AVT PvAPI

Programmer's Reference Manual

V1.28

20 March 2015

Allied Vision Technologies GmbH
Taschenweg 2a
D-07646 Stadtroda, Germany

ALLIED Vision Technologies

Contents

Contacting Allied Vision Technologies

Info

- **Technical information:**
  http://www.alliedvision.com
- **Support:**
  support@alliedvision.com

Allied Vision Technologies GmbH (Headquarters)
Taschenweg 2a
07646 Stadtroda, Germany
Tel: +49 36428-677-0
Fax: +49 36428-677-28
e-mail: info@alliedvision.com

Allied Vision Technologies Canada Inc.
101-3750 North Fraser Way
Burnaby, BC, V5J 5E9, Canada
Tel: +1 604-875-8855
Fax: +1 604-875-8856
e-mail: info@alliedvision.com

Allied Vision Technologies Inc.
38 Washington Street
Newburyport, MA 01950, USA
Toll Free number +1 877-USA-1394
Tel: +1 978-225-2030
Fax: +1 978-225-2029
e-mail: info@alliedvision.com

Allied Vision Technologies Asia Pte. Ltd.
82 Playfair Road
#07-02 D'Lithium, Singapore 368001
Tel: +65 6634-9027
Fax: +65 6634-9029
e-mail: info@alliedvision.com

Allied Vision Technologies (Shanghai) Co. Ltd.
2-2109 Hongwell International Plaza
1602# ZhongShanXi Road, Shanghai 200235, China
Tel: +86 21-64861133
Fax: +86 21-54233670
e-mail: info@alliedvision.com

Introduction

This document is the programmer's reference for Allied Vision Technologies' GigE Vision driver and its Application Programming Interface. Allied Vision Technologies PvAPI interface supports all GigE Vision cameras from Allied Vision Technologies, which includes:

- Bigeye G
- Mako G
- Manta
- Prosilica GB
- Prosilica GC
- Prosilica GE
- Prosilica GS
- Prosilica GT
- Prosilica GX

This document can be applied to all of these families.

Current versions

Component	Current version
SampleViewer	1.28
IPConfig	1.0.0.1
FilterDriver Installer	1.24.15
FilterDriver	1.24.15
PvAPI	1.28

Table 1: Current versions of PvAPI SDK

Driver stack

The PvAPI driver interface is a user DLL which communicates either with the AVT GigE Filter Driver, or kernel networking drivers.

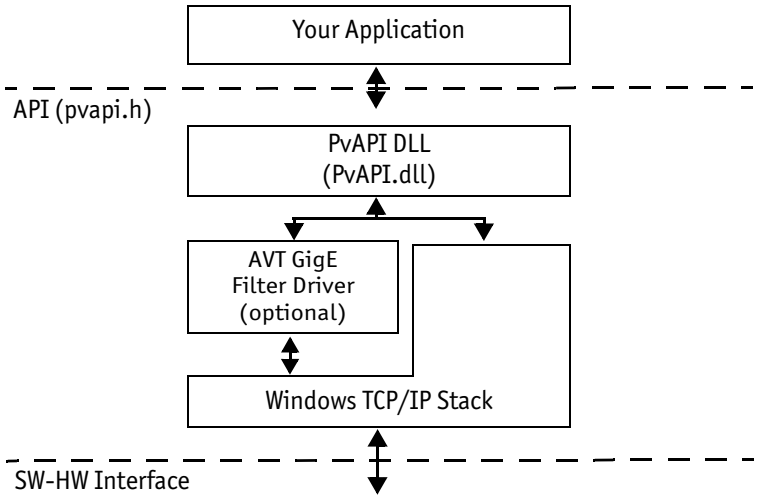

Figure 1: Allied Vision Technologies driver stack

Using the Driver

Platform

PvAPI SDK is supported on the following platforms:

- Windows XP (32bit or 64bit)
- Windows Vista (32bit or 64bit)
- Windows 7 (32bit or 64bit)
- Linux (x86, x64, arm)
- QNX 6.5
- Mac OS X

Programming Languages

PvAPI.dll is a standard-call DLL, which is accessible by most programming languages.

Required C header files, PvAPI.h and PvRegIO.h, are included in the SDK.

Most compiled languages need an import library to call a DLL. An import library, PvAPI.lib, for Microsoft Visual Studio 6.0 (and later) is included in the SDK. Most compilers come with a tool to generate an import library from a DLL; see your compiler's manual for more information.

Threading

The driver is thread-safe, with a few exceptions as noted in this document.

Distribution

The following files may be redistributed for use with AVT cameras only:

- PvAPI.dll
- PvNET.dll
- PvJNI.dll
- psligvfilter.inf
- psligvfilter_m.inf
- psligvfilter.sys
- Allied Vision Technologies GigE Filter Installer.exe
- Allied Vision Technologies Viewer Installer.exe
- libPvAPI.so
- libPvAPI.a
- libImagelib.a

No other files from the SDK may be redistributed without written permission from Allied Vision Technologies.

Driver Installation

PvAPI.dll should be installed in your application's directory.

AVT GigE Filter Driver

Available on Windows only. The AVT GigE Filter driver (see figure 1) is a Windows NDIS (Network Driver Interface Specification) miniport driver. Use of the filter driver increases performance of camera streaming and reduces CPU overhead. It is optional and once installed, the GigE Filter driver will display as a service in Network adapter properties, where you can enable/disable it.

Installation: Run "Allied Vision Technologies GigE Filter Installer.exe". You can use the command line option "/S" to perform a *silent* installation.

Using the API

Example Code

C++, C#, and VB example code is included in the GigESDK /examples directory.

Note

If stepping through code in a debugger, you may need to increase the camera *HeartbeatTimeout* parameter. See GigE Camera and Driver Attributes document for more info.

Module Version

New features may be added to future versions of PvAPI, however PvAPI will always remain backwards compatible. Use *PvVersion* to check the version number of PvAPI.

Module Initialization

Before calling any PvAPI functions (other than *PvVersion*), you must initialize the PvAPI module by calling *PvInitialize*. Please ensure to keep the calling thread alive until you call *PvUninitialize*. For example, call *PvInitialize* in your application's main function.

When you are finished with PvAPI, call *PvUnInitialize* to free resources. These two API functions must always be paired in the same thread. It is possible, although not recommended, to call the pair several times within the same program.

List available cameras

Function *PvCameraListEx* will enumerate all Allied Vision Technologies cameras connected to the system.

Example:

```
tPvCameraInfoEx list[10];
unsigned long numCameras;
numCameras = PvCameraListEx(list, 10, NULL,sizeof(tPvCameraInfoEx));
// Print a list of the connected cameras
for (unsigned long i = 0; i < numCameras; i++)
printf("%s [ID %u]", list[i].SerialNumber, list[i].UniqueId);
```

The *tPvCameraInfoEx* structure provides the following information about a camera:

UniqueId	A value unique to each camera shipped by Allied Vision Technologies
CameraName	People-friendly camera name (usually part name)

ModelName	Name of the camera part
PartNumber	Manufacturer's part number
SerialNumber	Camera's serial number
FirmwareVersion	Camera's firmware version
PermittedAccess	A combination of *tPvAccessFlags*
InterfaceId	Unique value for each interface or bus
InterfaceType	Interface type; see *tPvInterface*

To be notified when a camera is detected or disconnected, use *PvLinkCallbackRegister*. Your callback function must be thread safe.

Opening a camera

A camera must be opened to control and capture images. Function *PvCameraOpen* is used to open the camera.

Example:

```
tPvCameraInfoEx info;
unsigned long numCameras;
tPvHandle handle;

numCameras = PvCameraListEx(info, 1, NULL, sizeof(tPvCameraInfoEx));

// Open the first camera found, if it's not already open. (See
// function reference for PvCameraOpen for a more complex example.)
if ((numCameras == 1) && (info.PermittedAccess & ePvAccessMaster))
 PvCameraOpen(info.UniqueId, ePvAccessMaster, &handle);
```

The camera must be closed when the application is finished.

Camera Attributes

Attributes are used to control and monitor various aspects of the driver and camera.

www

See **GigE Camera and Driver Attributes** document for the complete description of camera attributes.

http://www.alliedvision.com/fileadmin/content/documents/products/cameras/various/features/GigE_Camera_and_Driver_Attributes.pdf

Attribute Type	Set	Get	Range
Enumeration	*PvAttrEnumSet*	*PvAttrEnumGet*	*PvAttrRangeEnum*
Uint32	*PvAttrUint32Set*	*PvAttrUint32Get*	*PvAttrRangeUint32*
Float32	*PvAttrFloat32Set*	*PvAttrFloat32Get*	*PvAttrRangeFloat32*
Int64	*PvAttrInt64Set*	*PvAttrInt64Get*	*PvAttrRangeInt64*
Boolean	*PvAttrBooleanSet*	*PvAttrBooleanGet*	*n/a*

Table 2: Functions for reading and writing attributes

Attribute Type	Set	Get	Range
String	*PvAttrStringSet*	*PvAttrStringGet*	*n/a*
Command	*PvCommand*	*n/a*	*n/a*

Table 2: Functions for reading and writing attributes

PvAPI currently defines the following attribute types (*tPvDatatype*):

Enumeration	A set of values. Values are represented as strings
Uint32	32-bit unsigned value
Float32	32-bit IEEE floating point value
Boolean	A simple Boolean value (true,false)
Int64	64-bit signed value
String	A string (null terminated, char[])
InterfaceType	Interface type; see *tPvInterface*
Command	Valueless; a function executes when the attribute is written

For example, to change the exposure time, set attribute *ExposureValue*:

```
PvAttrUint32Set(Camera, "ExposureValue", 10000); // 10000 ms
```

For example, to read the image size in bytes:

```
// If you want to ensure portable code, you might choose to use
// tPvUint32 or your own typedef, in place of "unsigned long".
unsigned long imageSize;
PvAttrUint32Get(Camera, "TotalBytesPerFrame", &imageSize);
```

Function *PvAttrList* is used to list all attributes available for a camera. This list remains static while the camera is opened. To get information on an attribute, such as its type and access flags, call function *PvAttrInfo*.

PvAPI currently defines the following access flags (*tPvAttributeFlags*):

Read	The attribute may be read.
Write	The attribute may be written.
Volatile	The camera may change the attribute value at any time. For example: *ExposureValue*, because the exposure is constantly changing if the camera is in auto-expose mode.
Constant	The attribute value will never change.

Image Acquisition and Capture

Image capture calls can be divided into two categories:

Host/driver calls:

1. *PvCaptureStart* – initialize the capture stream on driver.

2. *PvCaptureQueueFrame* – queue frame buffer(s). As images arrive from the camera, they are placed in the next frame buffer in the queue, and returned to the user. More on frame queuing in the next section.
3. *PvCaptureEnd* – close the capture stream on driver.

Host/driver calls set up the host/driver to receive data from the camera.

Camera calls:

1. *AcquisitionStart* attribute – readies camera to receive frame triggers.
2. *AcquisitionMode* attribute – determines how many frame triggers the camera receives.
3. *FrameStartTriggerMode* attribute – determines how frames are triggered.
4. *AcquisitionStop* – stops camera from receiving frame triggers.

Camera calls start the camera imaging, and sending data to host.

Example workflow:

```
//start driver stream
PvCaptureStart(Camera);
//queue frame
PvCaptureQueueFrame(Camera, Frame, NULL);
//set frame triggers to be generated internally
PvAttrEnumSet(Camera, "FrameStartTriggerMode", "Freerun");
//set camera to receive continuous number of frame triggers
PvAttrEnumSet(Camera, "AcquisitionMode", "Continuous");
//start camera receiving frame triggers
PvCommandRun(Camera, "AcquisitionStart");

do {
   //wait for frame to return to host
   PvCaptureWaitForFrameDone(Camera, Frame, PVINFINITE);
   //do something with returned frame
   //*****************
   //requeue frame
   PvCaptureQueueFrame(Camera, Frame, NULL);
}
while (some condition);
```

To guarantee a particular image is captured, you must ensure that a frame buffer is queued before the camera is sent a frame trigger.

Frame Queuing

Frames are structures containing image data and related info. See *tPvFrame* in PvApi.h. Users are responsible for managing their own queue of frames. This allows for flexibility in how the queue is managed. Example queues: a 3 buffer circular queue, 100 frame one time use queue.

To create a frame, fill out a *tPvFrame* structure with associated *tPvFrame* → *ImageBuffer* (use attribute *TotalBytesPerFrame* to calculate ImageBuffer size), and place the frame structure on the queue with *PvCaptureQueueFrame*.

Once a *tPvFrame* structure is queued, it can be filled with image data from the camera. There are two mechanisms available to determine when a queued frame has been filled with image data: *PvCaptureWaitForFrameDone*, which blocks your thread until the frame is filled, or by specifying a callback function with *PvCaptureQueueFrame*. Your callback function is run by the driver on a separate thread when image capture is complete.

When a frame is complete, always check that *tPvFrame* → *Status* equals *ePvErrSuccess*, to ensure the data is valid. Lost data over the GigE network will result in *ePvErrDataMissing*, meaning the complete frame has not been received by the host. See the GigE Installation Manual for optimizing GigE networks to prevent missing data.

Most applications need not queue more than 2 or 3 frames at a time, and constantly re-queue the frames. However, if you wish to perform a substantial amount of processing on the image inside a frame callback, you can quickly run into a situation where you are delaying your re-queuing of frames, and images will be returned from the camera with no waiting frame, resulting in a skipped image. I.e. if a current callback is not finished and the next frame is completed, this next frame callback (and all subsequent callbacks) is queued. If you delay long enough in the first callback, all frames are returned and none re-queued.

In this scenario, it may be better to delay processing of the images. You can allocate your own pool of any number of frames, and use your frame callbacks to simply manage frame queuing from this larger pool – delaying image processing until later.

If you want to cancel all the frames on the queue, call *PvCaptureQueueClear*. The status of the frame is set to *ePvErrCancelled* and, if applicable, the callbacks are run.

ModeActive / tPvFrame->AncillaryBuffer

As of latest camera firmware, frames may also receive the associated chunk mode data from the camera:

[Bytes 1 – 4] Acquisition count
[Byte 5] These 8 bits indicate the following EF lens settings: • *Bit 7 (Error)*: When this bit is set to 1, the EF lens is in an error state, bits 2 – 5 indicate enumerated value of last error, and all other bits and Bytes will be 0. • *Bit 6 (Lens attached)*: When this bit is set to 1, an EF lens is attached to camera. • *Bit 5 (Auto focus)*: When this bit is set to 1, the EF lens manual/auto focus switch is set to the auto focus position. • *Bits 2 – 4 (Last error)*: Enumerated error value: – 0: No error detected – 1: Lens failed query by camera – 2: Lens communication error (can occur when removing lens) – 3: Lens communication error (can occur when removing lens) – 4: Lens remained busy for longer than 10 seconds – 5: Lens focus “Zero Stop” not detected – 6: Lens focus “Infinity Stop” not detected • *Bits 0 – 1*: Upper 2 bits of focus percentage value (see **Byte 6**).
[Byte 6] These 8 bits in conjunction with bits 0 – 1 of Byte 5, indicate the current focus position of the EF lens in (percentage of maximum focus range) * 10 (i.e. 1000 = 100 percent = Infinity Stop). If the lens manual/auto focus switch is in the manual position these bits will be 0.
[Byte7] These 8 bits indicate the current aperture position of the EF lens in Dn. To convert Dn to FStop value, use formula: FStop = 2 (Dn – 8) /16.
[Byte 8] These 8 bits indicate the current focal length of the EF lens in mm.
[Bytes 9 – 12] Exposure value in µs.
[Bytes 13 – 16] Gain value in dB.
[Bytes 17 – 18] Sync in levels. A bit field. Bit 0 is sync-in 0, bit 1 is sync-in 1, etc. A bit value of 1 = level high, and a bit value of 0 = level low.
[Bytes 19 – 20] Sync out levels. A bit field. Bit 0 is sync-out 0, bit 1 is sync-out 1, etc. A bit value of 1 = level high, and a bit value of 0 = level low.
[Bytes 21 – 24] Reserved. 0
[Bytes 25 – 28] Reserved. 0
[Bytes 29 – 32] Reserved. 0
[Bytes 33 – 36] Reserved. 0
[Bytes 37 – 40] Reserved. 0
[Bytes 41 – 44] Chunk ID. 1000
[Bytes 45 – 48] Chunk length.

To enable the receiving of this data, allocate your *tPvFrame → AncillaryBuffer* and enable the *ChunkModeActive* attribute. *AncillaryBufferSize* = *NonImagePayloadSize* attribute value, valid when *ChunkModeActive* = *True*.

Error Codes

Most PvAPI functions return a *tPvErr*-type error code.

Typical errors are listed with each function in the reference section of this document. However, any of the following error codes might be returned:

ePvErrSuccess	Success – no error.
ePvErrCameraFault	Unexpected camera fault.
ePvErrInternalFault	Unexpected fault in PvAPI or driver.
ePvErrBadHandle	Camera handle is bad.
ePvErrBadParameter	Function parameter is bad.
ePvErrBadSequence	Incorrect sequence of API calls. For example, queuing a frame before starting image capture.
ePvErrNotFound	Returned by *PvCameraOpen* when the requested camera is not found.
ePvErrAccessDenied	Returned by *PvCameraOpen* when the camera cannot be opened in the requested mode, because it is already in use by another application.
ePvErrUnplugged	Returned when the camera has been unexpectedly unplugged.
ePvErrInvalidSetup	Returned when the user attempts to capture images, but the camera setup is incorrect.
ePvErrResources	Required system or network resources are unavailable.
ePvErrQueueFull	The frame queue is full.
ePvErrBufferTooSmall	The frame buffer is too small to store the image.
ePvErrCancelled	Frame is canceled. This is returned when frames are aborted using *PvCaptureQueueClear*.
ePvErrDataLost	The data for this frame was lost. The contents of the image buffer are invalid.
ePvErrDataMissing	Some of the data in this frame was lost.
ePvErrTimeout	Timeout expired. This is returned only by functions with a specified timeout.
ePvErrOutOfRange	The attribute value is out of range.
ePvErrWrongType	This function cannot access the attribute, because the attribute type is different.
ePvErrForbidden	The attribute cannot be written at this time.
ePvErrUnavailable	The attribute is not available at this time.
ePvErrFirewall	Windows' firewall is blocking the streaming port.

Function Reference

PvAttrBooleanGet

Get the value of a Boolean attribute.

Prototype

```
tPvErr PvAttrBooleanGet
(
  tPvHandle       Camera,
  const char*     Name,
  tPvBoolean*     pValue
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pValue	Value is returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Boolean type

PvAttrBooleanSet

Set the value of a Boolean attribute.

Prototype

```
tPvErr PvAttrBooleanSet
(
  tPvHandle        Camera,
  const char*      Name,
  tPvBoolean       Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	Value to set

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrOutOfRange	The value is out of range at this time
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Boolean type

PvAttrEnumGet

Get the value of an enumeration attribute.

Prototype

```
tPvErr PvAttrEnumGet
(
  tPvHandle        Camera,
  const char*      Name,
  char*            pBuffer,
  unsigned long    BufferSize,
  unsigned long*   pSize
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pBuffer	The value string (always null terminated) is copied here. This buffer is allocated by the caller
BufferSize	The size of the allocated buffer
pSize	The size of the value string is returned here. This may be bigger than *BufferSize*. Null pointer is allowed

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an enumeration type

PvAttrEnumSet

Set the value of an enumeration attribute.

Prototype

```
tPvErr PvAttrEnumSet
(
  tPvHandle         Camera,
  const char*       Name,
  const char*       Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	The enumeration value (a null terminated string)

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrOutOfRange	The value is not a member of the current enumeration set
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an enumeration type

PvAttrExists

Query: does an attribute exist?

Prototype

```
tPvErr PvAttrExists
(
  tPvHandle        Camera,
  const char*      Name
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	The attribute exists
ePvErrNotFound	The attribute does not exist

Notes

The result of this query is static for this camera; it won't change while the camera is open.

PvAttrFloat32Get

Get the value of a Float32 attribute.

Prototype

```
tPvErr PvAttrFloat32Get
(
  tPvHandle        Camera,
  const char*      Name,
  tPvFloat32*      pValue
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pValue	Value is returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Float32 type

PvAttrFloat32Set

Set the value of a Float32 attribute.

Prototype

```
tPvErr PvAttrFloat32Set
(
  tPvHandle        Camera,
  const char*      Name,
  tPvFloat32       Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	Value to set

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrOutOfRange	The value is out of range at this time
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Float32 type

PvAttrInfo

Get information, such as data type and access mode, on a particular attribute.

Prototype

```
tPvErr PvAttrInfo
(
  tPvHandle        Camera,
  const char*      Name,
  tPvAttributeInfo*pInfo
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pInfo	The attribute information is copied here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist

PvAttrInt64Get

Get the value of an Int64 attribute.

Prototype

```
tPvErr PvAttrInt64Get
(
  tPvHandle         Camera,
  const char*       Name,
  tPvInt64*         pValue
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pValue	Value is returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an Int64 type

PvAttrInt64Set

Set the value of an Int64 attribute.

Prototype

```
tPvErr PvAttrInt64Set
(
  tPvHandle        Camera,
  const char*      Name,
  tPvInt64         Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	Value to set

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrOutOfRange	The value is out of range at this time
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an Int64 type

PvAttrIsAvailable

Query: is the attribute available at this time / for this camera model?

Prototype

```
tPvErr PvAttrIsAvailable
(
  tPvHandle        Camera,
  const char*      Name
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	The attribute is available
ePvErrUnavailable	The attribute is unavailable at this time
ePvErrNotFound	The attribute does not exist

Notes

If an attribute is unavailable, it means the attribute cannot be read or changed. The result of this query is dynamic. The availability of a particular attribute may change at any time, depending on the state of the camera and the values of other attributes.

PvAttrIsValid

Query: is the value of an attribute valid / within range?

Prototype

```
tPvErr PvAttrIsValid
(
  tPvHandle        Camera,
  const char*      Name
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	The attribute value is in range
ePvErrOutOfRange	The attribute value is out of range
ePvErrNotFound	The attribute does not exist

PvAttrList

List all camera and driver attributes.

Prototype

```
tPvErr PvAttrList
(
   tPvHandle        Camera,
   tPvAttrListPtr*  pListPtr,
   unsigned long*   pLength
);
```

Parameters

Camera	Handle to open camera
pListPtr	The pointer to the attribute list is returned here. The attribute list is owned by the PvAPI module, and remains static while the camera is opened. The attribute list is an array of string pointers
pLength	The length of the attribute list is returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful

Example

List the available attributes:

```
tPvAttrListPtr listPtr;
unsigned long listLength;

if (PvAttrList(Camera, &listPtr, &listLength) == ePvErrSuccess)
{
   for (int i = 0; i < listLength; i++)
   {
      const char* attributeName = listPtr[i];
      printf("Attribute %s\n", attributeName);
   }
}
```

PvAttrRangeEnum

Get the set of values for an enumerated attribute.

Prototype

```
tPvErr PvAttrRangeEnum
(
   tPvHandle         Camera,
   const char*       Name,
   char*             pBuffer,
   unsigned long     BufferSize,
   unsigned long*    pSize
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pBuffer	A comma separated string (no white-space, always null terminated), representing the enumeration set, is copied here. This buffer is allocated by the caller
BufferSize	The size of the allocated buffer
pSize	The size of the enumeration set string is returned here. This may be bigger than *BufferSize*. Null pointer is allowed

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an enumeration type
ePvErrBadParameter	The supplied buffer is too small to fit the string

Notes

The enumeration set is dynamic. For some attributes, the set may change under various circumstances.

Example

List the acquisition modes (for clarity we use strtok, but please research its limitations):

```
charenumSet[1000];
if (PvAttrRangeEnum(Camera, "AcquisitionMode",
                    enumSet, sizeof(enumSet), NULL) == ePvErrSuccess)
{
   char* member = strtok(enumSet, ",");
   // strtok isn't always thread safe!
   while (member != NULL)
   {
      printf("Mode %s\n", member);
      member = strtok(NULL, ",");
   }
}
```

PvAttrRangeFloat32

Get the value range of a Float32 attribute.

Prototype

```
tPvErr PvAttrRangeFloat32
(
  tPvHandle        Camera,
  const char*      Name,
  tPvFloat32*      pMin,
  tPvFloat32*      pMax
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pMin	Minimum value returned here
pMax	Maximum value returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Float32 type

Notes

In some cases, the value range is dynamic.

PvAttrRangeInt64

Get the value range of an Int64 attribute.

Prototype

```
tPvErr PvAttrRangeInt64
(
  tPvHandle        Camera,
  const char*      Name,
  tPvInt64*        pMin,
  tPvInt64*        pMax
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pMin	Minimum value returned here
pMax	Maximum value returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not an Int64 type

Notes

In some cases, the value range is dynamic.

PvAttrRangeUint32

Get the value range of a Uint32 attribute.

Prototype

```
tPvErr PvAttrRangeUint32
(
  tPvHandle         Camera,
  const char*       Name,
  tPvUint32*        pMin,
  tPvUint32*        pMax
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pMin	Minimum value returned here
pMax	Maximum value returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Uint32 type

Notes

In some cases, the value range is dynamic.

PvAttrStringGet

Get the value of a string attribute.

Prototype

```
tPvErr PvAttrStringGet
(
   tPvHandle        Camera,
   const char*      Name,
   char*            pBuffer,
   unsigned long    BufferSize,
   unsigned long*   pSize
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pBuffer	The value string (always null terminated) is copied here. This buffer is allocated by the caller
BufferSize	The size of the allocated buffer
pSize	The size of the value string is returned here. This may be bigger than *BufferSize*. Null pointer is allowed

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a string type

PvAttrStringSet

Set the value of a string attribute.

Prototype

```
tPvErr PvStringSet
(
  tPvHandle        Camera,
  const char*      Name,
  const char*      Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	The string value (always null terminated)

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a string type

PvAttrUint32Get

Get the value of a Uint32 attribute.

Prototype

```
tPvErr PvAttrUint32Get
(
  tPvHandle         Camera,
  const char*       Name,
  tPvUint32*        pValue
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
pValue	Value is returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Uint32 type

PvAttrUint32Set

Set the value of a Uint32 attribute.

Prototype

```
tPvErr PvAttrUint32Set
(
  tPvHandle        Camera,
  const char*      Name,
  tPvUint32        Value
);
```

Parameters

Camera	Handle to open camera
Name	Attribute name
Value	Value to set

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrOutOfRange	The value is out of range at this time
ePvErrForbidden	The attribute cannot be set at this time
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a Uint32 type

PvCameraClose

Close a camera.

Prototype

```
tPvErr PvCameraClose
(
  tPvHandle        Camera
);
```

Parameters

Camera	Handle to open camera

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrBadHandle	Camera handle is bad

Notes

Open cameras should always be closed, even if they have been unplugged.

PvCameraCount

Get the number of Allied Vision Technologies cameras visible to this system.

Prototype

```
unsigned long PvCameraCount
(
   void
);
```

Parameters

None.

Return Value

The number of cameras visible to the system.

Notes

This returns the number of reachable cameras at the time the call is made. This number is dynamic, and will change as cameras become available/unavailable. Unreachable cameras, i.e. cameras on a different subnet than the host NIC, are not counted.

See *PvInitialize* for usage.

Example

See example for *PvInitialize*.

PvCameraEventCallbackRegister

Register a callback for any camera specific events.

Prototype

```
tPvErr PvCameraEventCallbackRegister
(
   tPvHandle         Camera,
   tPvCameraEventCallback Callback,
   void*             Context
);
```

Parameters

Camera	Handle to open camera
Callback	Callback function to be registered
Context	Defined by the caller. Passed to your callback

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

Callback will be issued for any/all enabled events. To enable an event see the *EventNotification* and *EventSelector* attributes.

In the callback function, see the *EventID* for each element of the *EventList* parameter to determine which event(s) are associated with the callback. EventID corresponds to the Uint32 value of *EventID* attribute. E.g. *EventAcquistionStart* = 40000.

Caution

It is possible to enable many events simultaneously, resulting in callback rates that can go beyond the frame rate of the camera.

Ensure that your code and system resources can handle these rates.

PvCameraEventCallbackUnregister

Unregister a callback for any camera specific events.

Prototype

```
tPvErr PvCameraEventCallbackUnregister
(
  tPvHandle        Camera,
  tPvCameraEventCallback Callback,
  void*            Context
);
```

Parameters

Camera	Handle to open camera
Callback	Callback function to be unregistered
Context	Defined by the caller. Passed to your callback

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

Unregistering a callback for events will not cause the camera to stop sending events. To disable an event see the *EventNotification* and *EventSelector* attributes.

PvCameraForceIP

Force the IP settings for an Ethernet camera. This command will work for all cameras on the local Ethernet network, including “unreachable” cameras or cameras with an invalid IP address (e.g. 0.0.0.253).

Prototype

```
tPvErr PvCameraForceIP
(
  Const char*      pMAC,
  unsigned long    Address,
  unsigned long    Subnet,
  unsigned long    Gateway
);
```

Parameters

pMAC	MAC address of the camera
Address	Static IP address to be assigned
Subnet	Subnet mask to be assigned
Gateway	Gateway address to be assigned

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	No error
ePvErrInternalFault	An internal fault occurred
ePvErrBadSequence	API is not initialized

Example

See *ForceCamera* example code.

PvCameraInfoEx

Get information on a specified camera.

Prototype

```
tPvErr PvCameraInfoEx
(
  unsigned long    UniqueId,
  tPvCameraInfoEx* pInfo,
  unsigned long    Size
);
```

Parameters

UniqueId	Unique ID of camera
pInfo	Camera information is returned here
Size	Size of the tPvCameraInfoEx structure

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

The specified camera must be visible to the system (i.e. on a local subnet), and using Allied Vision Technologies's driver.

See *PvCameraListEx* if you want to retrieve information for all cameras.

PvCameraInfoByAddrEx

Get information on a camera, specified by its IP address. This function is required if the GigE camera is not on the local IP subnet.

Prototype

```
tPvErr PvCameraInfoByAddrEx
(
   unsigned long    IpAddr,
   tPvCameraInfoEx* pInfo,
   tPvIpSettings*   pIpSettings,
   unsigned long    Size
);
```

Parameters

IpAddr	IP address of camera, in network byte order
pInfo	Camera information is returned here
pIpSettings	Camera IP settings is returned here. See PvApi.h
Size	Size of the tPvCameraInfoEx structure

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

This function works if a camera is on the other side of an IP gateway. In this case, the camera's IP address must be known, because it will not be visible to either *PvCameraListEx* or *PvCameraListUnreachableEx*.

PvCameraIpSettingsChange

Change the IP settings for a GigE Vision camera. This command will work for all cameras on the local Ethernet network, including “unreachable” cameras.

Prototype

```
tPvErr PvCameraIpSettingsChange
(
  unsigned long    UniqueId,
  const tPvIpSettings* pIpSettings
);
```

Parameters

UniqueId	Unique ID of camera
pIpSettings	Camera IP settings to be applied to the camera. See PvApi.h

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

All IP related fields in the *tPvIpSettings* structure are in network byte order. This command will not work for cameras accessed through an IP router.

PvCameraIpSettingsGet

Get the IP settings for a GigE Vision camera. This command will work for all cameras on the local Ethernet network, including “unreachable” cameras.

Prototype

```
tPvErr PvCameraIpSettingsGet
(
  unsigned long    UniqueId,
  tPvIpSettings*   pIpSettings
);
```

Parameters

UniqueId	Unique ID of camera
pIpSettings	Camera IP settings is returned here. See PvApi.h

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The specified camera could not be found

Notes

All IP related fields in the *tPvIpSettings* structure are in network byte order. This command will not work for cameras accessed through an IP router.

PvCameraListEx

List the Allied Vision Technologies cameras currently visible to this system.

Prototype

```
unsigned long PvCameraListEx
(
   tPvCameraInfoEx* pList,
   unsigned long    ListLength,
   unsigned long*   pConnectedNum,
   unsigned long    Size
);
```

Parameters

pList	Array of *tPvCameraInfoEx*, allocated by the caller. The camera list is returned in this array
ListLength	Length of *pList* array
pConnectedNum	The number of cameras found is returned here. This may be greater than *ListLength*. Null pointer is allowed
Size	Size of the tPvCameraInfoEx structure

Return Value

Number of *pList* array entries filled, up to *ListLength*.

Notes

Lists only the cameras which are turned on and using Allied Vision Technologies's drivers. If you expect a particular camera to be present, alternatively you can use *PvCameraInfoEx* to retrieve more information.

Example

See example for *PvCameraOpen*.

PvCameraListUnreachableEx

List all the cameras currently inaccessible by PvAPI. This lists the GigE Vision cameras which are connected to the local Ethernet network, but are on a different subnet.

Prototype

```
unsigned long PvCameraListUnreachableEx
(
   tPvCameraInfoEx* pList,
   unsigned long    ListLength,
   unsigned long*   pConnectedNum,
   unsigned long    Size
);
```

Parameters

pList	Array of *tPvCameraInfoEx*, allocated by the caller. The camera list is returned in this array
ListLength	Length of *pList* array
pConnectedNum	The number of cameras found is returned here. This may be greater than *ListLength*. Null pointer is allowed
Size	Size of the *tPvCameraInfoEx* structure

Return Value

Number of *pList* array entries filled, up to *ListLength*.

Notes

Lists only the cameras which are turned on, and connected to the local Ethernet network but on an inaccessible IP subnet. Usually this means the camera's IP settings are invalid. If you expect a particular camera to exist on a different subnet, use *PvCameraInfoByAddrEx* to retrieve more information.

Example

See example for *PvCameraOpen*.

PvCameraOpen

Open a camera.

Prototype

```
tPvErr PvCameraOpen
(
  unsigned long    UniqueId,
  tPvAccessFlags   AccessFlag,
  tPvHandle*       pCamera
);
```

Parameters

UniqueId	Camera's unique ID. This might be acquired through a previous call to *PvCameraListEx*
AccessFlag	Access mode: monitor (listen only) or master (full control)
pCamera	Handle to open camera returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrAccessDenied	Camera could not be opened in the requested access mode, because another application (possibly on another host) is using the camera
ePvErrNotFound	Camera with the specified unique ID is not found. You will also get this error if the camera was unplugged between *PvCameraListEx* and *PvCameraOpen*

Notes

Camera must be closed PvCameraClose on page 35 when no longer required.

Example

```
tPvHandle OpenFirstCamera(void)
{
 tPvCameraInfoEx list[10];
 unsigned long numCameras;
 // List available cameras.
 numCameras = PvCameraListEx(list, 10, NULL,sizeof(tPvCameraInfoEx));
 for (unsigned long i = 0; i < numCameras; i++)
 {
   // Find the first unopened camera...
   if (list[i].PermittedAccess == ePvAccessMaster)
   {
    tPvHandlehandle;
    // Open the camera.
    if (PvCameraOpen(list[i].UniqueId, &handle) == ePvErrSuccess)
    return handle;
   }
 }
 return 0;
}
```

PvCameraOpenByAddr

Open a camera using its IP address. This function can be used to open a GigE Vision camera located on a different IP subnet.

Prototype

```
tPvErr PvCameraOpen
(
  unsigned long    IpAddr,
  tPvAccessFlags   AccessFlag,
  tPvHandle*       pCamera
);
```

Parameters

IpAddr	Camera's IP address, in network byte order
AccessFlag	Access mode: monitor (listen only) or master (full control)
pCamera	Handle to open camera returned here

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrAccessDenied	Camera could not be opened in the requested access mode, because another application (possibly on another host) is using the camera
ePvErrNotFound	Camera with the specified IP address is not found. You will also get this error if the camera was unplugged between *PvCameraListUnreachableEx* and *PvCameraOpenByAddr*

Notes

Camera must be closed (see *PvCameraClose*) when no longer required.

PvCaptureAdjustPacketSize

Function will determine the maximum packet size supported by the system (Ethernet adapter) and then configure the camera to use this value.

Prototype

```
tPvErr PvCaptureAdjustPacketSize
(
  tPvHandle         Camera,
  unsigned long     MaximumPacketSize
);
```

Parameters

Camera	Handle to open camera
MaximumPacketSize	Upper limit: the packet size will not be set higher than this value

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged
ePvErrBadSequence	Capture already started

Notes

This cannot be called when a capture is in progress.

On power up, Allied Vision Technologies cameras have a packet size of 8228. If your network does not support this packet size, and you haven't called *PvCaptureAdjustPacketSize* to detect and set the maximum possible packet size, you will see dropped frames.

PvCaptureEnd

Shut down the image capture stream. This resets the FrameCount parameter.

Prototype

```
tPvErr PvCaptureEnd
(
  tPvHandle        Camera,
);
```

Parameters

Camera	Handle to open camera

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged

Notes

This cannot be called until the capture queue is empty. Function *PvCaptureQueueClear* can be used to cancel all remaining frames.

PvCaptureQuery

Query: has the image capture stream been started? That is, has *PvCaptureStart* been called?

Prototype

```
tPvErr PvCaptureQuery
(
  tPvHandle        Camera,
  unsigned long*   pIsStarted
);
```

Parameters

Camera	Handle to open camera
pIsStarted	Has the capture stream been started? 1=yes, 0=no

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged

PvCaptureQueueClear

Clear queued frames. Aborts actively written frame with *pFrame* → *Status* = *ePvErrDataMissing*. Further queued frames returned with *pFrame* → *Status* = *ePvErrCancelled*.

Prototype

```
tPvErr PvCaptureQueueClear
(
   tPvHandle        Camera
);
```

Parameters

Camera	Handle to open camera

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged

Notes

All applicable frame callbacks are run. After this call completes, all frame callbacks are complete.

This function cannot be run from a frame callback. See *PvCaptureQueueFrame* for details.

The completion timing of *PvCaptureWaitForFrameDone* is indeterminate, i.e. it may or may not complete before *PvCaptureQueueClear* completes.

If a frame is queued while *PvCaptureQueueClear* is ongoing, *PvCaptureQueueFrame* will return *ePvErrBadSequence*. Once *PvCaptureQueueClear* is complete, you can re-queue frames. If using frame callbacks, check that *pFrame→Status* != *ePvErrCancelled* before re-queuing frames.

PvCaptureQueueFrame

Place an image buffer onto the frame queue. This function returns immediately; it does not wait until the frame has been captured.

Prototype

```
tPvErr PvCaptureQueueFrame
(
  tPvHandle        Camera,
  tPvFrame*        pFrame,
  tPvFrameCallback Callback
);
```

Parameters

Camera	Handle to open camera
pFrame	Frame structure which describes the frame buffer. This structure, unique to each queued frame, must persist until the frame has been captured
Callback	Callback to run when the frame has been completed (either successfully, or in error). Optional; null pointer is allowed

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged
ePvErrBadSequence	You cannot queue frames until the capture stream has started
ePvErrQueueFull	The frame queue is full

Notes

PvCaptureQueueFrame cannot be called until the image capture stream has started.

PvCaptureQueueFrame enables the capture of an acquired frame, but it does not trigger the acquisition; see attributes *AcquisitionMode, AcquisitionStart*, and *AcquisitionStop*.

Before you call *PvCaptureQueueFrame*, these frame structure fields must be filled:

ImageBuffer	Pointer to your allocated image buffer. The allocated image buffer may be larger than required
ImageBufferSize	Size of your image buffer, in bytes
AncillaryBuffer	Pointer to your allocated ancillary buffer, if *AncillaryBufferSize* is non-zero
AncillaryBufferSize	Size of your ancillary buffer, in bytes. Can be 0

The use of field *Context[4]* is defined by the caller.

When the frame is complete, these fields are filled by the driver:

Status	*tPvErr* type error code.
ImageSize	Size of this frame, in bytes. May be smaller than *BufferSize*.
AncillarySize	Ancillary data size, in bytes.
Width	Width of this frame.
Height	Height of this frame.
RegionX	Start of readout region, left.
RegionY	Start of readout region, top.
Format	Format of this frame (see *tPvImageFormat*).
BitDepth	Bit depth of this frame.
BayerPattern	Bayer pattern, if applicable.
FrameCount	Rolling frame counter. For GigE Vision cameras, this corresponds to the block number, which rolls from 1 to 0xFFFF. Reset on *PvCaptureEnd*.
Timestamp	Time of exposure-start, in timestamp units.

PvCaptureQueueFrame is an asynchronous capture mechanism; it returns immediately, rather than waiting for a frame to complete.

To determine when a frame is complete, use one of these mechanisms:

1. Call *PvCaptureWaitForFrameDone*: The function *PvCaptureWaitForFrameDone* blocks the calling thread until the frame is complete.
2. Use a callback: When the frame is complete, the callback is run on an internal PvAPI thread. When the callback starts, the frame is complete and you are free to deallocate both the frame structure and the image buffer. The supplied callback function must be thread-safe. Note that *PvCaptureQueueClear* cannot be run from the callback.

To cancel all the frames on the queue, see *PvCaptureQueueClear*.

PvCaptureStart

Start the image capture stream. This initializes both the camera and the host in preparation to capture acquired images.

Prototype

```
tPvErr PvCaptureStart
(
  tPvHandle        Camera
);
```

Parameters

Camera	Handle to open camera

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrUnplugged	Camera was unplugged
ePvErrResources	Required system resources were not available
ePvErrBandwidth	Insufficient bandwidth to start image capture stream

Notes

As images arrive from the camera, they are stored in the buffer at the head of the frame queue. To submit buffers to the frame queue, call *PvCaptureQueueFrame*.

This function does not start image acquisition on the camera; rather, it establishes the data stream. To control image acquisition, see attributes *AcquisitionMode*, *AcquisitionStart,* and *AcquisitionStop*.

PvCaptureWaitForFrameDone

Block the calling thread until a frame is complete.

Prototype

```
tPvErr PvCaptureWaitForFrameDone
(
   tPvHandle        Camera,
   const tPvFrame*  pFrame,
   unsigned long    Timeout
);
```

Parameters

Camera	Handle to open camera
pFrame	Frame structure, as passed to *PvCaptureQueueFrame*
Timeout	Timeout, in milliseconds. Use *PVINFINITE* for no timeout

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful, or *pFrame* is not on the queue
ePvErrUnplugged	Camera was unplugged
ePvErrTimeout	Timeout occurred before exposure completed

Notes

This function cannot be run from the frame-done callback.

This function waits until the frame is complete. When this function completes, you may delete or modify your frame structure, and use the contents of the image buffer.

If *pFrame* is not on the frame queue, *ePvErrSuccess* is returned. The driver must assume that if the frame buffer is not on the queue, it is already complete.

PvCommandRun

Run a command. A command is a “valueless” attribute, which executes a function when written.

Prototype

```
tPvErr PvCommandRun
(
  tPvHandle        Camera,
  const char*      Name
);
```

Parameters

Camera	Handle to open camera
Name	Command (attribute) name

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	The attribute does not exist
ePvErrWrongType	The attribute is not a command type

PvInitialize

Initialize the PvAPI module. You can't call any PvAPI functions, other than *PvVersion*, until the module is initialized.

Prototype

```
tPvErr PvInitialize
(
   void
);
```

Parameters

None.

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrResources	Some required system resources were not available

Notes

After initialization, the PvAPI module will asynchronously search for connected cameras. It may take some time for cameras to show up, therefore check that *PvCameraCount()* does not return 0 before proceeding with a *PvCameraListEx* call.

Example

```
tPvCameraInfoEx list;

if(PvInitialize() == ePvErrSuccess)
{
 while(PvCameraCount() == 0)
 Sleep(250);        // wait for any camera

 PvCameraListEx(&list,1,NULL,sizeof(tPvCameraInfoEx));

 /* … */
}
else
 printf("failed to initialize the API\n");
```

PvInitializeNoDiscovery

Initialize the PvAPI module. You can't call any PvAPI functions, other than *PvVersion*, until the module is initialized.

Prototype

```
tPvErr PvInitializeNoDiscovery
(
   void
);
```

Parameters

None.

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrResources	Some required system resources were not available

Notes

Using this function instead of *PvInitialize* will cause the driver to not send regular discovery broadcast. You will have to rely on knowing the IP of the cameras to access them.

PvLinkCallbackRegister

Register a callback for link (interface) events, such as detecting when a camera is plugged in. When the event occurs, the callback is run.

Prototype

```
tPvErr PvLinkCallbackRegister
(
   tPvLinkCallback  Callback,
   tPvLinkEvent     Event,
   void*            Context
);
```

Parameters

Callback	Callback to run. Must be thread safe
Event	Event of interest
Context	Defined by the caller. Passed to your callback

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful

Notes

Multiple callback functions may be registered with the same event.

The same callback function may be shared by different events.

It is an error to register the same callback function with the same event twice.

Callback must be un-registered by *PvLinkCallbackUnRegister* when no longer required.

PvLinkCallbackUnRegister

Un-register a link event callback.

Prototype

```
tPvErr PvLinkCallbackUnRegister
(
  tPvLinkCallback  Callback,
  tPvLinkEvent     Event
);
```

Parameters

Callback	Callback to run. Must be thread safe
Event	Event of interest

Return Value

tPvErr type error code. Typical error codes for this function:

ePvErrSuccess	Function successful
ePvErrNotFound	Callback/event is not registered

PvUnInitialize

Un-initialize the PvAPI module. This frees system resources used by PvAPI.

Prototype

```
void PvUnInitialize
(
   void
);
```

Parameters

None.

Return Value

None.

PvUtilityColorInterpolate

Perform Bayer-color interpolation on raw Bayer images. This algorithm uses the average value of surrounding pixels.

Prototype

```
void PvUtilityColorInterpolate
(
   const tPvFrame*  pFrame,
   void*            BufferRed,
   void*            BufferGreen
   void*            BufferBlue,
   unsigned long    PixelPadding,
   unsigned long    LinePadding
);
```

Parameters

pFrame	Raw Bayer image, i.e. source data
BufferRed	Output buffer, pointer to the first red pixel. This buffer is allocated by the caller
BufferGreen	Output buffer, pointer to the first green pixel. This buffer is allocated by the caller
BufferBlue	Output buffer, pointer to the first blue pixel. This buffer is allocated by the caller
PixelPadding	Padding after each pixel written to the output buffer, in pixels. In other words, the output pointers skip by this amount after each pixel is written to the caller's buffer. Typical values: • RGB or BGR output:2 • RGBA or BGRA output:3 • planar output:0
LinePadding	Padding after each line written to the output buffers, in pixels

Return Value

None.

Example

Generating a Windows *Win32::StretchDIBits* compatible BGR buffer from a *Bayer8* frame:

```
#define ULONG_PADDING(x)(((x+3) & ~3) - x)

unsigned long line_padding = ULONG_PADDING(frame.Width*3);
unsigned long line_size = ((frame.Width*3) + line_padding;
unsigned long buffer_size = line_size * frame.Height;

ASSERT(frame.Format == ePvFmtBayer8);

unsigned char* buffer = new unsigned char[buffer_size];
PvUtilityColorInterpolate(&frame, &buffer[2], &buffer[1],&buffer[0],
2, line_padding);
```

PvVersion

Return the version number of the PvAPI module.

Prototype

```
void PvVersion
(
  unsigned long*   pMajor,
  unsigned long*   pMinor
);
```

Parameters

pMajor	Major version number returned here
pMinor	Minor version number returned here

Notes

This function may be called at any time.

For technical support, please contact support@alliedvision.com.
For comments or suggestions regarding this document, please contact info@alliedvision.com.

Disclaimer

Due to continual product development, technical specifications may be subject to change without notice. All trademarks are acknowledged as property of their respective owners. We are convinced that this information is correct. We acknowledge that it may not be comprehensive. Nevertheless, AVT cannot be held responsible for any damage in equipment or subsequent loss of data or whatsoever in consequence of this document.