the more common problems and questions encountered during application development.

18.1 Detecting USB Device Removal

The Apogee Alta and Ascent USB cameras are, like all USB devices, Plug and Play compatible. This means they may be removed or added from the system by simply connecting or disconnecting the USB device cable and/or the power cable to the camera. If an application is open while the camera is disconnected, further commands will obviously fail. Therefore, a provision is made for applications which poll the camera for status information, to learn that an error occurred while the application was attempting to communicate with the camera system.

In the case where a camera control application is running, and power is removed from the Alta or Ascent camera…on the next operation requested of the camera, the Apogee.DLL will internally monitor for a failed operation (if power is removed, the operation will certainly fail). If the Apogee.DLL detects a failure, it will immediately attempt to reconnect to the camera and retry the operation. If that reconnect/retry sequence fails, the DLL will not allow further access to the hardware until another successful *Init* operation has been performed. Any query to the *ImagingStatus* property will return *Apn_Status_ConnectionError*.

Camera control applications should periodically poll the camera to monitor status (see section on "Application Polling and Camera Status") and process the *Apn_Status_ConnectionError*. On detection of this error, the camera object should be closed using either the *Close* method or delete the object entirely. The application should inform the user of the connection error, and offer the user some means to reconnect to the camera when the Alta or Ascent camera is reconnected to the computer.

Applications which need additional detail about device removal should process the WM_DEVICECHANGE message, issued by Microsoft Windows Operating Systems to Plug and Play events such as USB device removal and addition. Application writers should consult the Microsoft documentation on processing this message, according to the particular programming language they are using.

18.2 Application Polling and Camera Status

Applications will want to periodically query the camera system to retrieve current status and temperature information. This allows an application to detect internal camera events such as when image data is ready to be downloaded, updates on temperature and cooler drive level, and also any particular error messages such as an *Apn_Status_ConnectionError* when a USB camera is disconnected while an application is still running.

The interval at which applications poll the camera for this type of status information is dependent on the frequency of desired updates. However, some care should be taken by applications which communicate with Ethernet camera systems, so that they do not saturate the camera with status requests, as network latency is far, far greater than the latency associated with a USB 2.0 connection.

For normal camera operation regarding status queries, Apogee Instruments recommends a polling frequency of not more than once per second for Ethernet systems, and not more than once per quarter second for USB systems. It is possible to poll more frequently, but, in the case of Ethernet systems, the

added network traffic may affect other camera operations such as displaying the Alta Ethernet camera web page. In the case of USB systems, higher polling rates are possible, but of limited value for nominal status queries of the system.

Some applications may wish to increase their polling frequency when checking the *ImagingStatus* property before issuing a call to the *GetImage* method. Applications should use whatever polling rate they feel is appropriate for their particular software architecture. If the application will use a frequent polling rate prior to issuing the *GetImage* call, Apogee Instruments recommends that, when taking long (multiple second) exposures, the application polls at a slower rate (i.e., perhaps once per second) until the exposure time remaining is less than roughly two seconds.

18.3 Retrieving Image Data

Prior to retrieving image data from the camera system, applications should query the *ImagingStatus* property to determine if the image data is ready to be downloaded. This is done by monitoring for a status of *Apn_Status_ImageReady*. Depending on the length of the exposure, the image data may not be available for download immediately. In this case, applications should not block while waiting for the status to change. Rather, applications should poll at some occasional frequency, so that the user's computer is still responsive and usable.

It is important to note that an application should only checking whether the conditional statement, is *ImagingStatus* equal to *Apn_Status_ImageReady*, is true. If an application attempts to check a broader range, such as greater than (less than) or equal to *Apn_Status_ImageReady*, it is possible that the application will request image data at the wrong time.

18.4 Camera State Persistence

Application writers should note that the Apogee.DLL driver does not maintain any permanent state information once an object has been closed or destroyed. Persistence of camera properties is the responsibility of the application.

18.5 Image Data and Pixel Format

The image data returned by the Apogee ActiveX/COM DLL is in a specific, consistent format. Image data is returned as unsigned, 16bit data. Saturation of a pixel is defined (in 16bit digitization mode) as a pixel value at or near (depending on the gain of the camera) 65535 counts. Cameras which support 12bit digitization of the sensor also return data in 16bit format; however, these cameras always have the upper four bits of each pixel as zero.

The first element of the image data returned to the application is the top left corner of the Region of Interest (ROI). The second element is the next pixel in the same row as the first element (i.e., immediately to the right of the first element). The rest of the first row follows, followed by the second row, and successively proceeding through the entire ROI.

18.6 Life Cycle of an Image

Every image acquired by the camera has a sort of life cycle associated with it. An image begins with a call to the *Expose* method. It ends with a call to either the *StopExposure* to *GetImage* method. An application must follow this rule for proper camera operation. In addition, an application should not call

Expose multiple times and then follow it with multiple *GetImage* calls. For example, the following is incorrect usage of the camera (showing only *Expose* and *GetImage* calls), and will not work:

Expose (Image 1)
Expose (Image 2)
GetImage (Image 1)
GetImage (Image 2)

Rather, the correct series of commands should be:

Expose (Image 1)
GetImage (Image 1)
Expose (Image 2)
GetImage (Image 2)

If an application wishes to do fast back to back exposures, then the software should use the camera's internal sequencing operations. Note that internal sequencing still begins the image acquisition process with a single call to *Expose*, but in that case, multiple *GetImage* calls are allowed (see that section of this document for additional information on correct usage of image sequences).

18.7 Correct Geometry when Binning

The pixel count values in the region of interest, *RoiPixelsH* and *RoiPixelsV*, are in terms of binned pixels. Therefore, when an application changes the horizontal or vertical binning (*RoiBinningH* and *RoiBinningV*), it must also change the pixel count (*RoiPixelsH* and *RoiPixelsV*) in the region of interest. Failing to set these parameters correctly may result in a condition where the camera's internal engine is set up to wait for a greater number of pixels than specified (for example, if the binning is set for 2x2, but the ROI parameters for width and height are not adjusted, the camera engine will be expecting approximately 4x the number of pixels it will receive during readout).

This behavior is by design. Because the camera allows subframing, the driver cannot correctly and automatically determine the pixel count based on only the binning parameters.

18.8 CameraMode and the Expose() Method

Application developers should keep in mind that regardless of the *CameraMode* setting, or other options enabled or disabled by various properties, the *Expose* method always acts as the official software starting point of taking an image. The *Expose* method is responsible for all final setup of the camera's internal state machine, and so must be called prior to any sort of image acquisition (ie, single exposures, triggered operations, sequences of images, etc.).

18.9 Avoiding the Use of ICamDiscover

Some application developers wish to avoid the use of discovering which cameras are connected to the system. For example, the developer may know in advance that their particular system or application will only ever have a single camera attached. In this case, the application may find it useful to simply call the Init method with the correct parameters, such as (in C++ code):

```
Init( Apn_Interface_USB, 0, 0x0, 0x0 );
```

This code will attempt to issue an Init call to the first USB camera enumerated on the user's system. This could also be used by ethernet camera systems (by modifying the appropriate fields), but such a scheme should only be used if the camera has been assigned a static IP address on the network, such that the assigned address remains the same over time.

18.10 TDI and Kinetics Mode and Interline Sensors

Because of their unique type of operation, interline sensors cannot be used with either TDI or Kinetics Mode.

18.11 Multithreaded applications

Some developers, for various reasons, write their applications using multiple threads. Generally speaking, we do not recommend this approach for most application developers, as the perceived performance gain is rarely worth the added development complexity and maintenance required by multithreaded software. Instead, we recommend developers using an event-type model, where requests to the camera driver are queued and passed along sequentially.

For developers that determine a multithreaded application is the right approach for the software they need to develop, the application should insure that commands sent to the camera do not interfere with one another. For example, if an application issues a GetImage call, the software should make sure that call returns before issues additional property or method requests to the camera. For example, suppose an application puts status and temperature reads in one thread, and other camera control operations in another thread. In that case, a GetImage call could be followed immediately by an interleaved temperature read request that is issued by the other thread, but before GetImage completes. Applications should take care to insure this does not happen. In general, the application should make sure that each camera request is sent serially, and completes/returns, before issuing another request to the camera.