Apogee Alta®
ActiveX / COM
API Specification

Supporting Ethernet and USB Interfaces

Specification Version 1.1

Revision Date: September, 2004
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® Camera Control development specification is provided as a courtesy to our customers, and comes without warranty of fitness for any purpose or application. The user assumes all risk for the use of the information contained in this document and the software it describes.

Copyright © 2003-2004 Apogee Instruments, Inc.
All rights reserved.

Alta is a registered trademark of Apogee Instruments, Inc.

All other trademarks mentioned in this document are the property of their respective owners, and are used herein solely for informational purposes only.

Apogee Instruments, Inc.
11760 Atwood Road, Suite #4
Auburn, CA  95603

(530) 888-0500
(530) 888-0540 FAX
## Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>First Applicable Driver Version</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>2.0.0</td>
<td>Initial Version</td>
</tr>
</tbody>
</table>
| 1.10             | 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 Introduction

Thank you for your interest in developing applications for the Apogee Alta line 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 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 line, 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 Alta 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:

**Apogee Alta ICamera2 Software Stack**

The remainder of this document will detail the various interfaces supported by the Alta systems.
2 Image Geometry

Alta 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 Methods

3.1 Init

3.1.1 Format:

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

3.1.2 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. Should be set to zero (0x0).

3.1.3 Description:

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

3.2 Close

3.2.1 Format:

    Close()
3.2.3 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.

3.3 ResetSystem

3.3.1 Format:

ResetSystem()

3.3.2 Parameters:

None.

3.3.3 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.

3.4 Expose

3.4.1 Format:

Expose( [in] double Duration,
        [in] Boolean Light )
3.4.2 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”.

3.4.3 Description:

The `Exposure()` 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.

3.5 PauseTimer

3.5.1 Format:

`PauseTimer( Boolean PauseState )`

3.5.2 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.

3.5.3 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.

3.6 StopExposure
3.6.1 Format:

StopExposure( Boolean Digitize )

3.6.2 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.

3.6.3 Description:

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

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

3.7 GetImage

3.7.1 Format:

GetImage(long pImageBuffer)

3.7.2 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.

3.7.3 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.
3.8 GetLine

3.8.1 Format:

GetLine(long pLineBuffer)

3.8.2 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.

3.8.3 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 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 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.

4.1 ShowIoDialog

4.1.1 Format:

ShowIoDialog()

4.1.2 Parameters:

None.

4.1.3 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.

The following graphic shows the I/O selection dialog:
4.2 ShowLedDialog

4.2.1 Format:

   ShowLedDialog()

4.2.2 Parameters:

   None.

4.2.3 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:

![LED Selection Dialog]

4.3 ShowTempDialog

4.3.1 Format:

   ShowTempDialog()
4.3.2 Parameters:

None.

4.3.3 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.

The following graphic shows the temperature control dialog:
## 5 ICamera2 Properties

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailableMemory</td>
<td>RO</td>
<td>Long</td>
<td>Returns the amount of available memory for storing images in terms of kilobytes (KB).</td>
</tr>
<tr>
<td>CameraInterface</td>
<td>RO</td>
<td>Apn_Interface</td>
<td>Returns the interface type supported by the camera. Valid values are listed below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0 (Apn_Interface_NET): Ethernet interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1 (Apn_Interface_USB): USB 2.0 interface.</td>
</tr>
<tr>
<td>CameraMode</td>
<td>R/W</td>
<td>Apn_CameraMode</td>
<td>Returns/Sets the operational mode of the camera. The default value for this variable after initialization is Apn_CameraMode_Normal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0 (Apn_CameraMode_Normal): Specifies nominal camera operation for exposure control. Single exposures, or sequences of exposures, are completely initiated by software control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1 (Apn_CameraMode_TDI): Specifies camera operation using time delayed integration (drift scan) mode. Used in conjunction with TDIRows and TDIRate. The actual TDI exposure is started with the Expose method, but the “Duration” parameter of Expose is not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x2 (Apn_CameraMode_Test): Specifies that the camera operation should be defined using simulated data for image parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3 (Apn_CameraMode_ExternalTrigger): Specifies camera operation using an external trigger to control the exposure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x4 (Apn_CameraMode_ExternalShutter): Deprecated driver feature. 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.</td>
</tr>
<tr>
<td>CameraModel</td>
<td>RO</td>
<td>String</td>
<td>Returns a camera model identifier for the device.</td>
</tr>
<tr>
<td>DataBits</td>
<td>R/W</td>
<td>Apn_Resolution</td>
<td>Digitization Resolution. Valid values are listed below. The default value for this variable after initialization is Apn_Resolution_SixteenBit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0 (Apn_Resolution_SixteenBit): Selects resolution of 16 bits per pixel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1 (Apn_Resolution_TwelveBit): Selects resolution of 12 bits per pixel.</td>
</tr>
<tr>
<td>DriverVersion</td>
<td>RO</td>
<td>String</td>
<td>Version number of the camera driver. This is the version of the driver stack as released by Apogee Instruments, and not specifically the file version of Apogee.DLL. A return value of &lt;=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.</td>
</tr>
<tr>
<td>FastSequence</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>FirmwareVersion</td>
<td>RO</td>
<td>Long</td>
<td>Version number of the camera control firmware.</td>
</tr>
</tbody>
</table>
| 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 temporarily disabled. 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 |
Apogee Alta Software Reference

| SerialASupport | RO | Boolean | Returns whether the camera supports Serial Port A. |
| SerialBSupport | RO | Boolean | Returns whether the camera supports Serial Port B. |

## Shutter Settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableShutter</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>ExternalIOReadout</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>ExternalShutter</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>ForceShutterOpen</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>ShutterAmpControl</td>
<td>R/W</td>
<td>Boolean</td>
<td>TRUE disables the CCD voltage while the shutter strobe is high. The default value of this variable after initialization is FALSE.</td>
</tr>
<tr>
<td>ShutterState</td>
<td>RO</td>
<td>Boolean</td>
<td>Returns TRUE if shutter is open; FALSE if closed.</td>
</tr>
<tr>
<td>ShutterStrobePeriod</td>
<td>R/W</td>
<td>Double</td>
<td>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)</td>
</tr>
<tr>
<td>ShutterStrobePosition</td>
<td>R/W</td>
<td>Double</td>
<td>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)</td>
</tr>
</tbody>
</table>

## LED Settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LedMode</td>
<td>R/W</td>
<td>Apn_LedMode</td>
<td>Format of the status LED lights. Must be one of following (Default is Apn_LedMode_EnableAll): 0x0 (Apn_LedMode_EnableAll): Enable all LEDs 0x1 (Apn_LedMode_EnableWhileExpose): Enable LEDs during exposure only 0x2 (Apn_LedMode_EnableAll): Enable LEDs at all times, overriding LED Mode DisableAll.</td>
</tr>
<tr>
<td>LedA</td>
<td>R/W</td>
<td>Apn_LedState</td>
<td>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</td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoPortAssignment</td>
<td>R/W</td>
<td>Long</td>
<td>Defines the signal usage for the I/O port. Valid range is for the 6 LSBS, 0x0 to 0x3F. The default value for this variable after initialization is 0x0.</td>
</tr>
<tr>
<td>Bit 0 (I/O Signal 1): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 1 (I/O Signal 2): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 2 (I/O Signal 3): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 3 (I/O Signal 4): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 4 (I/O Signal 5): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 5 (I/O Signal 6): 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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. |
| 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)

---

<table>
<thead>
<tr>
<th>IoPortData</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

### Cooler/Fan Settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoolerControl</td>
<td>RO</td>
<td>Boolean</td>
<td>Returns TRUE if the camera supports cooling, FALSE if no cooling is available.</td>
</tr>
<tr>
<td>CoolerRegulated</td>
<td>RO</td>
<td>Boolean</td>
<td>Returns TRUE if the camera supports regulated cooling, FALSE if regulated cooling is not available.</td>
</tr>
<tr>
<td>CoolerEnable</td>
<td>R/W</td>
<td>Boolean</td>
<td>Returns/Sets the Cooler operation. A value of TRUE will enable Cooler operation, and FALSE will turn the cooler off.</td>
</tr>
<tr>
<td>CoolerStatus</td>
<td>RO</td>
<td>Apn_CoolerStatus</td>
<td>Returns the current cooler status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0 (Apn_CoolerStatus_Off): Off (At or near Ambient). No drive applied to the Cooler.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1 (Apn_CoolerStatus_RampingToSetPoint): Ramp to the Set Point specified by the CoolerSetPoint property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x2 (Apn_CoolerStatus_AtSetPoint): At Set Point specified by the CoolerSetPoint property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>CoolerSetPoint</td>
<td>R/W</td>
<td>Double</td>
<td>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.</td>
</tr>
<tr>
<td>CoolerBackoffPoint</td>
<td>R/W</td>
<td>Double</td>
<td>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.</td>
</tr>
<tr>
<td>CoolerDrive</td>
<td>RO</td>
<td>Double</td>
<td>Drive level applied to the temp controller. Expressed as a percentage from 0% to 100%.</td>
</tr>
<tr>
<td>FanMode</td>
<td>R/W</td>
<td>Apn_FanMode</td>
<td>Returns/Sets the current fan speed. The default value of this variable after initialization is Apn_FanMode_Low.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x0 (Apn_FanMode_Off): Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1 (Apn_FanMode_Low): Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x2 (Apn_FanMode_Medium): Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3 (Apn_FanMode_High): High</td>
</tr>
<tr>
<td>TempCCD</td>
<td>RO</td>
<td>Double</td>
<td>Returns the current CCD temperature in degrees Celsius.</td>
</tr>
</tbody>
</table>
## TempHeatsink

**RO** Double

Returns the current Heatsink temperature in degrees Celsius.

### Geometry Settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysicalColumns, PhysicalRows</td>
<td>RO</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>ImagingColumns, ImagingRows</td>
<td>RO</td>
<td>Long</td>
<td>Returns the imaging area size in terms of unbinned pixels. These variables depend upon the particular geometry of the sensor used within the camera.</td>
</tr>
<tr>
<td>OverscanColumns</td>
<td>RO</td>
<td>Long</td>
<td>Returns the number of overscan columns in terms of unbinned pixels. This variable depends upon the particular sensor used within the camera.</td>
</tr>
<tr>
<td>DigitizeOverscan</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>RoiPixelsH, RoiPixelsV</td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>RoiStartX, RoiStartY</td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Binning Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxBinningH, MaxBinningV</td>
<td>RO</td>
<td>Long</td>
<td>Returns the maximum horizontal and vertical binning factors of the device.</td>
</tr>
<tr>
<td>RoiBinningH, RoiBinningV</td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
</tbody>
</table>

### TDI Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDICounter</td>
<td>RO</td>
<td>Long</td>
<td>Dynamically incrementing count during a TDI image. The final value of TDICounter equals TDIRows. Valid range is between 1 and 65535.</td>
</tr>
<tr>
<td>TDIRate</td>
<td>R/W</td>
<td>Double</td>
<td>Incremental rate between TDI rows. Range is from 5.12us to 336ms.</td>
</tr>
<tr>
<td>TDIRows</td>
<td>R/W</td>
<td>Long</td>
<td>Total number of rows in the TDI image. Range is between 1 and 65535.</td>
</tr>
</tbody>
</table>

### Sequence Parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>R/W</td>
<td>Data Type</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ImageCount</td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>SequenceCounter</td>
<td>RO</td>
<td>Long</td>
<td>Dynamically incrementing count during an image sequence. The final value of SequenceCounter equals ImageCount. Valid range is between 1 and 65535.</td>
</tr>
<tr>
<td>SequenceDelay</td>
<td>R/W</td>
<td>Long</td>
<td>Time delay between images of the sequence. Range is from 327us to 21.42s. Dependent on VariableSequenceDelay.</td>
</tr>
<tr>
<td>VariableSequenceDelay</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
</tbody>
</table>

**CCD Settings**

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>RO</td>
<td>Boolean</td>
<td>Returns TRUE is CCD sensor has color dyes; FALSE otherwise</td>
</tr>
<tr>
<td>GainSixteenBit</td>
<td>RO</td>
<td>Double</td>
<td>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.</td>
</tr>
<tr>
<td>GainTwelveBit</td>
<td>R/W</td>
<td>Long</td>
<td>A programmable value to select the actual gain of the camera being used. The default value is based on the particular sensor used with a camera model. 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.</td>
</tr>
<tr>
<td>PixelSizeX,</td>
<td>RO</td>
<td>Double</td>
<td>Returns the size (width and height) of the pixels in micrometers</td>
</tr>
<tr>
<td>PixelSizeY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor</td>
<td>RO</td>
<td>String</td>
<td>Returns the sensor model installed in the camera (i.e. &quot;KAF401E&quot;)</td>
</tr>
<tr>
<td>SensorTypeCCD</td>
<td>RO</td>
<td>Boolean</td>
<td>Returns TRUE if the sensor is a CCD; FALSE if CMOS</td>
</tr>
</tbody>
</table>

**Other**

<table>
<thead>
<tr>
<th>Variable</th>
<th>R/W</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CameraRegister[Index]</td>
<td>R/W</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>Image</td>
<td>RO</td>
<td>Variant</td>
<td>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.</td>
</tr>
<tr>
<td>Line</td>
<td>RO</td>
<td>Variant</td>
<td>Returns a 1D SAFEARRAY of type LONG (4 bytes per element) or INTEGER (2 bytes per element) which contains</td>
</tr>
<tr>
<td>Variable</td>
<td>Access</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ConvertShortToLong</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>OptionBase</td>
<td>R/W</td>
<td>Boolean</td>
<td>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.</td>
</tr>
</tbody>
</table>
6 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:

\[
\text{ShowDialog( Boolean Interactive )}
\]

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.

<table>
<thead>
<tr>
<th>ICamDiscover Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>DlgTitleBarText</td>
</tr>
<tr>
<td>DlgCheckEthernet</td>
</tr>
<tr>
<td>DlgCheckUsb</td>
</tr>
<tr>
<td>DlgNetworkMask</td>
</tr>
<tr>
<td>DlgShowEthernet</td>
</tr>
<tr>
<td>Property</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>DlgShowUsb</td>
</tr>
<tr>
<td>ValidSelection</td>
</tr>
<tr>
<td>SelectedInterface</td>
</tr>
<tr>
<td>SelectedCamIdOne</td>
</tr>
<tr>
<td>SelectedCamIdTwo</td>
</tr>
<tr>
<td>SelectedModel</td>
</tr>
</tbody>
</table>
7 I/O Port Usage

7.1 Overview

Alta camera systems provide an 8 pin MiniDIN connector, enabling various hardware signals to be controlled by the device. 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.

7.2 Hardware Description

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

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.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LVTTL Signal 1</td>
</tr>
<tr>
<td>2</td>
<td>LVTTL Signal 2</td>
</tr>
<tr>
<td>3</td>
<td>LVTTL Signal 3</td>
</tr>
<tr>
<td>4</td>
<td>LVTTL Signal 4</td>
</tr>
<tr>
<td>5</td>
<td>LVTTL Signal 5</td>
</tr>
<tr>
<td>6</td>
<td>LVTTL Signal 6</td>
</tr>
<tr>
<td>7</td>
<td>+12 volt power from the camera head</td>
</tr>
<tr>
<td>8</td>
<td>Ground</td>
</tr>
</tbody>
</table>

7.3 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 *CameraMode* property is set to *Apn_CameraMode_ExternalTrigger*, 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:

![Diagram of Shutter Output and Strobe Output Signals](image)

**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.

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.

**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.

**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 Examples

8.1 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 Alta 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.

```c++
#include <stdio.h>

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

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

    CoInitialize( NULL );      // Initialize COM library

    // Create the ICamera2 object
    hr = AltaCamera.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" );
        AltaCamera = NULL;  // Release ICamera2 COM object
        CoUninitialize();   // Close the COM library
        return;
    }
```
// Set the checkboxes to default to searching both USB and
// ethernet interfaces for Alta cameras
Discover->DlgCheckEthernet = true;
Discover->DlgCheckUsb = true;

// Display the dialog box for finding an Alta 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
    AltaCamera = NULL; // Release ICamera2 COM object
    CoUninitialize(); // Close the COM library
    return;
}

// Initialize camera using the ICamDiscover properties
hr = AltaCamera->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
    AltaCamera = NULL; // Release ICamera2 COM object
    CoUninitialize(); // Close the COM library
    return;
}

// Query the camera for a full frame image
long ImgXSize = AltaCamera->ImagingColumns;
long ImgYSize = AltaCamera->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( AltaCamera->CameraModel );
printf( "Camera Model: %s\n", (char*)szCamModel );

// Display the driver version
_bstr_t szDriverVer( AltaCamera->DriverVersion );
printf( "Driver Version: %s\n", (char*)szDriverVer );
// Do a 0.001s dark frame (bias)
printf( "Starting camera exposure...
" );
AltaCamera->Expose( 0.001, false );

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

// Get the image data from the camera
printf( "Retrieving image data from camera...
" );
AltaCamera->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)
AltaCamera->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)
AltaCamera->IoPortDirection = 0x18;

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

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

```vbnet
Module Module1

    Sub Main()

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

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

        FindDlg.DlgCheckEthernet = True
        FindDlg.DlgCheckUsb = True

        FindDlg.ShowDialog(True)

        If FindDlg.ValidSelection Then

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

            AltaCamera.Expose(0.001, False)

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

            ImageData = AltaCamera.Image

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

        End If
    End Sub

End Module
```
8.3 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 Alta 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.
# 9 Application Notes

## 9.1 Detecting USB Device Removal

The Apogee Alta 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 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 system is returned to the system.

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.

## 9.2 Application Polling and Camera Status

Applications will want to periodically query the Alta 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 \textit{ImagingStatus} property before issuing a call to the \textit{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 \textit{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.

9.3 Retrieving Image Data

Prior to retrieving image data from the camera system, applications should query the \textit{ImagingStatus} property to determine if the image data is ready to be downloaded. This is done by monitoring for a status of \textit{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.

9.4 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.

9.5 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.