AVT PvAPI

Programmers' Reference Manual

Version 1.22

March 10, 2010

Allied Vision Technologies Canada Inc.
101-3750 North Fraser Way
Burnaby, BC
V5J 5E9 / Canada

Table of Contents

Overview

This document is the programmer's reference for Allied Vision Technologies's 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 with NDIS (Network Driver Interface Specification**)** and kernel drivers. (see Figure 1).

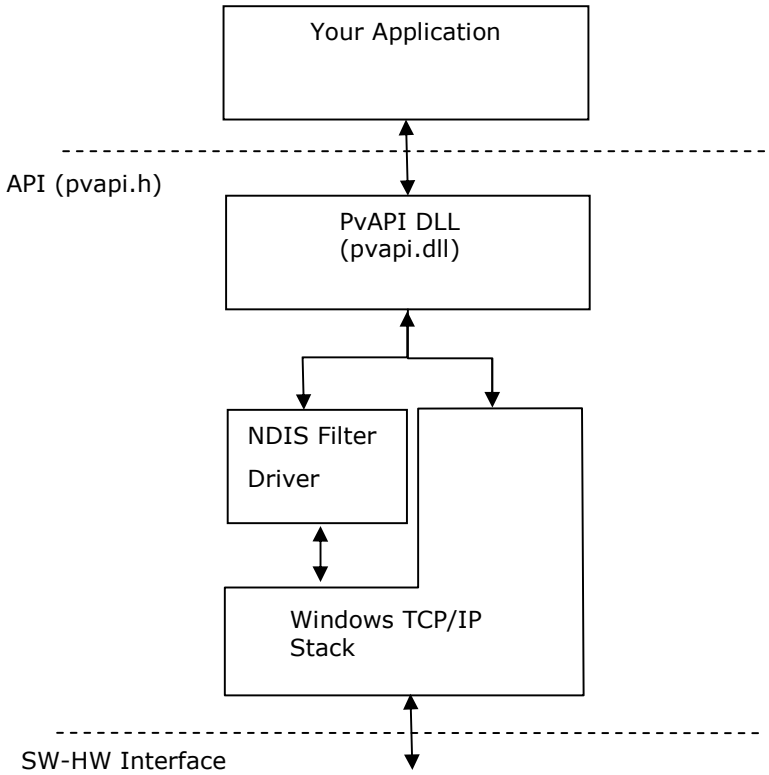

Figure 1. Allied Vision Technologies driver stack.

Using the Driver

Platform

The Allied Vision Technologies driver is supported on the following Microsoft platforms:

- Windows 2000
- Windows XP Professional or Home (32bit or 64bit)
- Windows Vista and Windows 7 (32bit or 64bit)

The following *alternative* platforms are also supported:

- 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)

The GigE Vision driver works with any Ethernet interface. If the optional GigE Filter driver is installed, the CPU load on the host will significantly be reduced (this is only available on Windows platforms). The Filter driver can be disabled from any adapter that is not used to stream from a camera.

Programming Languages (on Windows)

The user DLL ("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:

On Windows:

pvapi.dll
psligvfilter.inf
psligvfilter_m.inf
psligvfilter.sys
Allied Vision Technologies GigE Filter Installer.exe
Allied Vision Technologies Viewer Installer.exe

On other platforms:

libPvAPI.so

libPvAPI.a

libImagelib.a

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

Driver Installation

The PvAPI DLL should be installed in your application's directory. This ensures that the correct version of PvAPI is available to your application.

Here are two mechanisms for installing the GigE Filter driver (Windows only):

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:
   psligvfilter.sys - Copy to %system32%\drivers
   psligvfilter.inf - Copy to %windir%\inf
   psligvfilter_m.inf - Copy to %windir%\inf

Once installed, the GigE Filter driver will display as a service in Network adapter properties, where you can enable/disable it.

Using the API

Module Version

As new features are introduced to PvAPI, your software may not support older versions of PvAPI. In this case, 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 *PvCameraList* 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.

Setting up the camera & driver

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

For example, to start continuous acquisition, set attribute *AcquisitionMode* to *Continuous* and run the command-attribute *AcquisitionStart*:

```
PvCaptureStart(Camera);
PvAttrEnumSet(Camera, "AcquisitionMode", "Continuous");
PvCommandRun(Camera, "AcquisitionStart");
```

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);
```

Table 1 introduces the basic attributes found on all cameras. For a complete list, see the Camera Controls document. An attribute has a name, a type, and access flags such as read-permitted and write-permitted.

Table 1. List of the basic attributes, found on all cameras.

Attribute	Type	AccessFlags	Description
AcquisitionMode	Enumeration	R/W	The acquisition mode of the camera. Value set: {*Continuous,SingleFrame, MultiFrame, Recorder*}.
AcquisitionStart	Command		Start acquiring images.
AcquisitionStop	Command		Stop acquiring images.
AcquisitionAbort	Command		Stop acquiring images (abort any on-going exposure)
PixelFormat	Enumeration	R/W	The image format. Value set: {*Mono8, Mono16, Bayer8, Bayer16, Rgb24, Rgb48, Yuv411, Yuv422, Yuv444*}.
Width	Uint32	R/W	Image width, in pixels.
Height	Uint32	R/W	Image height, in pixels.
TotalBytesPerFrame	Uint32	R	Number of bytes per image.

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 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.
String	A string (null terminated, char[]).
Command	Valueless; a function executes when the attribute is written.

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. An example of a volatile attribute is *ExposureValue*, because the exposure is always changing if the camera is in auto-expose mode.
Constant	The attribute value will never change.

Table 2 lists the PvAPI functions used to access attributes.

Table 2. Functions for reading and writing attributes.

Attribute Type	Set	Get	Range
Enumeration	*PvAttrEnumSet*	*PvAttrEnumGet*	*PvAttrRangeEnum*
Uint32	*PvAttrUint32Set*	*PvAttrUint32Get*	*PvAttrRangeUint32*
Float32	*PvAttrFloat32Set*	*PvAttrFloat32Get*	*PvAttrRangeFloat32*
String	*PvAttrStringSet*	*PvAttrStringGet*	n/a
Command	*PvCommand*	n/a	n/a

Image Acquisition and Capture

To obtain an image from your camera, first setup PvAPI to capture images, then start acquisition on the camera. These two concepts – capture and acquisition – while related, are independent operations as it is shown below:

To capture images sent by the camera, follow these steps:

1. *PvCaptureStart* – initialize the image capture stream.
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.
3. When done, *PvCaptureEnd* – close the image capture stream.

None of the steps above cause the camera to acquire an image. To effect image acquisition on the camera, follow these steps:

1. Set attribute *AcquisitionMode*.
2. Run command attribute *AcquisitionStart*.
3. When done, depending on the application, run command attribute *AcquisitionStop*.

Normally, image capture is initialized and frame buffers are queued before the command *AcquisitionStart* is run, but the order can vary depending on the application. To guarantee a particular image is captured, you must ensure that your frame buffer is queued before the camera is instructed to start acquisition.

Image Capture

Images are captured using the asynchronous function *PvCaptureQueueFrame*. Allocate an image buffer (use attribute *TotalBytesPerFrame* or calculate the size yourself), fill out a *tPvFrame* structure, and place the frame structure on the queue with *PvCaptureQueueFrame*.

Before a queued image buffer can be used or the frame structure modified, the application needs to know when the image capture is complete. Two mechanisms are

available: either block your thread until capture is complete using *PvCaptureWaitForFrameDone*, or specify a callback function when you run *PvCaptureQueueFrame*. Your callback function is run by the driver when image capture is complete.

NOTE: Always check that tPvFrame->Status equals ePvErrSuccess, when a frame returned to you to ensure the data is valid. For example: lost data over the GigE network (usually the result of an improperly configured camera or network card, e.g. mismatch of packet size) will result in ePvErrDataMissing, meaning the complete frame has not been received by the host.

Many frames can be placed on the frame queue, and their image buffers will be filled in the same order they were queued. Up to 100 frames may be queued at one time. To capture more images, keep submitting new frames as the old frames complete. Most applications need not queue more than 2 or 3 frames at a time.

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.

Image Acquisition

Image acquisition is setup via attributes *AcquisitionMode*, *AcquisitionStart*, and *AcquisitionStop*. See the Camera Controls document for more information.

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 cancelled. 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

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.

Notes

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 the attributes applicable to a camera.

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):

```
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

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.

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

```
void 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

The number of cameras is dynamic; it may change at any time.

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.

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.

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* (page 36) 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* (page 32) to retrieve more information.

Example

See example for *PvCameraOpen* on page 38.

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 *PvCameraInfoByAddr* Ex(page 32) to retrieve more information.

Example

See example for *PvCameraOpen* on page 38.

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 *PvCameraList*.
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 *PvCameraList* and *PvCameraOpen*.

Notes

Camera must be closed (see *PvCameraClose* on page 28) 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

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 *PvCameraListUnreachable* and *PvCameraOpenByAddr*.

Notes

Camera must be closed (see *PvCameraClose* on page 28) 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* (page 44) 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 the frame queue. Incomplete frames are returned with 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* on page 45.

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

Note that if another frame is being queued at the same time as *PvCaptureQueueClear*, the results are indeterminate. If using frame callbacks, be sure to stop re-queuing frames before your call to *PvCaptureQueueClear*.

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* on page 44.

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:

ePvErrSuccess	Function successful.
ePvErrUnplugged	Camera was unplugged.
ePvErrResources	Required system resources were not available.
ePvErrBandwidth	Insufficient Firewire 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 45).

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 PvCameraList call.

Example

```
tPvCameraInfoEx list;

if(!PvInitialize())
{
      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* (page 53) 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.

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