



AVT PvAPI

Programmers' Reference Manual

Version 1.26
September 27, 2011

Table of Contents

Table of Contents.....	ii
Overview	1
Using the Driver	2
Platform	2
Programming Languages.....	2
Threading	2
Distribution.....	2
Driver Installation	3
Using the API	4
Example Code	4
Module Version	4
Module Initialization	4
List available cameras	4
Opening a camera.....	5
Camera Attributes.....	5
Image Acquisition and Capture	7
Frame Queuing	8
ChunkModeActive / tPvFrame->AncillaryBuffer.....	9
Error Codes.....	9
Function Reference.....	11
PvAttrBooleanGet	12
PvAttrBooleanSet	13
PvAttrEnumGet.....	14
PvAttrEnumSet.....	15
PvAttrExists	16
PvAttrFloat32Get	17
PvAttrFloat32Set	18
PvAttrInfo	19
PvAttrInt64Get.....	20
PvAttrInt64Set	21
PvAttrIsAvailable	22
PvAttrIsValid.....	23

PvAttrList.....	24
PvAttrRangeEnum	25
PvAttrRangeFloat32	27
PvAttrRangeInt64	28
PvAttrRangeUint32	29
PvAttrStringGet	30
PvAttrStringSet	31
PvAttrUint32Get	32
PvAttrUint32Set	33
PvCameraClose	34
PvCameraCount	35
PvCameraEventCallbackRegister.....	36
PvCameraEventCallbackUnregister.....	37
PvCameraInfoEx	38
PvCameraInfoByAddrEx.....	39
PvCameraIpSettingsChange.....	40
PvCameraIpSettingsGet.....	41
PvCameraListEx	42
PvCameraListUnreachableEx.....	43
PvCameraOpen	44
PvCameraOpenByAddr.....	46
PvCaptureAdjustPacketSize	47
PvCaptureEnd.....	48
PvCaptureQuery	49
PvCaptureQueueClear.....	50
PvCaptureQueueFrame	51
PvCaptureStart	53
PvCaptureWaitForFrameDone.....	54
PvCommandRun.....	55
PvInitialize.....	56
PvInitializeNoDiscovery.....	57
PvLinkCallbackRegister	58
PvLinkCallbackUnRegister.....	59
PvUnInitialize	60
PvUtilityColorInterpolate	61
PvVersion	63

Overview

This document is the programmer's reference for Allied Vision Technologies' GigE Vision driver and its Application Programming Interface.

The Allied Vision Technologies PvAPI interface supports all GigE Vision cameras from Allied Vision Technologies.

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

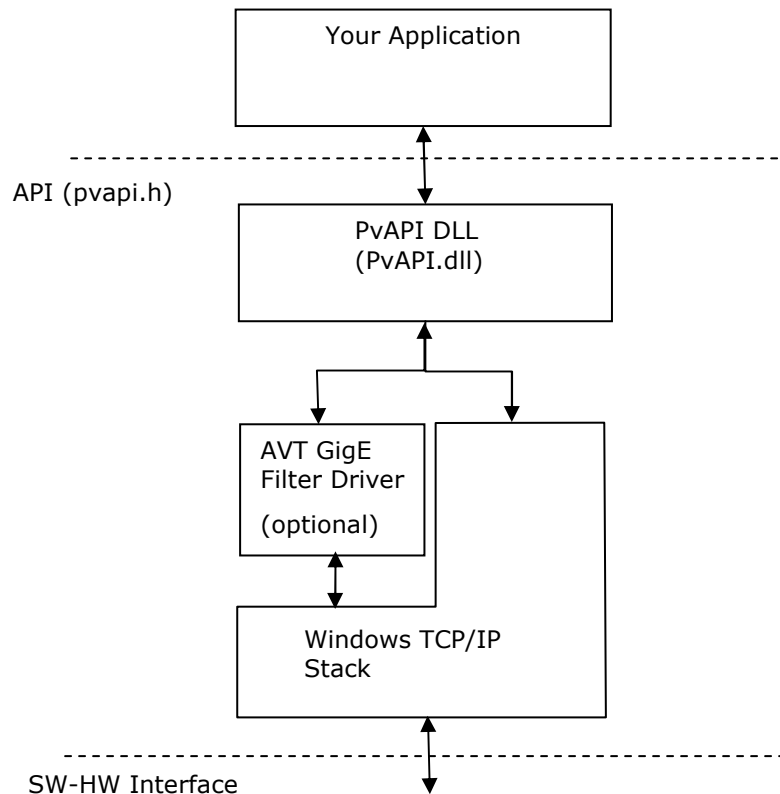


Figure 1. Allied Vision Technologies driver stack.

Using the Driver

Platform

PvAPI is supported on the following platforms:

- Windows 2000
- Windows XP Professional or Home (32bit or 64bit)
- Windows Vista (32bit or 64bit)
- Windows 7 (32bit or 64bit)
- Linux (x86, PPC, x64, arm)
- QNX 6.3 (x86), 6.3 + Core Networking 6.4, 6.4 Beta
- Mac OS X (x86, PPC 32bit, x64)

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 Prosilica/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.

There are two mechanisms for installing the GigE Filter driver:

1. Run "Allied Vision Technologies GigE Filter Installer.exe". You can use the command line option `/S` to perform a *silent* installation.
2. Install the following files:
 - pslignvfilter.sys - Copy to %system32%\drivers
 - pslignvfilter.inf - Copy to %windir%\inf
 - pslignvfilter_m.inf - Copy to %windir%\inf

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 “AVT 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*.

When you are finished with PvAPI, call *PvUnInitialize* to free resources. These two API functions must always be paired. 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.

See “AVT Camera and Driver Attributes” document for the complete description of camera attributes.

Table 1. Functions for reading and writing attributes.

Attribute Type	Set	Get	Range
Enumeration	<i>PvAttrEnumSet</i>	<i>PvAttrEnumGet</i>	<i>PvAttrRangeEnum</i>
UInt32	<i>PvAttrUInt32Set</i>	<i>PvAttrUInt32Get</i>	<i>PvAttrRangeUInt32</i>
Float32	<i>PvAttrFloat32Set</i>	<i>PvAttrFloat32Get</i>	<i>PvAttrRangeFloat32</i>
Int64	<i>PvAttrInt64Set</i>	<i>PvAttrInt64Get</i>	<i>PvAttrRangeInt64</i>
Boolean	<i>PvAttrBooleanSet</i>	<i>PvAttrBooleanGet</i>	n/a
String	<i>PvAttrStringSet</i>	<i>PvAttrStringGet</i>	n/a

Command	<i>PvCommand</i>	<i>n/a</i>	<i>n/a</i>
---------	------------------	------------	------------

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[]).
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 µs
```

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. E.g.: <i>ExposureValue</i> , 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.

Up to 100 frames can be placed on the frame queue, with their image buffers being filled in the same order they were queued. To capture more images, re-queue new frames as the old frames complete. If using frame callback functions, this is a good place to re-queue your frames.

Most applications need not queue more than 2 or 3 frames at a time, and constantly re-queue these 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 requeued.

In this scenario, it may be better to delay processing of the images. The driver allows you to queue up to 100 frames, but you can allocate your own pool of any number of frames beyond this, 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.

ChunkModeActive / tPvFrame->AncillaryBuffer

As of camera firmware version 1.42, frames may also receive the associated chunk mode data from the camera:

- [Bytes 1 – 4] acquisition count.
- [Bytes 5 – 8] user value. Not currently implemented. 0.
- [Bytes 9 – 12] exposure value.
- [Bytes 13 – 16] gain value.
- [Bytes 17 – 18] sync in levels.
- [Bytes 19 – 20] sync out levels.
- [Bytes 21 – 24] counter 1 value. Not currently implemented. 0.
- [Bytes 25 – 28] counter 2 value. Not currently implemented. 0.
- [Bytes 29 – 32] timer 1 value. Not currently implemented. 0.
- [Bytes 33 – 36] timer 2 value. Not currently implemented. 0.
- [Bytes 37 – 40] sequence count. Not currently implemented. 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 attr 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 <i>PvCameraOpen</i> when the requested camera is not found.

<code>ePvErrAccessDenied</code>	Returned by <i>PvCameraOpen</i> when the camera cannot be opened in the requested mode, because it is already in use by another application.
<code>ePvErrUnplugged</code>	Returned when the camera has been unexpectedly unplugged.
<code>ePvErrInvalidSetup</code>	Returned when the user attempts to capture images, but the camera setup is incorrect.
<code>ePvErrResources</code>	Required system or network resources are unavailable.
<code>ePvErrQueueFull</code>	The frame queue is full.
<code>ePvErrBufferTooSmall</code>	The frame buffer is too small to store the image.
<code>ePvErrCancelled</code>	Frame is cancelled. This is returned when frames are aborted using <i>PvCaptureQueueClear</i> .
<code>ePvErrDataLost</code>	The data for this frame was lost. The contents of the image buffer are invalid.
<code>ePvErrDataMissing</code>	Some of the data in this frame was lost.
<code>ePvErrTimeout</code>	Timeout expired. This is returned only by functions with a specified timeout.
<code>ePvErrOutOfRange</code>	The attribute value is out of range.
<code>ePvErrWrongType</code>	This function cannot access the attribute, because the attribute type is different.
<code>ePvErrForbidden</code>	The attribute cannot be written at this time.
<code>ePvErrUnavailable</code>	The attribute is not available at this time.
<code>ePvErrFirewall</code>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pValue</i>	Value is returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	Value to set.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrOutOfRange</i>	The value is out of range at this time.
<i>ePvErrForbidden</i>	The attribute cannot be set at this time.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	The enumeration value (a null terminated string).

Return Value

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

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

PvAttrExists

Query: does an attribute exist?

Prototype

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

Parameters

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.

Return Value

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

<i>ePvErrSuccess</i>	The attribute exists.
<i>ePvErrNotFound</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pValue</i>	Value is returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	Value to set.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrOutOfRange</i>	The value is out of range at this time.
<i>ePvErrForbidden</i>	The attribute cannot be set at this time.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pInfo</i>	The attribute information is copied here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.

Notes

PvAttrInt64Get

Get the value of an Int64 attribute.

Prototype

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

Parameters

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pValue</i>	Value is returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	Value to set.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrOutOfRange</i>	The value is out of range at this time.
<i>ePvErrForbidden</i>	The attribute cannot be set at this time.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.

Return Value

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

<i>ePvErrSuccess</i>	The attribute is available.
<i>ePvErrUnavailable</i>	The attribute is unavailable at this time.
<i>ePvErrNotFound</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.

Return Value

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

<i>ePvErrSuccess</i>	The attribute value is in range.
<i>ePvErrOutOfRange</i>	The attribute value is out of range.
<i>ePvErrNotFound</i>	The attribute does not exist.

PvAttrList

List all camera and driver attributes.

Prototype

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

Parameters

<i>Camera</i>	Handle to open camera.
<i>pListPtr</i>	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.
<i>pLength</i>	The length of the attribute list is returned here.

Return Value

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

<i>ePvErrSuccess</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	The attribute is not an enumeration type.
<i>ePvErrBadParameter</i>	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):

```
char    enumSet[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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pMin</i>	Minimum value returned here.
<i>pMax</i>	Maximum value returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pMin</i>	Minimum value returned here.
<i>pMax</i>	Maximum value returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pMin</i>	Minimum value returned here.
<i>pMax</i>	Maximum value returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	The string value (always null terminated).

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrForbidden</i>	The attribute cannot be set at this time.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>pValue</i>	Value is returned here.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Attribute name.
<i>Value</i>	Value to set.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrOutOfRange</i>	The value is out of range at this time.
<i>ePvErrForbidden</i>	The attribute cannot be set at this time.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrBadHandle</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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. *EventAcquisitionStart* = 40000.

PvCameraEventCallbackUnregister

Unregister a callback for any camera specific events

Prototype

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

Parameters

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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.

PvCameraInfoEx

Get information on a specified camera.

Prototype

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

Parameters

<i>UniqueId</i>	Unique ID of camera.
<i>pInfo</i>	Camera information is returned here.
<i>Size</i>	Size of the tPvCameraInfoEx structure

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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* (page 42) 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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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

<i>pList</i>	Array of <i>tPvCameraInfoEx</i> , allocated by the caller. The camera list is returned in this array.
<i>ListLength</i>	Length of <i>pList</i> array.
<i>pConnectedNum</i>	The number of cameras found is returned here. This may be greater than <i>ListLength</i> . Null pointer is allowed.
<i>Size</i>	Size of the <i>tPvCameraInfoEx</i> 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* (page 38) to retrieve more information.

Example

See example for *PvCameraOpen* on page 44.

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

<i>pList</i>	Array of <i>tPvCameraInfoEx</i> , allocated by the caller. The camera list is returned in this array.
<i>ListLength</i>	Length of <i>pList</i> array.
<i>pConnectedNum</i>	The number of cameras found is returned here. This may be greater than <i>ListLength</i> . Null pointer is allowed.
<i>Size</i>	Size of the <i>tPvCameraInfoEx</i> 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*(page 38) to retrieve more information.

Example

See example for *PvCameraOpen*.

PvCameraOpen

Open a camera.

Prototype

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

Parameters

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

Return Value

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

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

Notes

Camera must be closed (see *PvCameraClose* on page 34) 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)
        {
            tPvHandle    handle;

            // 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

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

Return Value

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

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

Notes

Camera must be closed (see *PvCameraClose* on page 34) 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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	Camera was unplugged.
<i>ePvErrBadSequence</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	Camera was unplugged.

Notes

This cannot be called until the capture queue is empty. Function *PvCaptureQueueClear* (page 50) 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

<i>Camera</i>	Handle to open camera.
<i>pIsStarted</i>	Has the capture stream been started? 1=yes, 0=no.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	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* on page 51.

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 requeue frames. If using frame callbacks, check that `pFrame->Status != ePvErrCancelled` before requeuing 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

<i>Camera</i>	Handle to open camera.
<i>pFrame</i>	Frame structure which describes the frame buffer. This structure, unique to each queued frame, must persist until the frame has been captured.
<i>Callback</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	Camera was unplugged.
<i>ePvErrBadSequence</i>	You cannot queue frames until the capture stream has started.
<i>ePvErrQueueFull</i>	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:

<i>ImageBuffer</i>	Pointer to your allocated image buffer. The allocated image buffer may be larger than required.
<i>ImageBufferSize</i>	Size of your image buffer, in bytes.
<i>AncillaryBuffer</i>	Pointer to your allocated ancillary buffer, if <i>AncillaryBufferSize</i> is non-zero.
<i>AncillaryBufferSize</i>	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	<i>tPvErr</i> 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 <i>tPvImageFormat</i>).
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 <i>PvCaptureEnd</i> .
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* on page 50.

The capacity of the frame queue is 100 frames. Pushing on the queue 100 frame is in most case not necessary as the best solution is to reuse previously acquired frames to store new frames.

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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrUnplugged</i>	Camera was unplugged.
<i>ePvErrResources</i>	Required system resources were not available.
<i>ePvErrBandwidth</i>	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* (page 51).

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

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

Return Value

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

<i>ePvErrSuccess</i>	Function successful, or <i>pFrame</i> is not on the queue.
<i>ePvErrUnplugged</i>	Camera was unplugged.
<i>ePvErrTimeout</i>	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

<i>Camera</i>	Handle to open camera.
<i>Name</i>	Command (attribute) name.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	The attribute does not exist.
<i>ePvErrWrongType</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrResources</i>	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:

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrResources</i>	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

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

Return Value

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

<i>ePvErrSuccess</i>	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* (page 59) when no longer required.

PvLinkCallbackUnRegister

Un-register a link event callback.

Prototype

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

Parameters

<i>Callback</i>	Callback to run. Must be thread safe.
<i>Event</i>	Event of interest.

Return Value

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

<i>ePvErrSuccess</i>	Function successful.
<i>ePvErrNotFound</i>	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

<i>pFrame</i>	Raw Bayer image, i.e. source data.
<i>BufferRed</i>	Output buffer, pointer to the first red pixel. This buffer is allocated by the caller.
<i>BufferGreen</i>	Output buffer, pointer to the first green pixel. This buffer is allocated by the caller.
<i>BufferBlue</i>	Output buffer, pointer to the first blue pixel. This buffer is allocated by the caller.
<i>PixelPadding</i>	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: <div> <div>RGB or BGR output: 2</div> <div>RGBA or BGRA output: 3</div> <div>planar output: 0</div> </div>
<i>LinePadding</i>	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

<i>pMajor</i>	Major version number returned here.
<i>pMinor</i>	Minor version number returned here.

Notes

This function may be called at any time.

Contacting Allied Vision Technologies

- **Technical information:**

<http://www.alliedvisiontec.com>

- **Support:**

support@alliedvisiontec.com

Allied Vision Technologies GmbH (Headquarters)

Taschenweg 2a
07646 Stadtroda, Germany
Tel.: +49.36428.677-0
Fax.: +49.36428.677-28
e-mail: info@alliedvisiontec.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@alliedvisiontec.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@alliedvisiontec.com