PROSILICA

Prosilica PvAPI

Programmers' Reference Manual

Version 1.18
October, 2008

Table of Contents

Overview

This document is the programmers' reference for Prosilica's GigE Vision driver and its Application Programming Interface.

The Prosilica PvApi interface supports all GigE Vision cameras from Prosilica.

The PvApi driver interface is a user DLL which communicates with NDIS (Network Driver Interface Specification) and kernel drivers. (see Figure 1).
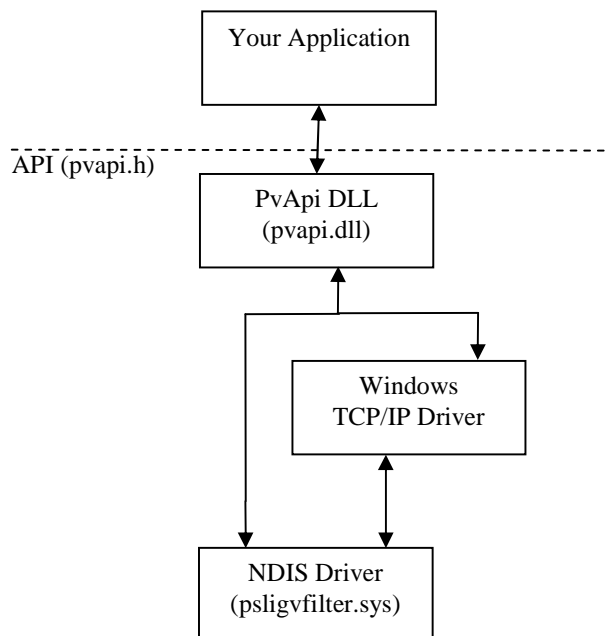
Figure 1. Prosilica driver stack.

Using the Driver

Platform

The Prosilica driver is supported on the following Microsoft platforms:

- Windows 2000

- Windows XP Professional or Home (32bit or 64bit)

- Windows Vista (32bit)

The following *alternative* platforms are also supported:

- Linux (x86, PPC, x64)

- QNX 6.3 (x86), 6.3 + Core Networking 6.4, 6.4 Beta

- Mac OS X (x86, PPC, 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)

Programming Languages

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 cameras only:
        pvapi.dll
        psligvfilter.inf
        psligvfilter_m.inf
        psligvfilter.sys
        Prosilica GigE Filter Installer.exe
        Prosilica Viewer Installer.exe

No other files from the SDK may be redistributed without written permission from Prosilica Inc.

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:

1. Run "Prosilica 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 Prosilica cameras connected to the system

Example:

```
tPvCameraInfo      list[10];
unsigned long      numCameras;

numCameras = PvCameraList(list, 10, NULL);

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

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

UniqueId	A value unique to each camera shipped by Prosilica.
SerialString	The full part & serial number of the camera, for example "02-1000A-10580".
PartNumber PartVersion	Together, the part number and part version identify the type of camera.
PermittedAccess	Type of access allowed: master (full control) or monitor (read only).
InterfaceId	An ID value for each interface or bus. The interface ID may change each time PvApi is initialized.
InterfaceType	The interface type, i.e. Firewire or Ethernet.
DisplayName	People-friendly camera name, for example "GE1380".

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:

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

numCameras = PvCameraList(info, 1, NULL);

// Open the first camera found, if it's not already open.  (See
// function reference for PvCameraOpen for a more complex example.)
if ((numCameras == 1) && (item.PermittedAccess & ePvAccessMaster))
      PvCameraOpen(item.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 Attribute Reference on page 51. 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* list 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 bellow:

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 *AcquisitionEnd*.

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 the image buffer can be used or the frame structure modified or deleted, the application needs to know when the image capture is complete. Two mechanisms are available: either block your thread until capture is complete (*PvCaptureWaitForFrameDone*), or specify a callback function when you run *PvCaptureQueueFrame*. Your callback function is run, by the driver, when image capture is complete.

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 Attribute Reference 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; the missing data was zero-filled.
`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

The attribute information is static for this camera; it won't change while the camera is open.

PvAttrIsAvailable

Query: is the attribute available at this time?

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?

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 Prosilica 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.

PvCameraInfo

Get information on a specified camera.

Prototype

```
tPvErr PvCameraInfo
(
    unsigned long       UniqueId,
    tPvCameraInfo*      pInfo
);
```

Parameters

UniqueId   Unique ID of camera.

pInfo   Camera information is returned here.

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 Prosilica's driver.

See *PvCameraList* (page 32) if you want to retrieve information for all cameras.

PvCameraInfoByAddr

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 PvCameraInfoByAddr
(
    unsigned long       IpAddr,
    tPvCameraInfo*      pInfo,
    tPvIpSettings*      pIpSettings
);
```

Parameters

IpAddr   IP address of camera, in network byte order.

pInfo   Camera information is returned here.

pIpSettings   Camera IP settings is returned here.

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 *PvCameraList* or *PvCameraListUnreachable*.

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.

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.

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.

PvCameraList

List the Prosilica cameras currently visible to this system.

Prototype

```
unsigned long PvCameraList
(
    tPvCameraInfo*      pList,
    unsigned long       ListLength,
    unsigned long*      pConnectedNum
);
```

Parameters

pList   Array of *tPvCameraInfo*, 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.

Return Value

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

Notes

Lists only the cameras which are turned on and using Prosilica's drivers.

If you expect a particular camera to be present, alternatively you can use *PvCameraInfo* (page 28) to retrieve more information.

Example

See example for *PvCameraOpen* on page 34.

PvCameraListUnreachable

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 PvCameraListUnreachable
(
    tPvCameraInfo*      pList,
    unsigned long       ListLength,
    unsigned long*      pConnectedNum
);
```

Parameters

pList   Array of *tPvCameraInfo*, 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.

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* (page 28) to retrieve more information.

Example

See example for *PvCameraOpen* on page 34.

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 26) when no longer required.

Example

```
tPvHandle OpenFirstCamera(void)
{
        tPvCameraInfo  list[10];
        unsigned long  numCameras;

        // List available cameras.
        numCameras = PvCameraList(list, 10, NULL);

        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 *PvCameraLisUnreachablet* and *PvCameraOpenByAddr*.

Notes

Camera must be closed (see *PvCameraClose* on page 26) 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.

PvCaptureEnd

Shut down the image capture stream.

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 39) 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 40).

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
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 39.

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 frame 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 40).

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 (attribute). 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

```
tPvCameraInfo list;

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

PvCameraList(&list,1,NULL);
…
}
else
        printf("failed to initialize the API\n");
```

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 47) 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.

Attribute Reference

Attributes

Important notes about attributes:

1) Not all attributes are available on all cameras (except where noted). In other words, don't assume an attribute is available.

2) For a particular enumeration attribute, the set may not contain all of the values documented below.

3) The value of some attributes impacts the availability or range of other attributes. For example, *BinningX* impacts the range of *Width.*

Note: many attributes in PvApi are equivalent to GenICam features, but the PvApi attribute system is not GenICam. Prosilica GigE Vision cameras are GenICam compliant, and we recommend you use a GenICam driver if you plan to support cameras from other manufacturers.

Image Mode

Image Mode attributes should be set up before Image Format attributes, since the region size and pixel formats may depend on these mode attributes.

Attribute	Type	Flags	Description
BinningX	Uint32	R/W	Horizontal binning. 1=no binning.
BinningY	Uint32	R/W	Vertical binning. 1=no binning.

Image Format

Image Format attributes control the data content of acquired images.

Attribute	Type	Flags	Description
Width	Uint32	R/W	Image width, in pixels.
Height	Uint32	R/W	Image height, in pixels
RegionX	Uint32	R/W	Start of region readout, in pixels; left edge.
RegionY	Uint32	R/W	Start of region readout, in pixels; top edge.
PixelFormat	Enumeration	R/W	The image format. Value set: {*Mono8, Mono16, Bayer8, Bayer16, Rgb24, Rgb48, Yuv411, Yuv422, Yuv444, Bgr24. Rgba32, Bgra32*}.
TotalBytesPerFrame	Uint32	R/W	Number of bytes per image.
MirrorX	Enumeration	R/W	Mirror the image in Width Value set: {On, Off}.

Acquisition Control

The Acquisition Control attributes control image acquisition and the trigger source.

Attribute	Type	Flags	Description
AcquisitionMode	Enumeration	R/W	The acquisition mode of the camera. Value set: *Continuous* — Continuous acquisition. *SingleFrame* — Take one image, then stop acquisition. *MultiFrame* — Take N images, then stop acquisition. N is set via *AcquisitionFrameCount.* *Recorder* Record N images, then stop acquisition. N is set via AcquisitionFrameCount.
AcquisitionStart	Command		Start acquiring images. Image acquisition will continue until *AcquisitionStop* is run, or AcquisitionFrameCount is reached.
AcquisitionStop	Command		Stop acquiring images.
AcquisitionAbort	Command		Abort the ongoing acquisition. The camera will come out of the acquisition mode as soon as possible, even when a very large exposure time is set.
AcquisitionFrameCount	Uint32	R/W	Frame count when the acquisition mode is *MultiFrame* or *Recorder*. When in the later mode, the value should not be larger than the value of the *StreamHoldCapacity* attribute.
RecorderPreEventCount	Uint32	R/W	Number of frames to record, pre-event. The number of post-event frames to be recorded will be *AcquisitionFrameCount – RecorderPreEventCount.*
FrameStartTriggerMode	Enumeration	R/W	The acquisition trigger. Value set: *Freerun* — Continuous trigger. *SyncIn1* — External trigger input. *SyncIn2* — External trigger input. *SyncIn3* — External trigger input. *SyncIn4* — External trigger input. *FixedRate* — Fixed frame-rate generator. *Software* — Acquire when FrameStartTrigger*Softwarr* command is run.
FrameStartTriggerEvent	Enumeration	R/W	External trigger event. Value set: *EdgeRising* *EdgeFalling* *EdgeAny* *LevelHigh* *LevelLow*
FrameStartTriggerDelay	Uint32	R/W	External trigger delay, in microseconds.
FrameRate	Float32	R/W	Fixed rate generator; frames per second.
FrameStartTriggerSoftware	Command		Software-controlled acquisition trigger.
AcqEndTriggerEvent	Enumeration	R/W	Acquisition end external trigger event. Value set: *EdgeRising* *EdgeFalling* *EdgeAny* *LevelHigh* *LevelLow*
AcqEndTriggerMode	Enumeration	R/W	Acquisition end external trigger mode. Value set:

			SyncIn1 External trigger input.
			SyncIn2 External trigger input.
			SyncIn3 External trigger input.
			SyncIn4 External trigger input.
			Disabled Disabled
AcqRecTriggerEvent	Enumeration	R/W	Recorder external trigger event. Value set:
			EdgeRising
			EdgeFalling
			EdgeAny
			LevelHigh
			LevelLow
AcqRecTriggerMode	Enumeration	R/W	Recorder external trigger mode. Value set:
			SyncIn1 External trigger input.
			SyncIn2 External trigger input.
			SyncIn3 External trigger input.
			SyncIn4 External trigger input.
			Disabled Disabled
AcqStartTriggerEvent	Enumeration	R/W	Acquisition start trigger event. Value set:
			EdgeRising
			EdgeFalling
			EdgeAny
			LevelHigh
			LevelLow
AcqStartTriggerMode	Enumeration	R/W	Acquisition start trigger mode. Value set:
			SyncIn1 External trigger input.
			SyncIn2 External trigger input.
			SyncIn3 External trigger input.
			SyncIn4 External trigger input.
			Disabled Disabled

Feature Control

Attribute	Type	Flags	Description
ExposureMode	Enumeration	R/W	Exposure mode. Value set:
			Manual Exposure is controlled by *ExposureValue*.
			Auto Continuous auto-exposure.
			AutoOnce Auto-exposure until complete, then revert to *Manual* mode.
ExposureValue	Uint32	R/W/V	Exposure time, in microseconds.
ExposureAutoAdjustDelay	Uint32	R/W	Number of frames to wait before making adjustments to the auto exposure setting. If changes to scene lighting return to initial levels within the delay setting then the camera will not change the settings.
ExposureAutoAdjustTol	Uint32	R/W	Range of scene lighting changes in which the auto exposure will not respond. This parameter can be used to limit exposure setting changes to only larger variations in scene lighting.

ExposureAutoAlg	Enumeration	R/W	auto exposure method to be used. Value set: {*Mean, FitRange*}.
			Using *Mean* will center the exposure range on the target setting (see *ExposureAutoTarget*) allowing some bright areas to saturate. *Fitrange* will fit all exposures so that the bright areas are not saturated. Generally, the *Mean* setting is preferred.
ExposureAutoMax	Uint32	R/W	Upper bound to the exposure setting in auto exposure mode. This parameter is very useful in situations where framerate is important and when the camera is run in *FreeRunning* mode. This value would normally be set to something less than 1x10^6/(desired frame rate). The units are microseconds.
ExposureAutoMin	Uint32	R/W	Lower bound to the exposure setting in auto exposure mode. Normally, this number would be set to the minimum exposure time that the camera is capable of. The units are microseconds.
ExposureAutoOutliers	Uint32	R/W	Percentage of the image pixels that do not have to fit into the proper exposure range. This parameter can be used to manipulate how the auto exposure function works.
ExposureAutoRate	Uint32	R/W	Rate at which the auto exposure function changes the exposure setting.
ExposureAutoTarget	Uint32	R/W	Controls the general lightness or darkness of the auto exposure feature. It determines the target setting for the Mean setting of *ExposureAutoAlg*. The units are percentage.
GainMode	Enumeration	R/W	Gain mode. Value set: {*Manual*}.
GainValue	Uint32	R/W	Gain value, unitless.
GainAutoAdjustDelay	Uint32	R/W	The auto-gain adjustment delay, in milliseconds. When the gain value is out of tolerance, the camera will wait for this amount of time before making an adjustment. If the gain value comes back into tolerance before this period expires, the adjustment is cancelled. Range is 0 to 10000.
GainAutoAdjustTol	Uint32	R/W	The auto-gain tolerance. Once the current gain value is beyond this tolerance from the calculated value, an auto-gain adjustment will occur. Tolerance is in percent, from 0 to 50.
GainAutoMax	Uint32	R/W	Maximum gain value allowed to be set by the auto-gain function.
GainAutoMin	Uint32	R/W	Minimum gain value allowed to be set by the auto-gain function.
GainAutoOutliers	Uint32	R/W	Number of top outliers to discard when calculating auto-gain. Value is in percent of the number of pixels in an image, from 0 to 50.
GainAutoRate	Uint32	R/W	Auto-gain adjustment rate, in percent. Range is 1 (slowest) to 100 (fastest).
GainAutoTarget	Uint32	R/W	In an automatic-gain mode, this sets the desired mean value of the image data, in percent. In other words, this allows the auto-gain level to be brightened or darkened. Value is from 0 to 100.
WhitebalAutoRate	Uint32	R/W	How fast the Auto white balance will update. This can be used to slow the rate of color balance change so that only

			longer period fluctuations affect color.
OffsetMode	Enumeration	R/W	Offset mode. Value set: *Manual* Offset controlled by *OffsetValue*
OffsetValue	Uint32	R/W/V	Offset value, unitless.
DSPSubregionLeft	Uint32	R/W	The DSP subregion for auto-exposure and auto-whitebalance algorithms. This DSP subregion is relative to the image region. To use the full image region, set the left and top to 0, and the right and bottom to 0xFFFFFFFF. The default DSP subregion is the full image region.
DSPSubregionTop	Uint32	R/W	
DSPSubregionRight	Uint32	R/W	
DSPSubregionBottom	Uint32	R/W	
DSPContinuousMode	Enumeration	R/W	When *On*, the camera will run in the background while waiting for a trigger, in order to perform adjustments (e.g auto-iris, auto exposure …) in a timely manner. (Attribute will be available only if supported by the camera)
DSPContinousLatency	Uint32	R/W	Frame trigger latency in microseconds when *DSPContinuousMode* is enabled. (Attribute will be available only if supported by the camera)

IrisMode	Enumeration	R/W	Auto-iris mode. Value set:
			
			    Disabled Off
			
			    Video The camera output a video-iris signal
			
			    VideoOpen Fully open the iris
			
			    VideoClosed Fully close the iris

IrisAutoTarget	Uint32	R/W	Desired mean value of the image data when in automatic mode. The units are percentage.
IrisVideoLevelMin	Uint32	R/W	Minimum video-iris level output by the camera in mV pp (range from 0 to 150).
IrisVideoLevelMax	Uint32	R/W	Maximum video-iris level output by the camera in mV pp (range from 0 to 150).
IrisVideoLevel	Uint32	R/V	Current video-iris level. In mV pp.
DefectMaskColumnEnable	Enumeration	R/W	When *On*, the camera will mask any factory known column defect.
DefectMaskPixelEnable	Enumeration	R/W	When On, the camera will mask any factory known pixel defect.

IO Control

Attribute	Type	Flags	Description
SyncInLevels	Uint32	R	input levels. Bit 0 is sync-in 0, Bit 1 is sync-in 1, etc.
SyncOutGpoLevels	Uint32	R/W	GPO output levels. Bit 0 is sync-out 0, bit 1 is sync-out 1, etc.
SyncOut1Mode	Enumeration	R/W	Function of sync-out 1. Value set: {*GPO, AcquisitionTriggerReady, FrameTriggerReady, FrameTrigger, Exposing, FrameReadout, Imaging,*

			Acquiring, SyncIn1, SyncIn2, SyncIn3, SyncIn4, Strobe1, *Strobe2, Strobe3, Strobe4}.*
SyncOut1Invert	Enumeration	R/W	Invert sync-out 1 line: *On* or *Off*.
SyncOut2Mode	Enumeration	R/W	See *SyncOut1Mode.*
SyncOut2Invert	Enumeration	R/W	See *SyncOut1Invert.*
SyncOut3Mode	Enumeration	R/W	See *SyncOut1Mode.*
SyncOut3Invert	Enumeration	R/W	See *SyncOut1Invert*
Strobe1Mode	Enumeration	R/W	Input signal into strobe 1. Value set: {*AcquisitionTriggerReady, FrameTriggerReady,* *FrameTrigger, Exposing, FrameReadout, Acquiring,* *SyncIn1, SyncIn2, SyncIn3, SyncIn4}.*
Strobe1Delay	Uint32	R/W	Strobe delay in microseconds, from strobe input to strobe output.
Strobe1ControlledDuration	Enumeration	R/W	When *On*, strobe duration is controlled. When *Off*, the strobe duration matches the input signal.
Strobe1Duration	Uint32	R/W	Duration in microseconds, when *StrobeXControlledDuration* is *On*.

GigE Vision

Attribute	Type	Flags	Description
PacketSize	Uint32	R/W	Size of image data packet. This size includes the GVSP, UDP, and IP headers.
StreamBytesPerSecond	Uint32	R/W	Bandwidth of image data, in bytes per second.
GvcpRetries	Uint32	R/W	Number of retries per GVCP command, before giving up.
HeartbeatTimeout	Uint32	R/W	GVCP heartbeat timeout, in milliseconds.
HeartbeatInterval	Uint32	R/W	Interval, in milliseconds, at which the API must send a heartbeat command to the camera. The value must be smaller than the *HeartbeatTimeout.*
StreamHoldEnable	Enumeration	R/W	Image stream hold: *On* to pause the image stream, *Off* for normal operation. For example, *StreamHold* could be turned *On* and then a number of frames could be captured into memory; when stream hold is turned *Off* again, those captured images are transmitted to the host.
StreamHoldCapacity	Uint32	R/V	Number of frame that can be captured in memory with the current frame settings.
DeviceEthAddress	String	R/C	MAC address of the camera
DeviceIPAddress	String	R/C	IP address of the camera
HostEthAddress	String	R/C	MAC address of the host (of the adapter on which the camera was detected)
HostIPAddress	String	R/C	IP address of the host (of the adapter on which the camera was detected)

Attribute	Type	Flags	Description
MulticastEnable	Enumeration	R/W	*On* to instructs the camera to multicast its stream instead of unicasting it. The value of the attribute should be changed before the stream is started by the *master* application. If the application is a *monitor*, it doesn't need to change this attribute. The API will detect that the camera is multicasting and handle such case automatically.
MulticastIPAddress	String	R/W	IP address to be used by the camera for the multicasting. A default value is provided. If you need to change it, make sure it is in the range of supported multicast addresses.
BandwidthCtrlMode	Enumeration	R/W	Allows to select the desired mode of bandwidth control. Value set : {StreamBytesPerSecond, SCPD, Both}.

Information

Attribute	Type	Flags	Description
CameraName	String	R/W	Human readable camera name, such as "EngineRoomCam1".
ModelName	String	R/W	Human readable model name, such as "GE650". Software should use the *PartNumber* and *PartVersion* to distinguish between models.
UniqueId	Uint32	R/C	An identifier unique to each Prosilica camera, regardless of model.
PartNumber	Uint32	R/C	The elements of a Prosilica serial number. For example, a camera labeled "02-2010A-04000" has a *PartNumber* 2010, *PartVersion* A, and *SerialNumber* 4000.
PartVersion	Uint32	R/C	
SerialNumber	String	R/C	The *SerialNumber* is not a unique identifier across models; software should use *UniqueId* instead.
PartRevision	String	R/C	Revision code. Generally unimportant, as functionality does not change between revisions.
FirmwareVerMajor	Uint32	R/C	Camera firmware version, major.
FirmwareVerMinor	Uint32	R/C	Camera firmware version, minor.
FirmwareVerBuild	Uint32	R/C	Camera firmware build. Generally unimportant.
SensorType	Enumeration	R/C	Sensor type. Values are "Mono" and "Bayer".
SensorBits	Uint32	R/C	Maximum bit depth of sensor ADC.
SensorWidth	Uint32	R/C	Maximum width of sensor.
SensorHeight	Uint32	R/C	Maximum height of sensor.
TimeStampFrequency	Uint32	R/C	Timestamp frequency, in Hz.

Non-Volatile Configuration Files

Attribute	Type	Flags	Description
ConfigFileLoad	Command		Load the camera configuration from the non-volatile memory file selected by *ConfigFileIndex*.
ConfigFileSave	Command		Save the current camera configuration to the non-volatile memory file selected by *ConfigFileIndex*.
ConfigFileIndex	Enumeration	R/W	Memory file to be used for loading or saving the camera configuration. "Factory" is the factory default settings file; this file cannot be overwritten.
ConfigFilePowerUp	Enumeration	R/W	Memory file loaded on camera power-up or reset.

Statistics

Attribute	Type	Flags	Description
StatDriverType	Enumeration	R/V	Type of the streaming driver in use. Value set = {*Standard*, *Filter*, *Performance*}
StatFilterVersion	String	R/C	Version of the filter driver installed on the host (Windows only)
StatFrameRate	Float32	R/V	Current frame rate of the camera
StatFramesCompleted	Uint32	R/V	Numbers of frames successfully acquired
StatFramesDropped	Uint32	R/V	Numbers of frames unsuccessfully acquired
StatPacketsErroneous	Uint32	R/V	Numbers of erroneous packet received
StatPacketsMissed	Uint32	R/V	Numbers of packets sent by the camera but not received by the host
StatPacketsReceived	Uint32	R/V	Number of packets sent by the camera and received by the host
StatPacketsRequested	Uint32	R/V	Number of missing packets requested to the camera for resent
StatPacketsResent	Uint32	R/V	Number of missing packets resent by the camera and received by the host