PVCAM 3.0

57-062-001
Version 3.0
September 13

Photometrics

3440 E. Britannia Dr.

Tucson, AZ 85706

TEL: 520-889-9933

FAX: 520-573-1944

Table of Contents

List of Tables

This page intentionally left blank.

Chapter 1: SDK

What is the SDK?

SDK — Photometrics' Software Development Kit — allows programmers to access and use the capabilities of PVCAM© — Programmable Virtual Camera Access Method Library. (PVCAM is described in detail in the chapters that follow.)

For developer convience, we have included a Windows environment variable into the PVCAM installer, which will guide developers to the location of all binaries and header files needed to develop with PVCAM as well as Visual Studio and similar environments. This environment variable is called PVCAM_SDK_PATH and can be discovered in the Windows explorer using %PVCAM_SDK_PATH% syntax and Visual Studio using $(PVCAM_SDK_PATH) syntax; additionally, if the developer desires to build with a special version of PVCAM different from the version installed the "system" environment variable may be overwritten using a "user" environment variable with the same name. All the developer must do to use the custom location is build a similar directory structure and create a "user" environment variable with the path of the folder (i.e. – "C:\alt_pvcam_sdk"). Please consult this Read Me file for information on:

- Linking PVCAM to your software
- Initializing PVCAM
- Basic Acquisition using PVCAM

Contact Information

Photometrics' headquarters are located at the following addresses:

Photometrics

3440 East Britannia Drive

Tucson, Arizona 85706 (USA)

TEL: 800-874-9789 / 520-889-9933

FAX: 520-573-1944

Tech Support E-mail: support@photometrics.com

For technical support and service outside the United States, see our web page at www.photometrics.com. An up-to-date list of addresses, telephone numbers, and e-mail addresses of Photometrics' overseas offices and representatives is maintained on the web page.

This page intentionally left blank.

Chapter 2:
PVCAM, A High-Level C Library

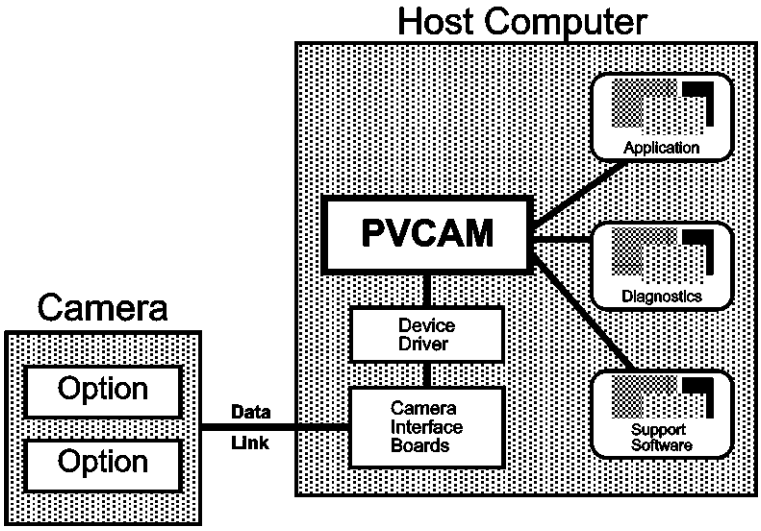
Introduction

PVCAM is an ANSI C library of camera control and data acquisition functions. This library, which is identical across platforms and operating systems, provides an interface that allows developers to specify the camera's setup, exposure, and data storage attributes.

Note: Many Photometrics cameras support ICL scripting language that provides detailed low-level control of exposure and CCD readout.

System Overview

To use PVCAM, a system must include camera hardware and software, a host computer, and the PVCAM library.



Hardware Support

PVCAM library supports all Photometrics brand hardware.

Library Classes

The basic PVCAM library supports the following five classes of camera and buffer control:

0. Camera Communications	These functions establish communication paths between the high-level application software and the device driver. They also establish some low-level functions for controlling the camera hardware.
1. Error Reporting	These functions monitor and report on other library functions. When an error occurs, a function can be called to return a unique error code.
2. Configuration/Setup	These functions initialize the library and set up the hardware and software environments. They also control and monitor the camera hardware, and allow the user to set parameters such as camera gain and temperature.
3. Data Acquisition	These functions define how the image data are collected.

Documentation Style

This manual describes the functional aspects of using PVCAM and various controls for Photometrics® cameras (Chapter 2), gives reference pages for all of the function calls (Chapter 3 through Chapter 6:
Data Acquisition (Class 3)) and provides code examples.

Defined Types

In order to work effectively across platforms, the number of bytes in a variable must be consistent. Therefore, new types have been defined for PVCAM. These typedefs are given in the header file `master.h`.

Type	Explanation
rs_bool*	true (non-0) or false (0) value
int8	signed 8-bit integral value
uns8	unsigned 8-bit integral value
int16	signed 16-bit integral value
uns16	unsigned 16-bit integral value
int32	signed 32-bit integral value
uns32	unsigned 32-bit integral value
enum	treat as unsigned 32-bit integral value
flt64	64-bit floating point value
ulong64	unsigned 64-bit integral value
long64	signed 64-bit integral value
smart_stream_type	Structure for S.M.A.R.T streaming
FRAME_INFO	Structure holding additional frame information, see pvcam.h

*__Note:__ The type ‘rs_bool’ has replaced the deprecated ‘boolean’ type. This is due to a size difference of the ‘boolean’ type on the Windows platform. Namely, <windows.h> defines a ‘boolean’ type of a different size. Including <windows.h> in the same translation unit as “master.h” compiles the wrong ‘boolean’ and causes subtle memory access violations. It is strongly recommended to use the new ‘rs_bool’ type instead to avoid this potential clash.

Since Photometrics® camera data and analyses depend on bit depth, the new types give values that are consistent with the size of the bit depth.

Each new type is composed of the appropriate combinations of int, short, long, or other types that give the appropriate length for each value. The 8-bit types are the smallest type that holds 8 bits, 16-bit types are the smallest type holding 16 bits, and so forth.

The following list includes the new types defined for use in PVCAM. Additional derived types always begin with the base name followed by `_ptr` or `_const_ptr`.

Type	Pointer	Pointer to Constant Type
rs_bool	rs_bool_ptr	rs_bool_const_ptr
char	char_ptr	char_const_ptr
int8	int8_ptr	int8_const_ptr
uns8	uns8_ptr	uns8_const_ptr

Type	Pointer	Pointer to Constant Type
int16	int16_ptr	int16_const_ptr
uns16	uns16_ptr	uns16_const_ptr
int32	int32_ptr	int32_const_ptr
uns32	uns32_ptr	uns32_const_ptr
flt64	flt64_ptr	flt64_const_ptr
ulong64	ulong64_ptr	
long64	long64_ptr	
rgn_type	rgn_ptr	rgn_const_ptr
smart_stream_type	smart_stream_type_ptr	
FRAME_INFO	PFRAME_INFO	

Naming Conventions

To shorten names and improve readability, standard abbreviations are used for common words and phrases. These abbreviations are used in function and variable names.

adc=analog-to-digital converter	dly=delay	ofs=offset
addr=address	dup=duplicate	par=parallel
bin=binning	err = error	pix=pixel
buf=buffer	exp=exposure	ptr=pointer
cam=camera	expt=export	rgn=region
cfg=configuration	hcam=camera handle	ser=serial
chan=channel	hi=high	shtr=shutter
clr=clear	init=initialize	spd=speed
cmd=command	len=length	tmp=temp
comm=communication	lo=low	totl=total
ctr=counter	mem=memory	xfr=transfer
ctrl=control	num=number	

In PVCAM, *num* always means **current selection number**, while *totl* or *entries* is used for **total different possibilities**.

A leading *h* usually signifies a type of handle, such as the camera handle (*hcam*). A handle is a 16-bit number that is used to uniquely identify an object.

Include Files

Any program using PVCAM must include the following files:

- master.h system-specific definitions and types

- `pvcam.h` constants and prototypes for all functions

`master.h` must be included before `pvcam.h`.

Parameter Passing and const

When parameters are passed in or out of functions, it may be difficult to determine which parameters the user should set and which parameters are set by the function. This is particularly difficult in PVCAM, because virtually all information is exchanged through parameters (the function return value is reserved for indicating errors).

A few simple rules help resolve the confusion:

- Pointers generally return information **from** a function.
- Non-pointers always send information **to** a function.

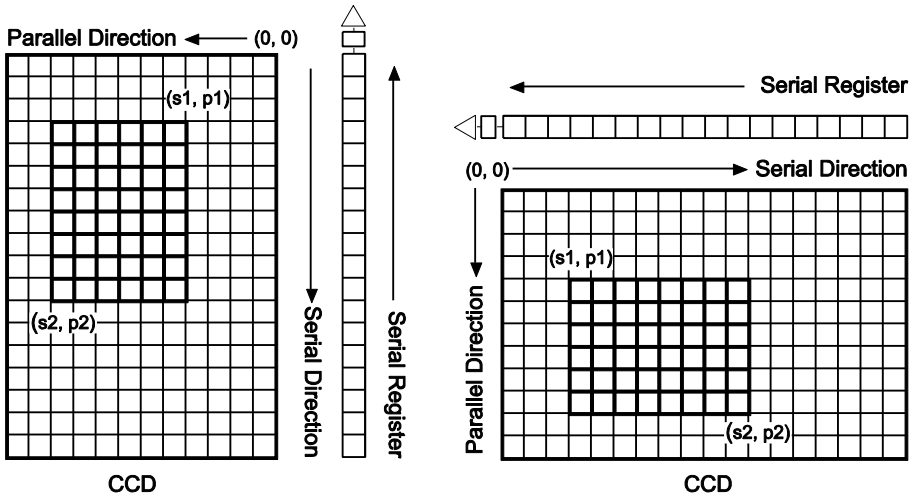
In a few cases, such as structures and arrays, a pointer is passed even though the data are being sent in to the function. This is done to reduce overhead and to speed function calls, but it conflicts with the rules above. To solve this problem, when a structure or array (pointer) is sent as input to a function, the *_const_ptr* type is used to indicate that the function will not (and can not) change the data.

Note: *const_ptr* (pointers to const) always sends data ***into*** a function. The data is not altered.

CCD Coordinates Model

In many cameras, the CCD orientation is fixed. This fixed position places the origin in a predetermined location and gives each pixel an x,y location.

In Photometrics cameras, the CCD orientation is not only different from camera to camera, but the orientation may also change when the application changes. Therefore, we use a **serial, parallel** (s,p) coordinates system. In this system, the origin is located in the corner closest to the serial register readout, and the coordinates increase as the locations move away from the origin. The diagram below illustrates how the coordinates are unaffected by the CCD orientation.



Regions and Images

A region is a user-defined, rectangular exposure area on the CCD. As seen in the diagram above, the user defines the region by selecting *s1,p1* and *s2,p2*,the diagonal corners of the region.

An image is the data collected from a region. PVCAM reads out the image, then stores it in a buffer.

Binning Factors

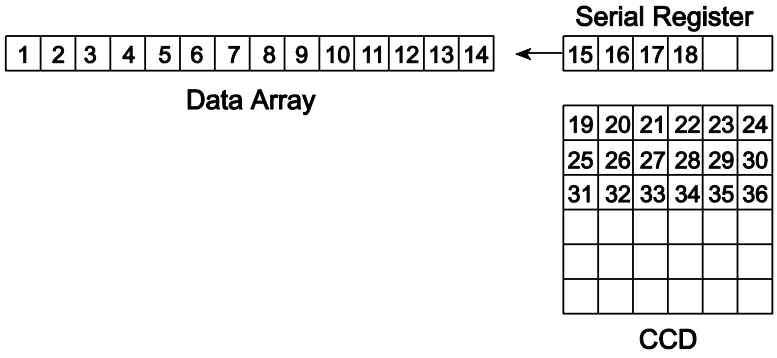
For data collection, two other parameters are needed: the serial and parallel binning factors. A binning of 1 in both directions reads out each pixel at full lateral resolution. A binning of 2 in both directions combines four pixels, cutting the lateral resolution in half, but quadrupling the light-collecting area. The number of pixels read out are determined as *(s2-s1+1)/sbin* in the serial direction, and *(p2-p1+1)/pbin* in the parallel direction. If these equations do not produce an integer result, the remaining pixels are ignored.

Including binning, a data collection region can be fully specified with six parameters: *s1, p1,s2,p2,sbin,pbin*. Since these values are 0 indexed, the following is true:

smax = serial size -1
pmax = parallel size -1

Data Array

When pixels are read out, they are placed in the data array indicated by the pointer passed into `pl_exp_start_cont or pl_exp_start_seq`. The pixels are placed into an array in the following order:



Display Orientation

0,0 is displayed in the upper left corner, and subsequent pixels are painted from left to right. Although video coordinate configuration can be done in the display routine, factors such as the optical path, the camera rotation, and which readout port is selected may cause the image to appear in a different position.

Port and Speed Choices

The CCD in a camera will have one or more output nodes from which the pixel stream will be read. These nodes are referred to as “Readout Ports”. The signal from a readout port will be passed through an analog to digital converter (ADC) in the case of a CCD and in the case of the CMOS style sensor, will be digitized internally. The ADC (either internal or external to the sensor) operates at one or more digitization rates and has a set of parameters associated with it. In PVCAM, the choice of speed (digitization rate) and associated ADC parameters are organized into a "speed table". In some cameras, different readout ports will be connected to different analog processing chains and different ADCs. The most general method for setting up the port and speed choices is to make the speed choices dependent upon the port selection.

To display more descriptive information about the current port settings, recall `PARAM_READOUT_PORT pl_get_param` with with the `ATTR_CURRENT`. Next, retrieve the

relevant descriptive string related to that readout port by calling `pl_get_enum_param` with `PARAM_READOUT_PORT` on the associated value.

To build the speed table, for each valid port call `pl_get_param` with `PARAM_SPDTAB_INDEX` with the `ATTR_COUNT` attribute to determine how many speed entries are allowed on your camera. Then iterate through each choice to get the associated information for that entry. The steps you should take in setting up the readout ports and associated speed tables are as follows:

1. `pl_get_param` with `PARAM_READOUT_PORT` with `ATTR_COUNT` to get the total number of valid ports.
2. `pl_get_enum_param` with `PARAM_READOUT_PORT` to get the enumerated port constants.
3. For each port constant, `pl_set_param` with `PARAM_READOUT_PORT`, and build a speed table for each.

Table 4 is an example of a camera with two readout ports. Port 1 has one speed associated with it and Port 2 has three speeds. Note that the terms "Port 1" and "Port 2" are generic and are only being used to illustrate the example.

The user chooses the port and then the speed table entry number, and the camera is configured accordingly. The user can then choose one of the gain settings available for that speed table entry number. For example, the user chooses Port 2 and speed index one. This selection provides a 16-bit camera with a pixel time of 500 nanoseconds (a 2 MHz readout rate). The CCD is reading out of Port 2. The gain is set to 2.

Readout Port	Entry	Bit Depth	Pixel Time	Current Gain	Max Gain
	PARAM_SPDTAB_INDEX	PARAM_BIT_DEPTH	PARAM_PIX_TIME	PARAM_GAIN_INDEX	PARAM_GAIN_INDEX with ATTR_MAX
PORT 1	0	12	500	2	16
PORT 2	0	12	100	1	3
	1	16	500	2	3
	2	12	500	2	3

It is the responsibility of the application program to remember variables associated with port and speed selections. The application must resend gain values when the user changes the current port or speed. Additionally, the application must resend the desired speed whenever a readout port is changed. Read-only values, such as bit depth, must be assumed to be unique for each speed-port combination. Once a selection is made, all settings remain in effect until the user resets them or until the camera hardware is powered down or reset.

Multi-tap Configuration and Readout

The term tap will be used to indicate a port that can participate in simultaneous readout. The configuration of multi-tap will be made using the speed table, and the frame data will be spliced together using firmware in the camera, and does not require any modifications to application functionality to support.

Frame Transfer

A frame transfer CCD is divided into two areas: one for image collection and one for image storage. After the CCD is exposed, the image is shifted to the storage array that is not light sensitive. A split clock allows the CCD to expose the next frame of the image array while simultaneously reading out from the storage array.

Since shifting an image to the storage array is many times faster than reading out the same image, frame transfer speeds up many sequences, in comparison to full frame sensors.

With most frame transfer CCDs, the image in the storage array must be completely read out before the next image is shifted into the storage array. Therefore, assuming that the *exposure_time* for each image within a sequence is equal, the shortest possible *exposure_time* would be exactly equal to the image readout time.

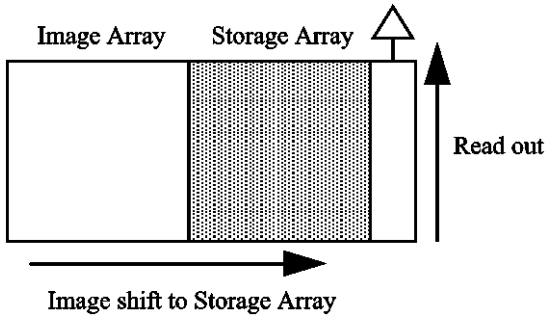
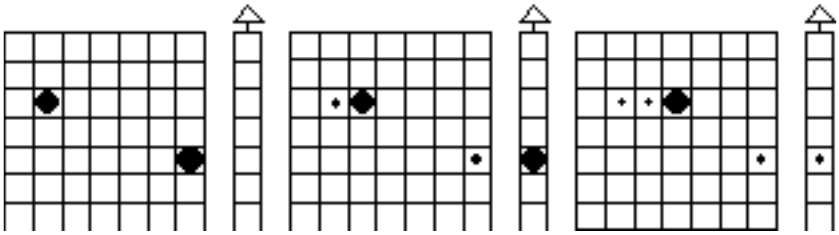
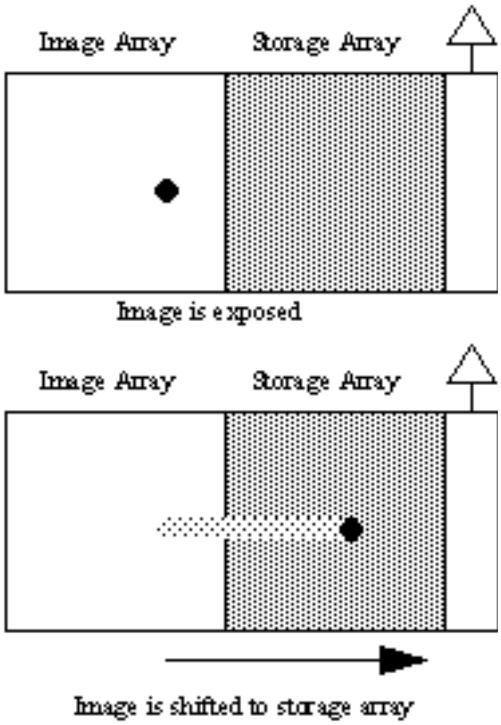


Image Smear

If an image is shifted while the shutter is open, the charge that collects while the image is moving makes the image look smeared. Smearing can occur in several situations: if the camera is set to read out without closing the shutter, if the shutter is set to close too slowly, or in frame transfer sequences where the shutter stays open while the image is shifted to the storage array.



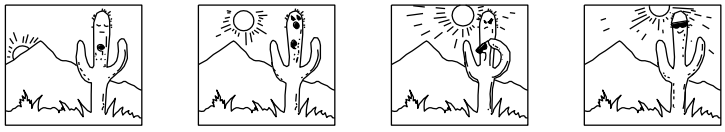
In most frame transfer applications, the shutter opens before the sequence begins and closes after the sequence ends. The charge gathered during the shift creates a smear across the image array.



Although the frame transfer time is usually small compared to the smallest useful exposure time, smearing cannot be eliminated when the shutter is left open for the entire sequence. The higher the ratio of the *exposure_time* to the frame transfer time, the brighter the image is in comparison to the pattern caused by smearing. An *exposure_time* that is too long will saturate the pixels and cause the image to lose all contrast.

Sequences

A sequence is a programmed series of exposures that is started by a single command. In the least complex sequences, a setup is called then the camera takes a series of exposures with a complete readout between each exposure. In these simple sequences, all the variables in the setup apply to all the exposures in the sequence. The diagram below illustrates a sequence of exposures taken as the day passes.



In most camera modes, you must load a new setup into the camera if you want to change a variable between sequences. PVCAM offers a few exceptions to this rule. In variable timed mode, calling a command between sequences sets the *exposure_time* for the next sequence.

Sequence Parameters IDs/Constants

When constructing a sequence, the following three items determine how the camera behaves before reading out:

- **PARAM_CLEAR_MODE parameter id:** Determines if and when the CCD is cleared of charge.
- **BULB_MODE, FLASH_MODE, STROBED_MODE, TIMED_MODE, TRIGGER_FIRST_MODE,** or **VARIABLE_ TIMED_MODE constant :** Determines if a program command or an external trigger starts and ends the exposure/nonexposure time within a sequence.
- **PARAM_SHTR_OPEN_MODE parameter id:** Determines if and when the shutter opens.

Although a single exposure may be considered a sequence of one, some options in triggering, shuttering, and CCD clearing only apply to multiple image sequences.

Circular Buffer

Circular buffers are a special case of sequences. In a sequence, you specify the number of frames to acquire and allocate a buffer large enough to hold all of the frames. Using a circular buffer allows you to acquire a continuous sequence; the camera will continue to acquire frames until you decide to stop it, rather than acquiring a specified number of frames. This mode is especially useful for usecases in which the user is looking for a particular event, as he has no guess as to when that event will occur. Additionally, this mode is recommended for displaying what is called a "focus loop", in which the system's optics are focused on the subject.

For a circular buffer, you allocate a buffer to hold a certain number of frames, and the data from the camera is stored in the buffer sequentially until the end of the buffer is reached. When the end is reached, the data is stored starting at the beginning of the buffer again, and so on as shown in the above figure.

The image buffer used for a circular buffer is passed to `pl_exp_start_cont`. The buffer is allocated by your application. Data readout is performed directly into the designated circular buffer. Depending on the circular buffer mode, overwriting this buffer may be flagged as an error.

Briefly, an example circular buffer setup:

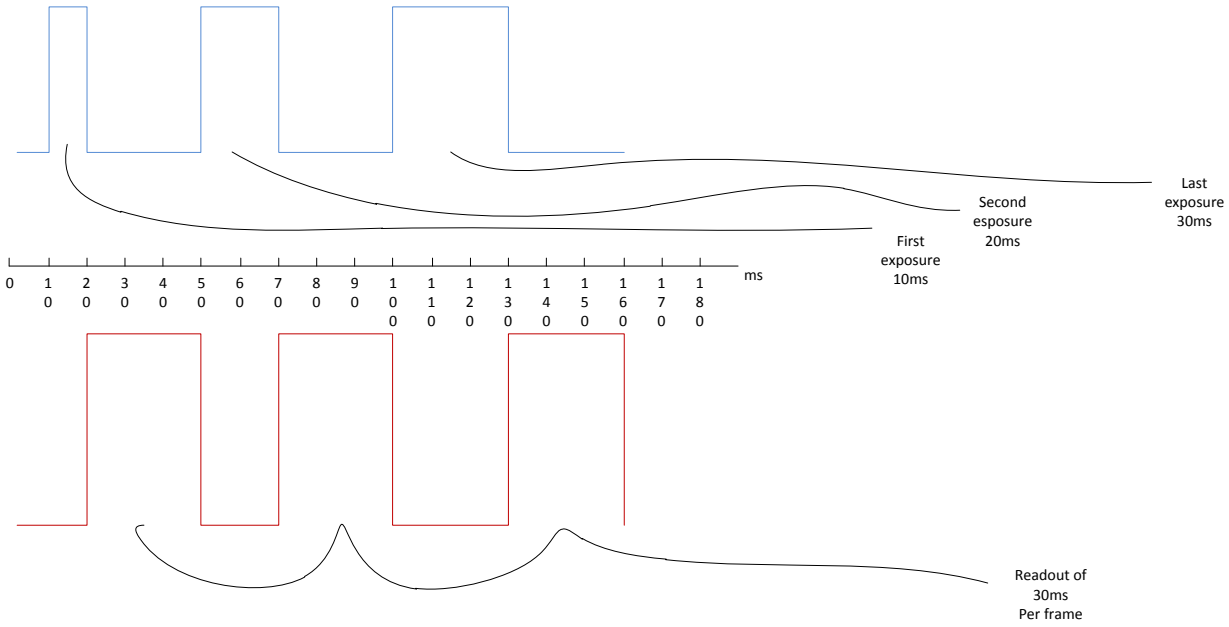
- `pl_exp_init_seq ()`: The camera is prepared to acquire and readout data.
- `pl_exp_setup_cont (circ_overwrite)`: The circular buffer mode is selected.
- `pl_exp_start_cont ()`: Continuous data acquisition is started.
- Frames begin arriving in the buffer.
- `pl_exp_check_cont_status ()`: The status of the buffer is checked.
- `pl_exp_get_latest_frame ()`: If there are one or more frames of data, the most recently stored frame is read out.
- Data is processed (for example, the data is displayed).
- The loop is repeated until continuous data acquisition is stopped with `pl_exp_stop_cont ()`, `pl_exp_finish_seq ()`, and `pl_exp_uninit_seq ()`.

Sequenced Multiple Acquisition Real Time (SMART) streaming

SMART streaming allows you to assign individual exposure settings to each frame of a continuous sequence; the camera will apply the settings just before the frame is captured. Please consult our sales department or camera manuals on information on which cameras support this mode.

The maximum number of SMART streaming entries varies from camera to camera. This parameter can be requested via the ATTR_MAX attribute.

The diagram below illustrates SMART streaming for a sequence of 3 frames with exposures of 10ms, 20ms, 30ms, and no delay between frames.



An Example SMART streaming acquisition setup:

- `pl_exp_init_seq ()`: The camera is prepared to acquire and readout data.
- `pl_exp_setup_seq() (or pl_exp_setup_cont)`: Either single sequence or circular buffer mode is selected.The exposure time in this call needs to be non-zero and it will be overwritten by the exposure times passed by `pl_set_param()` with `PARAM_SMART_STREAM_EXP_PARAMS`.
- `pl_set_param()` with `PARAM_SMART_STREAM_MODE_ENABLED`: SMART stream mode is enabled.
- `pl_set_param()` with `PARAM_SMART_STREAM_EXP_PARAMS`:The exposure parameters are passed in to the camera.
- `pl_exp_start_seq() or pl_exp_start_cont ()`: Data acquisition is started.
- The loop is repeated until the buffer fills up or continuous data acquisition is stopped with `pl_exp_stop_cont ()`, `pl_exp_finish_seq ()`, and `pl_exp_uninit_seq ()`.

SMART Streaming Data Types

A SMART streaming acquisition is programmed by sending the camera a list of the individual exposures or delays along with the frame count. To facilitate this process, the `smart_stream_type` encapsulates the required parameters. This data type consists of an `uns16` variable called `entries` and an `uns32_ptr` variable called `params`. The `params` variable points to a list of exposures or delays; the `entries` variable contains the number of entries in the list.

A variable of type `smart_stream_type` can be filled in two ways:

1. Statically. For example:

```
smart_stream_type ExposureArray;
uns32 exp_values[] = {10, 20 ,30,40};

ExposureArray.entries = sizeof(exp_values)/sizeof(uns32);
ExposureArray.params = exp_values;

/* acquire the data here */
```

2. Dynamically with the aid of the function `pl_create_smart_stream_struct()`. For example:

```
smart_stream_type_ptr pExposureArray;

pl_create_smart_stream_struct(&pExposureArray,4);

for (int i = 0; i < pExposureArray->entries; i++)
 pExposureArray->params[i] = 10 + i*10;

/* acquire the data here */

pl_release_smart_stream_struct(&pExposureArray);
```

Possible scenarios

a. S.M.A.R.T. Streaming enabled but no arrays passed in.

In this case, only the exposure as defined in the `pl_exp_setup_seq` will be used.

b. Exposure Count (N) > Sequence Size (M)

For a finite sequence, the first N entries will be used and the rest will be ignored. For a circular buffer sequence, all of the defined entries will be used in a round-robin fashion (i.e. after a frame has been captured with the last entry defined in the SMART stream parameter, the cycle will be repeated with the first defined entry, then the second, etc.)

c. Exposure Count (N) < Sequence Size (M)

For a finite sequence, the defined entries will be used in a round-robin fashion and repeated until all frames are consumed. For a circular buffer sequence, all of the defined entries will be used in a round-robin fashion (i.e. after a frame has been captured with the last entry defined in the SMART stream parameter, the cycle will be repeated with the first defined entry, then the second, etc.)

Refer to **Example 5: S.M.A.R.T Streaming Mode Acquisition** in Chapter 8 for an example of code for this type of operation.

Clear Modes

Clearing removes charge from the CCD by clocking the charge to the serial register then directly to ground. This process is much faster than a readout, because the charge does not go through the readout node or the amplifier. Note that not all clearing modes are available for all cameras. Be sure to check availability of a mode before attempting to set it.

The clear modes are described below:

- **CLEAR_NEVER:** Don't ever clear the CCD. Useful for performing a readout after an exposure has been aborted.
- **CLEAR_PRE_EXPOSURE:** Before each exposure, clears the CCD the number of times specified by the *clear_cycles* variable. This mode can be used in a sequence. It is most useful when there is a considerable amount of time between exposures.
- **CLEAR_PRE_SEQUENCE:** Before each sequence, clears the CCD the number of times specified by the *clear_cycles* variable. If no sequence is set up, this mode behaves as if the sequence has one exposure. The result is the same as using CLEAR_PRE_EXPOSURE.
- **CLEAR_POST_SEQUENCE:** Clears continuously after the sequence ends. The camera continues clearing until a new exposure is set up or started, the abort command is sent, the speed entry number is changed, or the camera is reset.
- **CLEAR_PRE_POST_SEQUENCE:** Clears *clear_cycles* times before each sequence and clears continuously after the sequence ends. The camera continues clearing until a new exposure is set up or started, the abort command is sent, the speed entry number is changed, or the camera is reset.
- **CLEAR_PRE_EXPOSURE_POST_SEQ:** Clears *clear_cycles* times before each exposure and clears continuously after the sequence ends. The camera continues clearing until a new exposure is set up or started, the abort command is sent, the speed entry number is changed, or the camera is reset.

Normally during the idle period, the Camera parallel and serial clock drivers revert to a low power state that saves both power and heat. When CLEAR_..._POST options are used, the clearing prevents these systems from entering low-power mode. This state generates a small amount of additional heat in the electronics unit and the camera head.

The pl_exp_abort() function stops the data acquisition and the camera goes into the clean cycle. Again, the CCD chip is continuously being cleaned.

Clear Modes decide when to empty the CCD wells.

Exposure Modes

During sequences, the exposure mode determines how and when each exposure begins and ends:

TIMED_MODE	STROBED_MODE
VARIABLE_ TIMED_MODE	BULB_MODE
TRIGGER_FIRST_MODE	FLASH_MODE

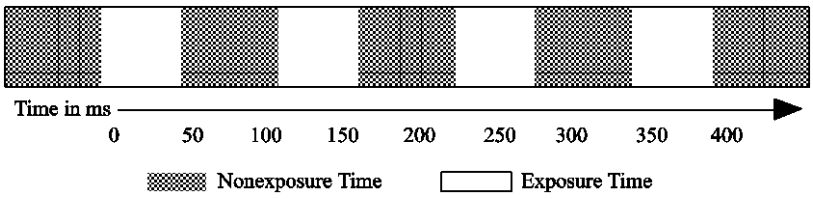
In general, the settings in `pl_exp_setup_seq` apply to each exposure within a sequence. They also apply to every sequence until the *setup* is reset. The only exceptions are in `VARIABLE_TIMED_MODE` and `BULB_MODE`. These two modes ignore the *`exposure_time`* parameter in setup, and rely on a function or trigger to determine the exposure time.

Every sequence has alternating periods of exposure and nonexposure time. During the time the CCD is not exposing, the camera could be in several states, such as waiting for `pl_exp_start_seq`, reading out, or performing clearing. In the diagrams that follow, each exposure mode shows the exposure time in white and the time between exposures in gray.

Exposure: TIMED_MODE

In `TIMED_MODE`, all settings are read from the *setup* parameters, making the duration of each exposure time constant and the interval times between exposures constant. In this mode, every sequence has the same settings.

The diagram below represents a sequence in `TIMED_MODE`.

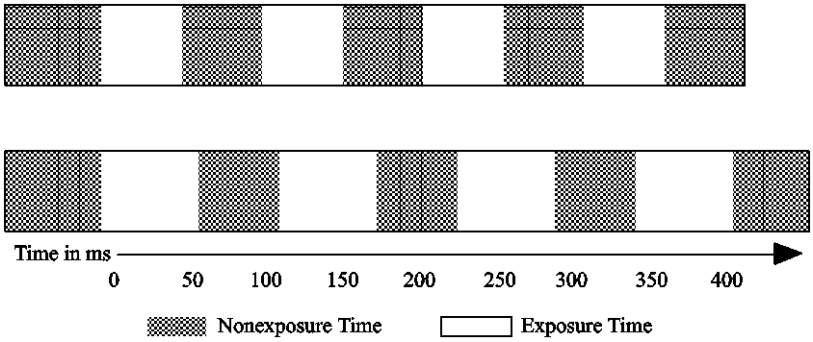


Exposure: VARIABLE_TIMED_MODE

Use `VARIABLE_TIMED_MODE` when you want to change the *exposure_time* between sequences.

In `VARIABLE_TIMED_MODE`, all settings except *`exposure_time`* are read from the setup parameters. The *`exposure_time`* must be set with parameter id `PARAM_EXP_TIME`. If you do not call `PARAM_EXP_TIME` before the first sequence, a random time will be assigned. The camera will not read the first exposure time from the *`exposure_time`* in setup, because this mode ignores the *`exposure_time`* parameter.

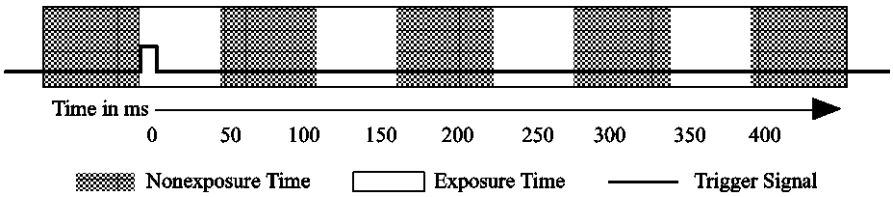
Application example: A filter wheel is used to change the filter color between sequences. The exposure time needed for the darkest filter saturates the pixels when lighter filters are used. The diagram on the next page shows two sample sequences from this example.



The first sequence runs with a filter that uses exposure and nonexposure times that are equal. In the second sequence, the exposure time is longer, but the time between exposures remains the same as in the first sequence.

Exposure: TRIGGER_FIRST_MODE

Use `TRIGGER_FIRST_MODE` when you want an external trigger to signal the start of the sequence.



In `TRIGGER_FIRST_MODE`, *`pl_exp_start_seq`* starts the camera, which enters the clear mode while it waits for a trigger signal. The black line in the diagram illustrates a trigger signal coming from an external trigger source.

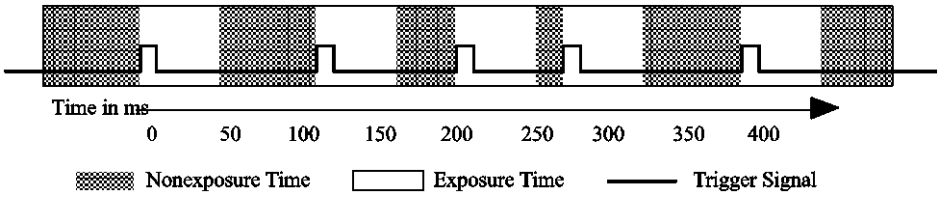
Once the outside event triggers the camera to start exposing, the sequence follows the conditions generated in *`pl_exp_setup_seq`*. Note that all exposure times are equal, and the time intervals between exposures are equal.

You must have an external trigger signal connected to your camera for `TRIGGER_FIRST_MODE` to function. If your equipment fails to send a trigger signal, you can stop the sequence by calling *`pl_exp_abort`*.

Note: If you do not use one of the `CLEAR_PRE_EXPOSURE` modes, the CCD will begin exposing immediately after *`pl_exp_start_seq`* is called. Once the trigger is received, the CCD will continue to expose for the *`exposure_time`* specified in *`pl_exp_setup_seq`*. In other words, the first exposure in your sequence may have a longer exposure time than the subsequent exposures.

Exposure: STROBED_MODE

Use `STROBED_MODE` when you want an external trigger to start each exposure in the sequence.



In STROBED_MODE, *pl_exp_start_seq* starts the camera. The camera enters clear mode while it waits for the first trigger signal to start the first exposure. As shown in the diagram above, each new exposure waits for an external trigger signal. Notice that the intervals between exposures can vary greatly, but the exposure times are constant.

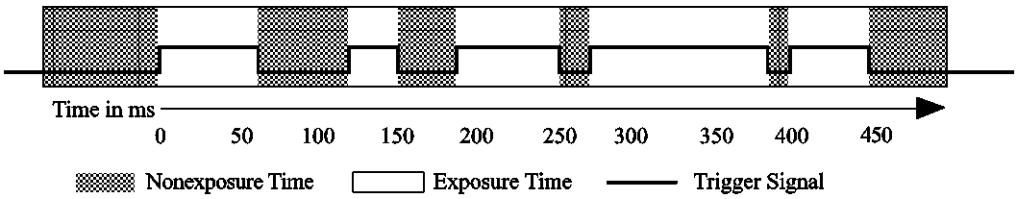
You must have an external trigger signal connected to your camera for this mode to function. If your equipment fails to send a trigger signal, you can stop the sequence by calling *pl_exp_abort*.

Application example: In a nature study of birds passing through a restricted area, the motion of each bird sends a trigger signal to the camera. The camera exposes, reads out, and waits for the next trigger signal. The result is an image of each bird as it crosses the camera's field of view.

Note: If you do not use one of the CLEAR_PRE_EXPOSURE modes, the CCD will begin exposing immediately after *pl_exp_start_seq* is called. Once the trigger is received, the CCD will continue to expose for the *exposure_time* specified in *pl_exp_setup_seq*. In other words, the first exposure in your sequence may have a longer exposure time than the subsequent exposures.

Exposure: BULB_MODE

Use BULB_MODE, when you want an external trigger signal to control the beginning and end of each exposure.



In BULB_MODE, *pl_exp_start_seq* calls the setup. The camera enters clear mode while it waits for a **true** external trigger signal to start each exposure. The CCD continues to expose until a **false** trigger signal ends the exposure. In the diagram above, the trigger signal line moves up to represent a **true** trigger and down to represent a **false** trigger.

Notice that the exposure times and the intervals between exposures vary greatly. Since the **true** and **false** signals determine exposure time, the *exposure_time* set in *pl_exp_setup_seq* is ignored.

You must have an external trigger signal connected to your camera for BULB_MODE to function. If your equipment fails to send a trigger signal, you can stop the sequence by calling *pl_exp_abort*.

Note: If you do not use one of the CLEAR_PRE_EXPOSURE modes, the CCD exposes until receiving a false trigger signal, then reads out. After reading out, the CCD exposes again without clearing and waits for the true trigger. Once the external event causes a true trigger, the CCD continues to expose until receiving a false trigger, then reads out. In other words, the CCD will expose from the end of readout until the next false trigger.

Exposure: FLASH_MODE

Some PVCAM cameras include a flash port—several outside pins with a software-controllable signal. Photometrics uses these pins to drive factory test fixturing. However, the signal can be used to drive other equipment. Aside from the signal on the pins, FLASH_MODE is identical to TIMED_MODE. Consult your camera hardware documentation to see flash port availability and electrical specifications.

Extended Exposure Modes

Since PVCAM 3.0.1 the newly added Exposure Modes are not defined in the PVCAM header but can be retrieved from the camera directly. Application developers are encouraged to not hard code the exposure modes in the source code but read the supported modes from the camera dynamically. This change was introduced together with the addition of the Expose Out Modes.

The following example shows how to use the Extended Exposure Modes together with Expose Out Modes and still keep the backward compatibility with older cameras.

Define a helper function to retrieve available parameter values:

```
//
// A helper function that enumerates a given parameter from the camera
//
void enumerateParameter(int16 hCam, uns32 paramID, std::vector<int32>& values,
                        std::vector<std::string>& names)
{
    rs_bool bAvail = FALSE;
    uns32 count = 0;
    values.clear();
    names.clear();
    // Check the availability of the parameter
    if (pl_get_param(hCam, paramID, ATTR_AVAIL, &bAvail) != PV_OK || bAvail == FALSE)
        return;
    // Get the number of expose out modes
    if (pl_get_param(hCam, paramID, ATTR_COUNT, &count) != PV_OK)
        return;
    // Get the mode values and names
    for ( uns32 i = 0; i < count; ++i )
    {
        uns32 enumStrLen;
        if (pl_enum_str_length(hCam, paramID, i, &enumStrLen) == PV_OK)
        {
            char* enumStr = new char[enumStrLen+1](); // Allocate and null a string buffer
            int32 enumVal;
            if (pl_get_enum_param(hCam, paramID, i, &enumVal, enumStr, enumStrLen)
                == PV_OK)
            {
                values.push_back(enumVal);
                names.push_back(std::string(enumStr));
            }
            delete[] enumStr;
        }
    }
}
```

Setup the acquisition:

```
void SetupAcquisition(int16 hCam)
{
    rs_bool bAvail;

    std::vector<int32> tringModeVals;
    std::vector<std::string> tringModeStrs;
    int16 selectedTrigMode = 0;

    std::vector<int32> expOutModeVals;
    std::vector<std::string> expOutModeStrs;
    int16 selectedExpOutMode = 0;

    if (pl_get_param(hCam, PARAM_EXPOSURE_MODE, ATTR_AVAIL, &bAvail) != PV_OK
        || bAvail == FALSE)
    {
        selectedTrigMode = TIMED_MODE;
    }
    else
    {
        // The enumeration should be done upon opening the camera when an UI element
        // can be populated with available exposure modes
```

```
        enumerateParameter(hCam, PARAM_EXPOSURE_MODE, tringModeVals, tringModeStrs);
        selectedTrigMode = (int16)tringModeVals[0]; // Or any other selected by user
    }

    if (pl_get_param(hCam, PARAM_EXPOSE_OUT_MODE, ATTR_AVAIL, &bAvail) != PV_OK
        || bAvail == FALSE)
    {
        selectedExpOutMode = 0; // This will have no effect when doing bitwise OR
    }
    else
    {
        enumerateParameter(hCam, PARAM_EXPOSE_OUT_MODE, expOutModeVals, expOutModeStrs);
        selectedExpOutMode = (int16)expOutModeVals[0]; // Or any other selected by user
    }

    const int16 finalExpMode = selectedTrigMode | selectedExpOutMode;

    const rgn_type roi = { 0, m_ccdSerSz-1, 1, 0, m_ccdParSz-1, 1 }; // Acquire full frame
    uns32 bufferSize;
    // Setup the acquisition, 1 frame, 1 roi, 10ms exposure
    if ( pl_exp_setup_seq( hCam, 1, 1, &roi, finalExpMode, 10, &bufferSize ) != PV_OK )
        return false;

    // Next: Allocate the buffer and call pl_exp_start_cont()
}
```

Expose Out Modes

Expose Out Modes determine the behavior of the camera expose out IO signal. This parameter is camera dependent, please refer to your camera manual for more information.

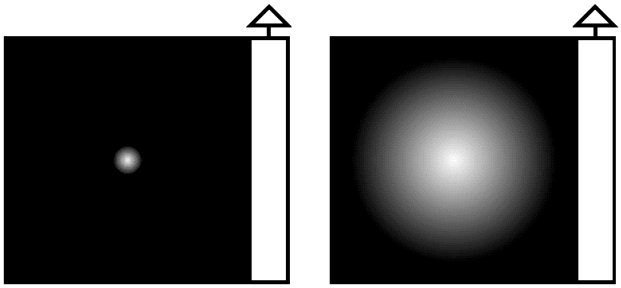
Since PVCAM 3.0.1 the new enumerable parameter values are not hardcoded in the PVCAM header but can be retrieved dynamically from the currently connected camera.

Please refer to the "*Extended Exposure Modes*" chapter on page 26 for a code example showing how to retrieve and use the Expose Out Modes with new cameras.

Open Delay, Close Delay

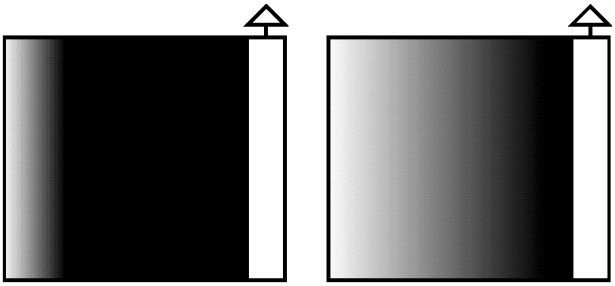
In order to ensure that the entire CCD is exposed for the specified *exposure_time*, the mechanical limitations of the shutter must be considered. Open delay (PARAM_SHTR_OPEN_DELAY) and close delay (PARAM_SHTR_CLOSE_DELAY) account for the time necessary for the shutter to open and close. Remember that the camera is exposing while the shutter is opening and closing, so some pixels are exposed longer than others.

Iris Shutter



An Iris shutter opens in an expanding circular pattern.

Barn Door Shutter



A Barn Door shutter slides across the exposure area.

If the shutter is still closing when the image shifts for a frame transfer or readout, the image will smear. (See the section *"Image Smear"* for a more complete explanation on smearing.) PARAM_SHTR_CLOSE_DELAY allows time for the shutter to close before the image shifts.

The default open and close delay values will vary depending on the brand of camera and the shutter used. Open delay may be up to 15 milliseconds with a close delay of up to 30 milliseconds. Change the default values only if you are using a shutter other than the shutter shipped with your camera. **If you are using a standard Photometrics shutter, changing PARAM_SHTR_OPEN_DELAY/CLOSE_DELAY default values will not increase the frame transfer rate.**

Shutter Control

The shutter open modes determine how the shutter in a camera behaves when a single exposure is taken or when a sequence is run. Remember that the camera is exposing while the shutter is opening. Because not all supported cameras have programmable shutter control, remember to check for availability of a particular mode.

- **OPEN_PRE_EXPOSURE:** Opens the shutter before every exposure, then closes the shutter after the exposure is finished.
- **OPEN_PRE_SEQUENCE:** Opens the shutter before the sequence begins, then closes the shutter after the sequence is finished.
- **OPEN_PRE_TRIGGER:** Opens the shutter, then clears or exposes (set in clear mode) until a trigger signal starts the exposure.
- **OPEN_NEVER:** Keeps shutter closed during the exposure. Used for dark exposures.
- **OPEN_NO_CHANGE:** Sends no signals to open or close the shutter.

Exposure Loops

Within an exposure loop, the interaction of the exposure, clear, and shutter open modes determines how the camera behaves during a sequence. In the following pages, sample command sequences show how each exposure mode acts in combination with each clear and shutter open mode. As mentioned above in *"Shutter Control"*, not all supported cameras have programmable shutter control, remember to check for availability of a particular mode.

Key	Description
ClearN	Clear CCD N times as specified in `clear_cycles`
OS	Open shutter and perform `PARAM_SHTR_OPEN_DELAY`
CS	Close shutter and perform `PARAM_SHTR_CLOSE_DELAY`
EXP	Expose CCD for `exposure_time`
I->S	Transfer image array to storage array (frame transfer)
Readout	Readout CCD (readout storage array for frame transfer)
WaitT	Wait until trigger
EXP Until notT	Expose CCD until trigger end (`BULB_MODE`)
Items in *ITALICS* repeat M times for a sequence of M exposures.	
Items in **BOLD** are outside of the sequence loop.	

EXPOSURE: TIMED_MODE		
Clear Mode	**Shutter Mode**	**Command Sequence**
CLEAR_PRE_EXPOSURE	OPEN_PRE_EXPOSURE	*ClearN, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS** , *ClearN, EXP, I->S, Readout*, **CS**
	OPEN_PRE_TRIGGER	*ClearN, OS, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	*ClearN, EXP, I->S, Readout*
	OPEN_NEVER	***CS,*** *ClearN, EXP, I->S, Readout*
CLEAR_PRE_SEQUENCE	OPEN_PRE_EXPOSURE	**ClearN**,*OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS, ClearN**, *EXP, I->S, Readout*, **CS**
	OPEN_PRE_TRIGGER	**ClearN**, *OS, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	**ClearN**, *EXP, I->S, Readout*
	OPEN_NEVER	***CS,*** **ClearN**, *EXP, I->S, Readout*
CLEAR_NEVER	OPEN_PRE_EXPOSURE	*OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS,** *EXP, I->S, Readout*, **CS**
	OPEN_PRE_TRIGGER	*OS, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	*EXP, I->S, Readout*
	OPEN_NEVER	***CS,*** *EXP, I->S, Readout*

EXPOSURE: TRIGGER_FIRST_MODE		
Clear Mode	**Shutter Mode**	**Command Sequence**
CLEAR_PRE_EXPOSURE	OPEN_PRE_EXPOSURE	**EXP+WaitT**, *ClearN, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS, EXP+WaitT**, *ClearN, EXP, I->S, Readout*, **CS**
	OPEN_PRE_TRIGGER	**EXP+WaitT**, *OS, ClearN, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	**EXP+WaitT**, *ClearN, EXP, I->S, Readout*
	OPEN_NEVER	***CS,*** **EXP+WaitT**, *ClearN, EXP, I->S, Readout*
CLEAR_PRE_SEQUENCE	OPEN_PRE_EXPOSURE	**Clear+WaitT**, *ClearN, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS, Clear+WaitT**, *EXP, I->S, Readout*, **CS**
	OPEN_PRE_TRIGGER	**Clear+WaitT**, *OS, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	**Clear+WaitT**, *EXP, I->S, Readout*
	OPEN_NEVER	***CS,*** **Clear+WaitT**, *EXP, I->S, Readout*
CLEAR_NEVER	OPEN_PRE_EXPOSURE	**EXP+WaitT**, *ClearN, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS**, **EXP+WaitT,** *EXP, I->S, Readout*, **CS**

EXPOSURE: TRIGGER_FIRST_MODE		
Clear Mode	**Shutter Mode**	**Command Sequence**
	OPEN_PRE_TRIGGER	**EXP+WaitT**, *OS, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	**EXP+WaitT**, *EXP, I->S, Readout*
	OPEN_NEVER	***CS,* EXP+WaitT**, *EXP, I->S, Readout*

EXPOSURE: STROBED_MODE		
Clear Mode	**Shutter Mode**	**Command Sequence**
CLEAR_PRE_EXPOSURE	OPEN_PRE_EXPOSURE	*Clear+WaitT, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS**, *Clear+WaitT, EXP, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	*OS, Clear+WaitT, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	*Clear+WaitT, EXP, I->S, Readout*
	OPEN_NEVER	***CS,** Clear+WaitT, EXP, I->S, Readout*
CLEAR_PRE_SEQUENCE	OPEN_PRE_EXPOSURE	**ClearN**, *EXP+WaitT, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS, ClearN,** *EXP+WaitT, EXP, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	**ClearN**, *OS, EXP+WaitT, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	**ClearN**, *EXP+WaitT, EXP, I->S, Readout*
	OPEN_NEVER	***CS,* ClearN**, *EXP+WaitT, EXP, I->S, Readout*
CLEAR_NEVER	OPEN_PRE_EXPOSURE	*EXP+WaitT, OS, EXP, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS,** *EXP+WaitT, EXP, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	*OS, EXP+WaitT, EXP, CS, I->S, Readout*
	OPEN_NO_CHANGE	*EXP+WaitT, EXP, I->S, Readout*
	OPEN_NEVER	***CS,** EXP+WaitT, EXP, I->S, Readout*

EXPOSURE: BULB_MODE		
Clear Mode	**Shutter Mode**	**Command Sequence**
CLEAR_PRE_EXPOSURE	OPEN_PRE_EXPOSURE	*Clear+WaitT, OS, EXP Until notT, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS**, *Clear+WaitT, EXP Until notT, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	*OS, Clear+WaitT, EXP Until notT, CS, I->S, Readout*
	OPEN_NO_CHANGE	*Clear+WaitT, EXP Until notT, I->S, Readout*
	OPEN_NEVER	***CS,*** *Clear+WaitT, EXP Until notT, I->S, Readout*
CLEAR_PRE_SEQUENCE	OPEN_PRE_EXPOSURE	**ClearN,** *EXP+WaitT, OS, EXP Until notT, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS, ClearN,** *EXP+WaitT, EXP Until notT, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	**ClearN**, *OS, EXP+WaitT, EXP Until notT, CS, I->S, Readout*
	OPEN_NO_CHANGE	**ClearN**, *EXP+WaitT, EXP Until notT, I->S, Readout*
	OPEN_NEVER	***CS,*** **ClearN**, *EXP+WaitT, EXP Until notT, I->S, Readout*
CLEAR_NEVER	OPEN_PRE_EXPOSURE	*EXP+WaitT, OS, EXP Until notT, CS, I->S, Readout*
	OPEN_PRE_SEQUENCE	**OS**, *EXP+WaitT, EXP Until notT, I->S, Readout,* **CS**
	OPEN_PRE_TRIGGER	*OS, EXP+WaitT, EXP Until notT, CS, I->S, Readout*
	OPEN_NO_CHANGE	*EXP+WaitT, EXP Until notT, I->S, Readout*
	OPEN_NEVER	***CS,*** *EXP+WaitT, EXP Until notT, I->S, Readout*

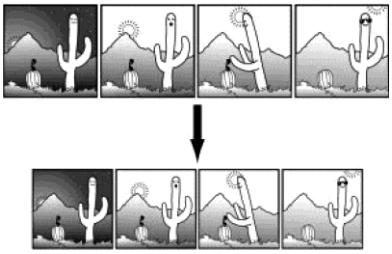
Image Buffers

When exposures include multiple images and complex sequences, you may choose to store the images in a buffer. PVCAM has a number of buffer routines that handle memory allocation and freeing. The following list describes images you may choose to store in a buffer.

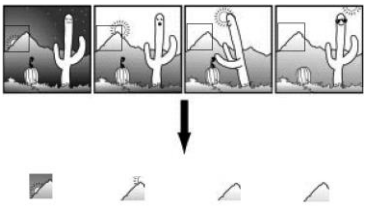
- **Full CCD:** A single exposure where the entire CCD is treated as one region and image data are collected over the full CCD. All the data are stored in a single buffer.

- **Single Exposure, Custom Region:** A single exposure with a region. Less data than the full CCD produces is stored in the single buffer.

- **Sequences:** A series of exposures with identical regions. The data are stored in several image arrays that are stored inside a single buffer.



- **Multiple Exposures, Custom Region:** A series of exposures with a single region. Each exposure must have an identical region. The data is all stored in a single buffer.



PVCAM collects data very efficiently, but moving the data in and out of a buffer involves extra processing time. If speed is crucial, the following options may minimize processing time:

- Don't use an extra buffer. The data are collected in a user-specified pixel stream at maximum efficiency (see *pl_exp_start_seq*). As discussed in *"Data Array"*, this array can be accessed directly. However, when a region is collected, the stream becomes more complex.
- Defer decoding. The original call to *pl_exp_setup_seq* sets up internal structures used to decode pixel_stream into a buffer structure. However, *pl_exp_finish_seq* does not need to be called immediately. As long as the camera (and library) remains open, and *pl_exp_setup_seq* is not called with a new setup, the decoding structures remain valid.

This allows a program to collect data quickly, then decode the data when more time is available. Of course, this is impossible if users must be given immediate feedback.

Chapter 3: Camera Communications (Class 0)

Introduction

The functions in this category provide a pipeline for bidirectional communications. The table below lists the current Class 0 functions, and the "Class 0 Functions" section provides detailed descriptions of each. For more information about the `pl_get_param` and `pl_set_param` parameter ids, refer to "*Chapter 5:Configuration/Setup (Class 2)*", starting on page 61.

List of Available Class 0 Functions

Library	Camera
`pl_pvcam_init`	`pl_cam_close`
`pl_pvcam_uninit`	`pl_cam_get_name`
`pl_pvcam_get_ver`	`pl_cam_get_total`
	`pl_cam_open`
	`pl_cam_register_callback`
	`pl_cam_register_callback_ex`
	`pl_cam_register_callback_ex2`
	`pl_cam_register_callback_ex3`
	`pl_cam_deregister_callback`

List of Available Class 0 Parameter IDs

The following are available Class 0 parameters used with `pl_get_param()`, `pl_set_param()`, `pl_get_enum_param()`, and `pl_enum_str_length()` functions specified in Chapter 5.

`PARAM_DD_INFO`	`PARAM_DD_TIMEOUT`
`PARAM_DD_INFO_LENGTH`	`PARAM_DD_VERSION`
`PARAM_DD_RETRIES`	

Class 0 Functions

PVCAM	**Class 0: Camera Communications**	**pl_cam_close(0)**

NAME

`pl_cam_close` – frees the current camera, prepares it for power-down.

SYNOPSIS

```
rs_bool
   pl_cam_close(int16 hcam)
```

DESCRIPTION

This has two effects. First, it removes the listed camera from the reserved list, allowing other users to open and use the hardware. Second, it performs all cleanup, close-down, and shutdown preparations needed by the hardware. A camera can only be closed if it was previously opened; *`hcam`* must be a valid camera handle.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_open(0),pl_pvcam_init(0),pl_pvcam_uninit(0)`

NOTES

`pl_pvcam_uninit` automatically calls a `pl_cam_close` on all cameras opened by the current user.

PVCAM **Class 0: Camera Communications** **pl_cam_get_name(0)**

NAME

`pl_cam_get_name` – returns the name of a camera.

SYNOPSIS

```
rs_bool
   pl_cam_get_name(int16 cam_num,char_ptr cam_name)
```

DESCRIPTION

This function allows a user to learn the string identifier associated with every camera on the current system. This is a companion to the `pl_cam_get_total` function. Cam_num input can run from 0 to (*`total_cams`* - 1), inclusive. The user must pass in a string that is at least `CAM_NAME_LEN` characters long; `pl_cam_get_name` then fills that string with an appropriate null-terminated string. *`Cam_name`* can be passed directly into the `pl_cam_open` function. It has no other use, aside from providing a brief description of the camera.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_get_total(0),pl_cam_open(0),pl_cam_close(0)`

NOTES

This call reports the names of all cameras on the system, even if all the cameras are not available. If the hardware is turned off, or if another user has a camera open, the camera name is reported, but is not available.

`pl_cam_get_name` returns a name, and `pl_cam_open` gives information on availability of that camera. This function actually searches for all device drivers on the system, without checking hardware. To build a complete list of every camera on the system, it is necessary to cycle through all entries, as shown below:

```
int total_cameras;
char cam_name[CAM_NAME_LEN];
…
pl_cam_get_total(&total_cameras );
for( I=0; I<total_cameras; I++ ) {
pl_cam_get_name(I,cam_name);
printf("Camera%d is called '%s'\n",I,cam_name);
}
```

PVCAM	**Class 0: Camera Communication**	**pl_cam_get_total(0)**

NAME

`pl_cam_get_total` – returns the number of cameras attached to the system.

SYNOPSIS

```
rs_bool
   pl_cam_get_total(int16_ptr total_cams)
```

DESCRIPTION

This reports on the number of cameras on the system. All listed cameras may not all be available; on multi-tasking systems, some cameras may already be in use by other users. A companion function, `pl_cam_get_name`, can be used to learn the string identifier associated with each camera.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_get_name(0),pl_cam_open(0),pl_cam_close(0)`

NOTES

This function actually searches for all device drivers on the system, without checking hardware. The list of cameras is obtained during `pl_pvcam_init`. Thus, if a new camera (new device driver) is added after the library was opened, the system won't know that the new camera is there. The system also won't notice if a camera is removed. (Obviously, this is only important on multi-tasking systems). A cycle of *`uninit/init`* regenerates the list of available cameras, updating the system for any additions or deletions.

PVCAM	**Class 0: Camera Communications**	**pl_cam_open(0)**

NAME

`pl_cam_open` – reserves and initializes the camera hardware.

SYNOPSIS

```
rs_bool
   pl_cam_open(char_ptr cam_name,int16_ptr hcam,int16 o_mode)
```

DESCRIPTION

The string cam_name should be identical to one of the valid camera names returned by `pl_cam_get_name`. If the name is valid, `pl_camera_open` completes a short set of checks and diagnostics as it attempts to establish communications with the camera electronics unit. If successful, the camera is opened and a valid camera handle is passed back in *`hcam`*. Otherwise, `pl_cam_open` returns with a failure. An explanation is shown in `pl_error_code`.

The *`o_mode`* setting controls the mode under which the camera is opened. Currently, the only possible choice is `OPEN_EXCLUSIVE`. On multi-user systems, opening a camera under the exclusive mode reserves it for the current user, locking out all other users on the system. If `pl_cam_open` is successful, the user has sole access to that camera until the camera is closed or `pl_pvcam_uninit` is called.

WARNING

Despite the above paragraph, a **successful** `pl_cam_open` does not mean that the camera is in working order. It **does** mean that you can communicate with the device driver associated with the camera. After a successful `pl_cam_open`, call `pl_error_message`, which reports any error conditions.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_get_name(0), pl_cam_get_total(0), pl_cam_close(0), pl_pvcam_init(0), pl_pvcam_uninit(0)`

NOTES

PVCAM **Class 0: Camera Communication** **pl_pvcam_get_ver(0)**

NAME

`pl_pvcam_get_ver` – returns the PVCAM version number.

SYNOPSIS

```
rs_bool
   pl_pvcam_get_ver(uns16_ptr version)
```

DESCRIPTION

This returns a version number for this edition of PVCAM. The version is a highly formatted hexadecimal number, of the style:

low byte	----------	----------
high byte	hi nibble	low nibble
major version	minor version	trivial version

For example, the number 0x11F1 indicates major release 17, minor release 15, and trivial change 1.

A major release is defined as anything that alters the interface, calling sequence, parameter list, or interpretation of any function in the library. This includes new functions and alterations to existing functions, but it does not include alterations to the options libraries, which sit on top of PVCAM (each option library includes its own, independent version number).

A new major release often requires a change in the PVCAM library, but wherever possible, major releases are backward compatible with earlier releases.

A minor release should be completely transparent to higher-level software (PVCAM) but may include internal enhancements. The trivial version is reserved for use by the software staff to keep track of extremely minor variations. The last digit is used for build numbers, and should be ignored. Minor and trivial releases should require no change in the calling software.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

parameter id `PARAM_DD_VERSION`

NOTES

PVCAM	**Class 0: Camera Communication**	**pl_pvcam_init(0)**

NAME

`pl_pvcam_init` – opens and initializes the library.

SYNOPSIS

```
rs_bool
   pl_pvcam_init(void)
```

DESCRIPTION

The PVCAM library requires significant system resources: memory, hardware access, etc. `pl_pvcam_init` prepares these resources for use, as well as allocating whatever static memory the library needs. Until `pl_pvcam_init` is called, every PVCAM function (except for the error reporting functions) will fail and return an error message that corresponds to "`library has not been initialized`".

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_pvcam_uninit(0),pl_cam_open(0),pl_error_code(1)`

NOTES

If this call fails, `pl_error_code` contains the code that lists the reason for failure.

PVCAM	**Class 0: Camera Communication**	**pl_pvcam_uninit(0)**

NAME

`pl_pvcam_uninit` – closes the library, closes all devices, frees memory.

SYNOPSIS

```
rs_bool
   pl_pvcam_uninit(void)
```

DESCRIPTION

This releases all system resources that `pl_pvcam_init` acquired. It also searches for all cameras that the user has opened. If it finds any, it will close them before exiting. It will also unlock and free memory, and clean up after itself as much as possible.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_pvcam_init(0),pl_cam_close(0),pl_error_code(1)`

KNOWN BUGS

If the hardware is involved in acquiring data, the system may not be able to disconnect immediately.

PVCAM **Class 0: Camera Communication** **pl_cam_register_callback(0)**

NAME

`pl_cam_register_callback` – installs a function that will be called when an event occurs in a camera system.

SYNOPSIS

```
rs_bool
   pl_cam_register_callback(
      int16 hcam,
      PL_CALLBACK_EVENT event,
      void *Callback
      )
```

DESCRIPTION

Use this API call to install a function that will be called when the specified event occurs with respect to the camera system indicated.

The **hcam** parameter must reference an open camera system.

The **event** parameter must be one of the following:

PL_CALLBACK_BOF
PL_CALLBACK_EOF
PL_CALLBACK_CHECK_CAMS
PL_CALLBACK_CAM_REMOVED
PL_CALLBACK_CAM_RESUMED

The Callback function must be a function taking no parameters and returning no value. For example:

```
void BOFCallback (void)
{
     BOFCount++;
     return;
}
```

WARNING

`pl_exp_finish_seq` must be called if acquiring in sequential mode (using `pl_exp_setup_seq` and `pl_exp_start_seq`) with callbacks notification after a frame is read out and before new exposure is started by calling `pl_exp_start_seq`.

Not all callbacks will be available for all camera systems/interfaces. The callback descriptions below indicate which callbacks are available on which interfaces.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_deregister_callback(0)`

NOTES

Callback Descriptions:

- **PL_CALLBACK_BOF**: Called when data arrives corresponding to the beginning of frame readout. This can be used as a trigger to move filter wheels, stages, etc., as depending on the clearing mode, the camera should not be exposing. This is a potentially high-frequency event; long duration processing should not be done directly in this callback, but queued for processing in another thread instead. Taking too long to process a BOF or EOF event could result in missing subsequent events.

- **PL_CALLBACK_EOF**: Called when data arrives corresponding to the end of the frame, usually indicating the beginning of exposure. This is also a potentially high-frequency event; see **PL_CALLBACK_BOF** above.
- **PL_CALLBACK_CHECK_CAMS**: On cameras with hot-pluggable buses (IEEE1394), this indicates that there is a potential for cameras to have been added to the bus. The application can use this as an indication that it should close PVCAM, re-open it, and look for new cameras.
- **PL_CALLBACK_CAM_REMOVED**: This callback is called when a hot-pluggable camera has been removed from the system, and is an indication that the camera should be closed.
- **PL_CALLBACK_CAM_RESUMED**: On camera systems supporting suspend/resume, and for camera systems with hot-pluggable buses, this indicates that the system has come back from a low-power state. If your camera is not self-powered, it probably lost power and therefore any settings that your application may have sent it. For those camera systems, this is an indication that the application should re-initialize the system.

PVCAM **Class 0: Camera Communication** **pl_cam_register_callback_ex(0)**

NAME

`pl_cam_register_callback_ex` – installs a function that will be called when an event occurs in a camera system with context.

SYNOPSIS

```
rs_bool
   pl_cam_register_callback_ex(
      int16 hcam,
      PL_CALLBACK_EVENT event,
      void *Callback,
      void *Context
      )
```

DESCRIPTION

Use this API call to install a function that will be called when the specified event occurs with respect to the camera system indicated supplying a context that will be echoed back when the callback is invoked.

The **hcam** parameter must reference an open camera system.

The **event** parameter must be one of the following:

PL_CALLBACK_BOF
PL_CALLBACK_EOF
PL_CALLBACK_CAM_REMOVED

The Callback function must be a function taking void pointer and returning no value. The contents of the context are whatever the application requires, but should be reference to the camera handle. For example:

```
void BOFCallback (void *Context)
{
     if (*(int16 *)(Context) == hCamera1)
        BOFCountCamera1++;
     else if (*(int16 *)(Context) == hCamera2)
        BOFCountCamera2++;
     return;
}
```

WARNING

`pl_exp_finish_seq` must be called if acquiring in sequential mode (using `pl_exp_setup_seq` and `pl_exp_start_seq)` with callbacks notification after a frame is read out and before new exposure is started by calling `pl_exp_start_seq`.

Not all callbacks will be available for all camera systems/interfaces. The callback descriptions below indicate which callbacks are available on which interfaces.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_deregister_callback(0)`

NOTES

Callback Descriptions:

- **PL_CALLBACK_BOF**: Called when data arrives corresponding to the beginning of frame readout. This can be used as a trigger to move filter wheels, stages, etc., as depending on the clearing mode, the camera should not be exposing. This is a potentially high-frequency event; long duration processing should not be done directly in this callback, but queued for processing in another thread instead. Taking too long to process a BOF or EOF event could result in missing subsequent events.
- **PL_CALLBACK_EOF**: Called when data arrives corresponding to the end of the frame, usually indicating the beginning of exposure. This is also a potentially high-frequency event; see **PL_CALLBACK_BOF** above.
- **PL_CALLBACK_CAM_REMOVED**: This callback is called when a hot-pluggable camera has been removed from the system, and is an indication that the camera should be closed.

PVCAM **Class 0: Camera Communication** **pl_cam_register_callback_ex2(0)**

NAME

`pl_cam_register_callback_ex2` – installs a function that will be called when an event occurs in a camera providing information about frame via FRAME_INFO type.

SYNOPSIS

```
rs_bool
   pl_cam_register_callback_ex2(
      int16 hcam,
      PL_CALLBACK_EVENT event,
      void *Callback,
      )
```

DESCRIPTION

Use this API call to install a function that will be called when the specified event occurs providing additional frame information. Input parameter of the callback function must be of PFRAME_INFO type in order to receive information about the frame (timestamp with precision of 0.1ms, frame counter number, ID (handle) of the camera that produced the frame).

The **hcam** parameter must reference an open camera system.

The **event** parameter must be one of the following:

PL_CALLBACK_BOF
PL_CALLBACK_EOF
PL_CALLBACK_CAM_REMOVED

The Callback function must be a function taking PFRAME_INFO and returning no value. For example:

```
void EOFCallbackHandler (PFRAME_INFO pNewFrameInfo)
{
     int32 frameNr = pNewFrameInfo->FrameNr;
     long64 frameTime = pNewFrameInfo->TimeStamp;
     int16 camID = pNewFrameInfo->hCam;
     //display or process frame info etc…
     return;
}
```

WARNING

`pl_exp_finish_seq` must be called if acquiring in sequential mode (using `pl_exp_setup_seq` and `pl_exp_start_seq)` with callbacks notification after a frame is read out and before new exposure is started by calling `pl_exp_start_seq`.

Not all callbacks will be available for all camera systems/interfaces. The callback descriptions below indicate which callbacks are available on which interfaces.

Variable pointed to by *`pFrameInfo`* must be created with `pl_create_frame_info_struct(2)`.

RETURN VALUE TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO `pl_cam_deregister_callback(0)`

NOTES

Callback Descriptions:

- **PL_CALLBACK_BOF**: Called when data arrives corresponding to the beginning of frame readout. This can be used as a trigger to move filter wheels, stages, etc., as depending on the clearing mode, the camera should not be exposing. This is a potentially high-frequency event; long duration processing should not be done directly in this callback, but queued for processing in another thread instead. Taking too long to process a BOF or EOF event could result in missing subsequent events.
- **PL_CALLBACK_EOF**: Called when data arrives corresponding to the end of the frame, usually indicating the beginning of exposure. This is also a potentially high-frequency event; see **PL_CALLBACK_BOF** above.
- **PL_CALLBACK_CAM_REMOVED**: This callback is called when a hot-pluggable camera has been removed from the system, and is an indication that the camera should be closed.

PVCAM **Class 0: Camera Communication** **pl_cam_register_callback_ex3(0)**

NAME

`pl_cam_register_callback_ex3` – installs a function that will be called when an event occurs in a camera providing information about frame via FRAME_INFO type and with user context information. This function combines functionality provided by `pl_cam_register_callback_ex` and `pl_cam_register_callback_ex2`

SYNOPSIS

```
rs_bool
   pl_cam_register_callback_ex3(
      int16 hcam,
      PL_CALLBACK_EVENT event,
      void *Callback, void *Context
      )
```

DESCRIPTION

Use this API call to install a function that will be called when the specified event occurs providing additional frame information. Input parameter of the callback function must be of PFRAME_INFO type in order to receive information about the frame (timestamp with precision of 0.1ms, frame counter number, ID (handle) of the camera that produced the frame). Also pointer to a context that will be echoed back when the callback is invoked can be passed to PVCAM in this function.

The **hcam** parameter must reference an open camera system.

The **event** parameter must be one of the following:

PL_CALLBACK_BOF
PL_CALLBACK_EOF
PL_CALLBACK_CAM_REMOVED

The Callback function must be a function taking PFRAME_INFO and void pointer and returning no value. For example:

```
void EOFCallbackHandler (PFRAME_INFO pNewFrameInfo,
                         void * Context)
{
     int32 frameNr = pNewFrameInfo->FrameNr;
     long64 frameTime = pNewFrameInfo->TimeStamp;
     int16 camID = pNewFrameInfo->hCam;
     //display or process frame info etc…
     if (*(int16 *)(Context) == hCamera1)
        EOFCountCamera1++;
     else if (*(int16 *)(Context) == hCamera2)
        EOFCountCamera2++;
     return;
}
```

WARNING

`pl_exp_finish_seq` must be called if acquiring in sequential mode (using `pl_exp_setup_seq` and `pl_exp_start_seq)` with callbacks notification after a frame is read out and before new exposure is started by calling `pl_exp_start_seq`.

Not all callbacks will be available for all camera systems/interfaces. The callback descriptions below indicate which callbacks are available on which interfaces.

Variable pointed to by *`pFrameInfo`* must be created with `pl_create_frame_info_struct(2)`.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_cam_deregister_callback(0)`

NOTES

Callback Descriptions:

- **PL_CALLBACK_BOF**: Called when data arrives corresponding to the beginning of frame readout. This can be used as a trigger to move filter wheels, stages, etc., as depending on the clearing mode, the camera should not be exposing. This is a potentially high-frequency event; long duration processing should not be done directly in this callback, but queued for processing in another thread instead. Taking too long to process a BOF or EOF event could result in missing subsequent events.
- **PL_CALLBACK_EOF**: Called when data arrives corresponding to the end of the frame, usually indicating the beginning of exposure. This is also a potentially high-frequency event; see **PL_CALLBACK_BOF** above.
- **PL_CALLBACK_CAM_REMOVED**: This callback is called when a hot-pluggable camera has been removed from the system, and is an indication that the camera should be closed.

PVCAM	**Class 0: Camera Communication** **pl_cam_deregister_callback(0)**
NAME	`pl_cam_deregister_callback` – uninstalls a function for camera system event
SYNOPSIS	**`rs_bool`** **`pl_cam_deregister_callback(`** **`int16 hcam,`** **`PL_CALLBACK_EVENT event,`** **`)`**
DESCRIPTION	Use this API call to uninstall a function for the specified camera system event. The **hcam** parameter must reference an open camera system. The **event** parameter must be one of the following: PL_CALLBACK_BOF PL_CALLBACK_EOF PL_CALLBACK_CAM_REMOVED
RETURN VALUE	TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO	`pl_cam_register_callback(0),`
	`pl_cam_register_callback_ex(0),`
	`pl_cam_register_callback_ex2(0),`
	`pl_cam_register_callback_ex3(0)`

Class 0 Parameter IDs

The following parameter IDs are used with `pl_get_param`, `pl_set_param`, `pl_get_enum_param`, and `pl_enum_str_length` functions described in Chapter 5.

Note: Before trying to use or retrieve more information about a parameter, it is always recommended to call an ATTR_AVAIL to see if the system supports it.

Class 0 Parameter ID	Description
PARAM_DD_INFO **Camera Dependent**	Returns an information message for each device. Some devices have no message. The user is responsible for allocating enough memory to hold the message string (`PARAM_DD_INFO_LENGTH`). Datatype: **char_ptr**
PARAM_DD_INFO_LENGTH **Camera Dependent**	Returns the length of an information message for each device. Some devices have no message. In other words, they return a value of 0 for bytes. Datatype: **int16**
PARAM_DD_RETRIES **Camera Dependent**	Reads/sets the maximum number of command retransmission attempts that are allowed. When a command or status transmission is garbled, the system signals for a retransmission. After a certain number of failed transmissions (an initial attempt + max_retries), the system abandons the attempt and concludes that the communications link has failed. The camera won't close, but the command or status read returns with an error. The maximum number of retries is initially set by the device driver, and is matched to the communications link, hardware platform, and operating system. It may also be reset by the user. Datatype: **uns16**
PARAM_DD_TIMEOUT **Camera Dependent**	Reads/sets the maximum time the driver waits for acknowledgment (i.e., the slowest allowable response speed from the camera). This is a crucial factor used in the device driver for communications control. If the driver sends a command to the camera and doesn't receive acknowledgment within the timeout period, the driver times out and returns an error. Unless reset by the user, this timeout is a default setting that is contained in the device driver and is matched to the communications link, hardware platform, and operating system. Datatype: **uns16**

Class 0 Parameter ID	Description
PARAM_DD_VERSION	Returns a version number for the device driver used to access the camera `hcam`. The version is a formatted hexadecimal number, of the style: high byte low byte ---------- ---------- ---------- hi nibble low nibble major version minor version trivial version For example, the number 0xB1C0 indicates major release 177, minor release 12, and trivial change 0. A major release is defined as anything that alters the user interface, calling sequence, or parameter interpretation of any device driver interface function (anything that would alter the driver's API). A new major release often requires the calling software to change, but wherever possible, major releases are backward compatible with earlier releases. A minor release should be completely transparent to higher level software, but may include internal enhancements. A trivial change is reserved for use by the software staff to keep track of extremely minor variations. The last digit may also be used to flag versions of the driver constructed for unique customers or situations. Minor and trivial releases should require no change in the calling software. Open the camera before calling this parameter. Note that different cameras on the same system may use different drivers. Thus, each camera can have its own driver, and its own driver version. Datatype: **uns16**

This page intentionally left blank.

Chapter 4:
Error Reporting (Class 1)

Introduction

Virtually every PVCAM function resets the error code to 0 (no error). This means that pl_error_code only reports the error status of the most recent function used. Since all PVCAM functions universally return a TRUE for no error/success, and a FALSE for a failure, you can use the following construction to report errors:

```
char msg[ERROR_MSG_LEN];
if (! pl_pvcam_do_something(. . .) ) {
pl_error_message ( pl_errror_code(),msg ) ;
printf("pvcam_do_thing failed with message '%s'/n",msg):
}
```

If you need to check whether the function works before executing further code, you could use the sample construction below:

```
if(pl_pvcam_do_something(. . .) ) { /* function succeeded */
. . . code . . .
}
else {               /* function failed, print msg*/
pl_error_message(pl_error_code(),msg);
printf("pvcam_do_thing failed with message'%s'/n",msg):
}
```

Although the (`function==TRUE`) style works well in many cases, you may prefer a more explanatory comparison. In that case, the following two constants are defined for your use:

```
#define PV_OK TRUE
#define PV_FAIL FALSE
```

Using these two constants, the code above can be rewritten as follows:

```
if(pvcam_do_thing()==PV_OK){ /*func succeeded */
. . .
```

or

```
if(pvcam_do_thing()==PV_FAIL){/*func failed, print msg*/
. . .
```

Use any of the styles illustrated above in any mix. The differences are only a matter of stylistic preference.

Error Codes

All successful functions reset `pl_error_code` to 0, which produces the message "`No error`". All unsuccessful functions return a numeric value, where that value corresponds to a number linked to a published list of error code messages.

List of Available Class 1 Functions

Class 1 Error Code functions are listed below:

```
pl_error_code
pl_error_message
```

Class 1 Functions

PVCAM	**Class 1: Error Reporting** **pl_error_code(1)**
NAME	`pl_error_code` – returns the most recent error condition.
SYNOPSIS	**`int16`** **`pl_error_code(void)`**
DESCRIPTION	As every PVCAM function begins, it resets the error code to 0. If an error occurs later in the function, the error code is set to a corresponding value.
RETURN VALUE	The current error code. Note that a call to `pl_error_code` does not reset the error code.
SEE ALSO	`pl_error_message(1)`
NOTES	`pl_error_code` works even before `pl_pvcam_init` is called. This allows a message to be returned if `pl_pvcam_init` fails. In the error codes structure, the thousands digit indicates the class of the failed function.
KNOWN BUGS	The PVCAM library does not intercept signals. Errors that interrupt the normal process (divide by zero, etc.) may cause the software to crash, and `pl_error_code` may or may not contain useful information.

PVCAM **Class 1: Error Reporting** **pl_error_message(1)**

NAME

`pl_error_message` – returns a string explaining input error code.

SYNOPSIS

```
rs_bool
   pl_error_message(int16 err_code,char_ptr msg)
```

DESCRIPTION

This function fills in the character string *`msg`* with a message that corresponds to the value in *`err_code`*. The msg string is allocated by the user, and should be at least `ERROR_MSG_LEN` elements long.

RETURN VALUE

TRUE if a message is found corresponding to the input code, FALSE if the code is out of range or does not have a corresponding message (*`msg`* will be filled with the string "`unknown error`"). Even if a FALSE is returned, the value of `pl_error_code` is not altered.

SEE ALSO

`pl_error_code(1)`

NOTES

`pl_error_message` works even before `pl_pvcam_init` is called. This allows a message to be printed if `pl_pvcam_init` fails.

Most error messages are lower case sentence fragments with no ending period.

Chapter 5: Configuration / Setup (Class 2)

Note: `pl_pvcam_init` must be called before any other function in the library! Until it is called, all functions will fail and return a FALSE. `pl_pvcam_init` is necessary, even if no hardware interaction is going to occur.

Introduction

The basic idea of Get/Set functions is to determine if a feature exists in a camera set, what its attributes are, and how can it be changed (if at all). The main function is `pl_get_param`. This function is called with a parameter id (*`param_id`*) and an attribute (*`param_attrib`*) and returns the attribute for that parameter. Usually, the user would start off with *`ATTR_AVAIL`*, which checks to see if the *`param_id`* is supported in the software and hardware. If FALSE is returned in the *`param_value`*, the *`param_id`* is not supported in either the software or the hardware. If TRUE is returned, the *`param_id`* is supported and the user can get the access rights (*`ATTR_ACCESS`*).

`ATTR_ACCESS` tells if the *`param_id`* can be written to or read or, if it cannot be written to or read, tells whether a feature is possible. If the parameter can be either written to or read the next step is to determine its data type.

Data type determination can be done by calling the parameter id with the attribute of data type (*`ATTR_TYPE`*), this will report the data type: string (*`TYPE_CHAR_PTR`*), integer (*`TYPE_INT8, TYPE_UNS8, TYPE_INT16, TYPE_UNS16, TYPE_INT32, TYPE_UNS32`*), floating point (*`TYPE_FLT64`*), boolean (*`TYPE_BOOLEAN`*), or an enumerated type (*`TYPE_ENUM`*). The user can then get the current value (*`ATTR_CURRENT`*) and the default value (*`ATTR_DEFAULT`*) for the parameter id. If the data type is not the enumerated type, the user can also get the minimum value (*`ATTR_MIN`*), the maximum value (*`ATTR_MAX`*), and the increment (*`ATTR_INCREMENT`*). Finally, if the data type is enumerated, the user can get the number of enumerated types that are legal (*`ATTR_COUNT`*), and passing the parameter id and index (which has to be between 0 and less than *`ATTR_COUNT`*), the user can call `pl_get_enum_param` and get the exact enumerated value along with a string that describes the enumerated type.

Notes:

- *`hcam`* specifies which camera and which device driver are being used. `hcam` must be a valid camera handle.
- If the data type coming back from `ATTR_TYPE` is `TYPE_CHAR_PTR` (and not an enumerated type), then the `ATTR_COUNT` is the number of characters in the string plus a `NULL` terminator.

List of Available Class 2 Functions

Class 2 functions represent camera settings. The current Class 2 functions are listed below according to their respective types and are further described in the "Class 2 Functions" section, starting on page 64. Although these functions have been superseded by `pl_get_param` and `pl_set_param` parameter ids, the list of these functions and their descriptions have been included for reference purposes.

Camera Settings

`pl_get_param`

`pl_set_param`

`pl_get_enum_param`

`pl_enum_str_length`

`pl_pp_reset`

`pl_create_frame_info_struct`

`pl_release_frame_info_struct`

`pl_create_smart_stream_struct`

`pl_release_smart_stream_struct`

List of Available Class 2 Parameter IDs

The following are available Class 2 parameters used with `pl_get_param()`, `pl_set_param()`, `pl_get_enum_param()`, and `pl_enum_str_length()` functions specified in Chapter 5.

CCD Clearing

`PARAM_CLEAR_CYCLES`

`PARAM_CLEAR_MODE`

Temperature Control

`PARAM_COOLING_MODE`

`PARAM_TEMP`

`PARAM_TEMP_SETPOINT`

CCD Physical Attributes

`PARAM_COLOR_MODE`

`PARAM_FWELL_CAPACITY`

`PARAM_PAR_SIZE`

`PARAM_PIX_PAR_DIST`

`PARAM_PIX_PAR_SIZE`

`PARAM_PIX_SER_DIST`

`PARAM_PIX_SER_SIZE`

`PARAM_POSTMASK`

`PARAM_POSTSCAN`

`PARAM_PIX_TIME`

`PARAM_PREMASK`

`PARAM_PRESCAN`

`PARAM_SER_SIZE`

`PARAM_SUMMING_WELL`

Gain

PARAM_GAIN_INDEX
PARAM_GAIN_MULT_ENABLE
PARAM_GAIN_MULT_FACTOR
PARAM_PREAMP_DELAY
PARAM_PREAMP_OFF_CONTROL

Shutter

PARAM_EXPOSURE_MODE
PARAM_PREFLASH
PARAM_SHTR_CLOSE_DELAY
PARAM_SHTR_OPEN_DELAY
PARAM_SHTR_OPEN_MODE
PARAM_SHTR_STATUS

I/O

PARAM_IO_ADDR
PARAM_IO_BITDEPTH
PARAM_IO_DIRECTION
PARAM_IO_STATE
PARAM_IO_TYPE
PARAM_LOGIC_OUTPUT

Post-Processing

PARAM_ACTUAL_GAIN
PARAM_READ_NOISE
PARAM_PP_INDEX
PARAM_PP_FEAT_NAME
PARAM_PP_PARAM_INDEX
PARAM_PP_PARAM_NAME
PARAM_PP_PARAM

CCD Readout

PARAM_CCS_STATUS
PARAM_PMODE
PARAM_READOUT_PORT
PARAM_READOUT_TIME
PARAM_EXPOSE_OUT_MODE

ADC Attributes

PARAM_ADC_OFFSET
PARAM_BIT_DEPTH
PARAM_SPDTAB_INDEX

Capabilities

PARAM_ACCUM_CAPABLE
PARAM_FRAME_CAPABLE
PARAM_MPP_CAPABLE

S.M.A.R.T Streaming

PARAM_SMART_STREAM_MODE_ENABLED
PARAM_SMART_STREAM_MODE
PARAM_SMART_STREAM_EXP_PARAMS

Other

PARAM_CAM_FW_VERSION
PARAM_CHIP_NAME
PARAM_HEAD_SER_NUM_ALPHA
PARAM_PCI_FW_VERSION
PARAM_SERIAL_NUM

Class 2 Functions

PVCAM **Class 2: Configuration/Setup** **pl_get_param(2)**

NAME

`pl_get_param` – returns the requested attribute for a PVCAM parameter.

SYNOPSIS

```
rs_bool
   pl_get_param (int16 hcam,uns32 param_id,int16 param_attrib,
                 void_ptr param_value)
```

DESCRIPTION

This function returns the requested attribute for a PVCAM parameter.

`param_id` is an enumerated type that indicates the parameter in question. See *"*Class 0 Parameter IDs*"*, *"Class 2 Parameter IDs"*, and *"Class 3 Parameter IDs"* for information about valid parameter ids.

`param_value` points to the value of the requested attribute for the parameter. It is a *`void_ptr`* because it can be different data types: the user is responsible for passing in the correct data type (see attribute descriptions that follow).

`param_attrib` is used to retrieve characteristics of the parameter. Possible values for *`param_attrib`* are:

ATTR_ACCESS	ATTR_INCREMENT
ATTR_AVAIL	ATTR_MAX
ATTR_COUNT	ATTR_MIN
ATTR_CURRENT	ATTR_TYPE
ATTR_DEFAULT	

ATTR_ACCESS

Reports if the `param_id` can be written to and/or read or (if it cannot be written to and/or read) tells whether a feature exists. If the `param_id` can be either written to or read the next step is to determine its data type.

The access types are enumerated:

ACC_ERROR	ACC_EXIST_CHECK_ONLY
ACC_READ_ONLY	ACC_WRITE_ONLY
ACC_READ_WRITE	

The data type for this attribute is `TYPE_UNS16`.

Note: This is an exception where an enum type is not treated as an unsigned 32-bit integral value

ATTR_AVAIL

Feature available with attached hardware and software. The data type for this attribute is `TYPE_BOOLEAN`.

PVCAM **Class 2: Configuration/Setup** **pl_get_param(2)**

ATTR_COUNT

Number of possible values for enumerated and/or array data types. If the data type returned by ATTR_TYPE is TYPE_CHAR_PTR (and not an enumerated type), then the ATTR_COUNT is the number of characters in the string plus a NULL terminator. If 0 or 1 is returned, ATTR_COUNT is a scalar (single element) of the following data types: TYPE_INT8, TYPE_UNS8, TYPE_INT16, TYPE_UNS16, TYPE_INT32, TYPE_UNS32, TYPE_FLT64, TYPE_BOOLEAN.

The data type for ATTR_COUNT is TYPE_UNS32.

ATTR_CURRENT

Current value. The data type for this attribute is defined by ATTR_TYPE.

ATTR_DEFAULT

Default value. The data type for this attribute is defined by ATTR_TYPE.

ATTR_INCREMENT

Step size for values (zero if non-linear or has no increment). The data type for this attribute is defined by ATTR_TYPE.

ATTR_MAX

Maximum value. The data type for this attribute is defined by ATTR_TYPE.

ATTR_MIN

Minimum value. The data type for this attribute is defined by ATTR_TYPE.

ATTR_TYPE

Data type of parameter (int16, float 64, enumerated, etc.). The data type for this is TYPE_UNS16. If the data type coming back from ATTR_TYPE is TYPE_CHAR_PTR (and not an enumerated type), then the ATTR_COUNT is the number of characters in the string plus a NULL terminator.

Data type used by pl_get_param with attribute type (ATTR_TYPE).

TYPE_CHAR_PTR	string
TYPE_INT8	
TYPE_UNS8	
TYPE_INT16	
TYPE_UNS16	
TYPE_INT32	
TYPE_UNS32	
TYPE_FLT64	
TYPE_ENUM	treat as uns32
TYPE_BOOLEAN	
TYPE_VOID_PTR	ptr to void
TYPE_VOID_PTR_PTR	ptr to a void ptr

RETURN VALUE TRUE for success, FALSE for a failure. Failure sets pl_error_code.

SEE ALSO pl_set_param and pl_get_enum_param

NOTES The data type of *param_value* is documented in PVCAM.H for each *param_id*. It can be retrieved using the *pl_get_param* function, with the *ATTR_TYPE* attribute.

PVCAM **Class 2: Configuration/Setup** **pl_set_param(2)**

NAME

`pl_set_param` – sets the current value for a PVCAM parameter.

SYNOPSIS

```
rs_bool
   pl_set_param(int16 hcam,uns32 param_id,void_ptr
                param_value)
```

DESCRIPTION

This function sets the current value for a PVCAM parameter.

`param_id` is an enumerated type that indicates the parameter in question. See *"*Class 0 Parameter IDs*"*, *"Class 2 Parameter IDs"*, and *"Class 3 Parameter IDs"* for information about valid parameter ids.

`param_value` points to the new value of the parameter.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_get_param(2)`

NOTES

The data type of *`param_value`* is documented in `PVCAM.H` for each *`param_id`*. It can be retrieved using the `pl_get_param` function, using the *`ATTR_TYPE`* attribute.

The user should call the `pl_get_param` function with the attribute *`ATTR_ACCESS`*, to verify that the parameter id is writeable (settable), before calling the `pl_set_param` function.

PVCAM **Class 2: Configuration/Setup** **pl_get_enum_param(2)**

NAME

`pl_get_enum_param` – returns the enumerated value of the parameter *`param_id`* at *`index`*.

SYNOPSIS

```
rs_bool
   pl_get_enum_param (int16 hcam,uns32 param_id,uns32
                      index,int32_ptr value,char_ptr
                      desc,uns32 length)
```

DESCRIPTION

This function will return the enumerated value of the parameter *`param_id`* at *`index`*. It also returns a string associated with the enumerated type (*`desc`*). *`length`* indicates the maximum length allowed for the returned description. See "Class 0 Parameter IDs", *"Class 2 Parameter IDs"*, and *"Class 3 Parameter IDs"* for information about valid parameter ids.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_get_param, pl_set_param, and pl_enum_str_length`

NOTES

The user should call the `pl_get_param` function with the attribute *`ATTR_TYPE`*, to verify that the parameter id is an enumerated data type before calling the `pl_get_enum_param`. The user should also call the `pl_get_param` function with the attribute *`ATTR_COUNT`* to determine how many valid enumerated values the parameter id has.

Example: Suppose there is a parameter for camera readout speed. This parameter can be set to 1MHz, 5MHz, or 10MHz. If the readout speed is currently set to 5MHz, a call to *`pl_get_param`* returns a value of 1. A call to *`pl_get_enum_param`* for the readout speed parameter at `index` 1 returns the enumerated type *`5MHz`* (which may or may not be equal to 1). The *`desc`* would contain "5Mhz".

PVCAM **Class 2: Configuration/Setup** **pl_enum_str_length(2)**

NAME

`pl_enum_str_length` – returns the length of the descriptive string for the parameter *param_id* at index.

SYNOPSIS

```
rs_bool
   pl_enum_str_length(int16 hcam,uns32 param_id,uns32 index,
                      uns32_ptr length)
```

DESCRIPTION

This function will return the length (length) of the descriptive string for the parameter *param_id* at index. The length includes the terminating null ("\0") character.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_get_enum_param`

NOTES

This function can be used to determine the amount of memory to allocate for the descriptive string when calling the `pl_get_enum_param` function. Using the example in `pl_get_enum_param`, the length returned would be 5 (4 printable characters plus 1 null character).

PVCAM	**Class 2: Configuration/Setup**	**pl_pp_reset**
NAME	`pl_pp_reset` – fails if post-processing modules are not available in current camera or if *hcam* is not the handle of an open camera.	
SYNOPSIS	**`rs_bool`** **`pl_pp_reset(int16hcam)`**	
DESCRIPTION	This function will reset all post-processing modles to their default values.	
RETURN VALUE	TRUE for a successful reset, FALSE for an unsuccessful reset.	

SEE ALSO	`PARAM_PP_FEAT_NAME`
	`PARAM_PP_PARAM_INDEX`
	`PARAM_PP_PARAM_NAME`
	`PARAM_PP_PARAM`	

PVCAM	**Class 2: Configuration/Setup**	**pl_create_frame_info_struct(2)**

NAME

`pl_create_frame_info_struct` – creates and allocates variable of FRAME_INFO type and returns pointer to it.

SYNOPSIS

```
rs_bool
   pl_create_frame_info_struct(PFRAME_INFO * pNewFrameInfo)
```

DESCRIPTION

This function will create a variable of FRAME_INFO type and return a pointer to access it. The GUID field of the FRAME_INFO structure is assigned by this function.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_release_frame_info_struct`

`pl_exp_get_latest_frame_ex`

`pl_exp_get_oldest_frame_ex`

`pl_exp_check_cont_status_ex`

`pl_cam_register_callback_ex2`

`pl_cam_register_callback_ex3`

NOTES

This function is used to create a variable of FRAME_INFO type. GUID filed of the FRAME_INFO structure is assigned by this function. Other fields are updated by PVCAM at the time of frame reception.

PVCAM	**Class 2: Configuration/Setup** **pl_release_frame_info_struct(2)**
NAME	`pl_release_frame_info_struct` – deletes variable of FRAME_INFO type.
SYNOPSIS	**`rs_bool`** **`pl_release_frame_info_struct(PFRAME_INFO pFrameInfoToDel)`**
DESCRIPTION	This function will deallocate FRAME_INFO variable created by `pl_create_frame_info_struct`.
RETURN VALUE	TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO	`pl_create_frame_info_struct`
	`pl_exp_get_latest_frame_ex`
	`pl_exp_get_oldest_frame_ex`
	`pl_exp_check_cont_status_ex`
	`pl_cam_register_callback_ex2`
	`pl_cam_register_callback_ex3`

NOTES	This function is used to deallocate a variable of FRAME_INFO type if the GUID field of the variable matches the value assigned by `pl_create_frame_info_struct`.

PVCAM	**Class 2: Configuration/Setup** **pl_create_smart_stream_struct(2)**
NAME	`pl_create_smart_stream_struct` – creates and allocates variable of `smart_stream_type` type with the number of entries passed in via the `entries` parameter and returns pointer to it.
SYNOPSIS	**`rs_bool pl_create_smart_stream_struct(smart_stream_type_ptr *pSmtStruct, uns16 entries)`**
DESCRIPTION	This function will create a variable of `smart_stream_type` type and return a pointer to access it. The `entries` parameter passed by the user determines how many entries the structure will contain.
RETURN VALUE	TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.
SEE ALSO	`pl_release_smart_stream_struct`
NOTES	

PVCAM	**Class 2: Configuration/Setup** **pl_release_smart_stream_struct(2)**
NAME	`pl_release_smart_stream_struct` – frees the space previously allocated by the `pl_create_smart_stream_struct` function.
SYNOPSIS	**`rs_bool pl_release_smart_stream_struct(smart_stream_type_ptr *pSmtStruct)`**
DESCRIPTION	This function will deallocate a `smart_stream_type_ptr` variable created by `pl_create_smart_stream_struct`.
RETURN VALUE	TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.
SEE ALSO	`pl_create_smart_stream_struct`
NOTES	This function is used to deallocate a variable of `smart_stream_type`.

Class 2 Parameter IDs

The following parameter IDs are used with `pl_get_param`, `pl_set_param`, `pl_get_enum_param`, and `pl_enum_str_length` functions described in Chapter 5.

Note: Before trying to use or retrieve more information about a parameter, it is always recommended to call an ATTR_AVAIL to see if the system supports it.

Class 2 Parameter ID	Description
PARAM_ACCUM_CAPABLE **Camera Dependent**	Returns TRUE if the camera has accumulation capability. Accumulation functionality is provided with the Class 93 FF plug-in. Datatype: **rs_bool**
PARAM_ADC_OFFSET **Camera Dependent**	Bias offset voltage. The units do not correspond to the output pixel values in any simple fashion (the conversion rate should be linear, but may differ from system to system) but a lower offset voltage will yield a lower value for all output pixels. Pixels brought below zero by this method will be clipped at zero. Pixels raised above saturation will be clipped at saturation. Before you can change the offset level, you must read the current offset level. The default offset level will also vary from system to system and may change with each speed and gain setting. ***Note:*** THIS VALUE IS SET AT THE FACTORY AND SHOULD NOT BE CHANGED. If you would like to change this value, please contact customer service before doing so. Datatype: **int16**
PARAM_BIT_DEPTH	Number of bits output by the currently selected speed choice. Although this number might range between 6 and 16, the data will always be returned in an unsigned 16-bit word. This value indicates the number of valid bits within that word. Datatype: **int16**
PARAM_CAM_FW_VERSION **Camera Dependent**	Returns the firmware version of the camera, as a hexadecimal number in the form MMmm, where MM is the major version and mm is the minor version. For example, 0x0814 corresponds to version 8.20. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_CCS_STATUS **Camera Dependent**	This holds sixteen bits of status data from the Camera Control Subsystem (CCS). Only the lowest 2 bits are currently implemented. These 2 bits give the status of the CCS: Value CCS State 0idle 1initializing 2running 3continuously clearing A running state occurs any time the CCS is in the process of performing a camera operation (including opening or closing the shutter, exposing, clearing the CCD before a sequence or exposure, parallel or serial shifting, and readout/digitization). After the CCD has finished reading out, the setup determines if the CCS goes to idle or enters continuous clearing mode. Datatype: **int16**
PARAM_CHIP_NAME	The name of the CCD. The name is a null-terminated text string. The user must pass in a character array that is at least `CCD_NAME_LEN` elements long. Datatype: **char_ptr**
PARAM_CLEAR_CYCLES	This is the number of times the CCD must be cleared to completely remove charge from the parallel register. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_CLEAR_MODE **Camera Dependent**	This defines when clearing takes place. See enum below for possible values. CLEAR_NEVER CLEAR_PRE_EXPOSURE CLEAR_PRE_SEQUENCE CLEAR_POST_SEQUENCE CLEAR_PRE_POST_SEQUENCE CLEAR_PRE_EXPOSURE_POST_SEQ CLEAR_NEVER Don't ever clear the CCD. CLEAR_PRE_EXPOSURE Clear clear_cycles times before each exposure starts. CLEAR_PRE_SEQUENCE Clear clear_cycles times before the sequence starts. CLEAR_POST_SEQUENCE Do continuous clearing after the sequence ends. CLEAR_PRE_POST_SEQUENCE Clear clear_cycles times before the sequence starts and continuous clearing after the sequence ends. CLEAR_PRE_EXPOSURE_POST_SEQ Clear clear_cycles times before each exposure starts and continuous clearing after the sequence ends. The CLEAR_NEVER setting is particularly useful for performing a readout after an exposure has been aborted. Note that normally during the idle period, the CCS parallel clock drivers and serial drivers revert to a low power state. This saves on both power and heat. If any CLEAR_..._POST options are used, these systems will not enter low power mode. This will generate extra heat in both the electronics unit and the camera head. Datatype: **enum**
PARAM_COLOR_MODE **Camera Dependent**	The color mode of the CCD. See enum below for possible values. COLOR_NONE=0 COLOR_RGGB=2 COLOR_NONE = monochrome COLOR_RGGB = RGGB color mask Datatype: **enum**

Class 2 Parameter ID	Description
PARAM_COOLING_MODE	This is the type of cooling used by the current camera. See enum below for possible values. NORMAL_COOL CRYO_COOL NORMAL_COOL This is a thermo-electrically (TE)-cooled camera with air or liquid assisted cooling. CRYO_COOL The camera is cryogenically cooled. A camera cooled via Liquid Nitrogen (LN) in an attached Dewar is an example of a cryo-cooled camera. Datatype: **enum**
PARAM_EXPOSURE_MODE	This parameter cannot be set but its value can be retrieved. Possible values: TIMED_MODE STROBED_MODE BULB_MODE TRIGGER_FIRST_MODE FLASH_MODE VARIABLE_TIMED_MODE ***Note:*** See *"Exposure Mode Constants"* on page 89 for information about these modes. Datatype: **enum**
PARAM_EXPOSE_OUT_MODE **Camera Dependent**	This parameter cannot be set but its value can be retrieved Note: See "*Extended Exposure Modes*" on page 26 for information about Expose Out Modes. Datatype: **enum**
PARAM_FRAME_CAPABLE **Camera Dependent**	If true, this camera can run in frame transfer mode (set through PARAM_PMODE). Datatype: **rs_bool**
PARAM_FWELL_CAPACITY **Camera Dependent**	Gets the full-well capacity of this CCD, measured in electrons. Datatype: **uns32**

Class 2 Parameter ID	Description
PARAM_GAIN_INDEX	Gain setting for the current speed choice. The valid range for a gain setting is 1 through *PARAM_GAIN_INDEX* with *ATTR_MAX*, where the max gain may be as high as 16. Values outside this range will be ignored. Note that gain settings may not be linear! Values 1-16 may not correspond to 1x - 16x, and there are holes between the values. However, when the camera is initialized, and every time a new speed is selected, the system will always reset to run at a gain of 1x. Datatype: **int16**
PARAM_GAIN_MULT_ENABLE **Camera Dependent**	Gain multiplier on/off indicator for cameras with the multiplication gain functionality. This parameter may be read-only, in which case the gain is always on. Datatype: **rs_bool**
PARAM_GAIN_MULT_FACTOR **Camera Dependent**	Gain multiplication factor for cameras with multiplication gain functionality. The valid range is 1 through *PARAM_GAIN_MULT_FACTOR* with *ATTR_MAX*. Datatype: **uns16**
PARAM_HEAD_SER_NUM_ALPHA **Camera Dependent**	Returns the alphanumeric serial number for the camera head. The serial number for Photometrics-brand cameras has a maximum length of MAX_ALPHA_SER_NUM_LEN. Datatype: **char_ptr**
PARAM_IO_ADDR **Camera Dependent**	Sets and gets the currently active I/O address. The number of available I/O addresses can be obtained using the *ATTR_COUNT* attribute with the *PARAM_IO_ADDR* parameter ID. Datatype: **uns16**
PARAM_IO_BITDEPTH **Camera Dependent**	Gets the bit depth for the signal at the current address. The bit depth has different meanings, depending on the I/O Type: IO_TYPE_TTL The number of bits read or written at this address. IO_TYPE_DAC The number of bits written to the DAC. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_IO_DIRECTION **Camera Dependent**	Gets the direction of the signal at the current address. Possible values are: IO_DIR_INPUT IO_DIR_OUTPUT IO_DIR_INPUT_OUTPUT Datatype: **enum**
PARAM_IO_STATE **Camera Dependent**	Sets and gets the state of the currently active I/O signal. The new (when setting) or return (when getting) value has different meanings, depending on the I/O Type: IO_TYPE_TTL A bit pattern, indicating the current state (0 or 1) of each of the control lines (bit 0 indicates line 0 state, etc.). IO_TYPE_DAC The value of the desired analog output (only applies to pl_set_param). The minimum and maximum range for the signal can be obtained using the *ATTR_MIN* and *ATTR_MAX* attributes, respectively, with the *PARAM_IO_ADDR* parameter ID. When outputting signals, the state is the desired output. For example, when setting the output of a 12-bit DAC with a range of 0-5V to half-scale, the state should be 2.5 (volts), not 1024 (bits). Datatype: **flt64**
PARAM_IO_TYPE **Camera Dependent**	Gets the type of I/O available at the current address. Possible values are: IO_TYPE_TTL IO_TYPE_DAC Datatype: **enum**
PARAM_MPP_CAPABLE **Camera Dependent**	Indicates whether this CCD runs in MPP mode. The actual value returned is equal to one of four constants: Possible values. MPP_UNKNOWN MPP_ALWAYS_OFF MPP_ALWAYS_ON MPP_SELECTABLE Datatype: **enum**

Class 2 Parameter ID	Description
PARAM_PAR_SIZE	This is the parallel size of the CCD, in active rows. The full size of the parallel register is actually (`par_size` + `premask` + `postmask`). Datatype: **uns16**
PARAM_PCI_FW_VERSION **Camera Dependent**	Returns the version number of the PCI firmware. This number is a single 16-bit unsigned value. Datatype: **uns16**
PARAM_PIX_PAR_DIST	This is the center-to-center distance between pixels (in the parallel direction) measured in nanometers. This is identical to *PARAM_PIX_PAR_SIZE* if there are no interpixel dead areas. Datatype: **uns16**
PARAM_PIX_PAR_SIZE	This is the size of the active area of a pixel, in the parallel direction, measured in nanometers. Datatype: **uns16**
PARAM_PIX_SER_DIST	This is the center-to-center distance between pixels (in the serial direction), in nanometers. This is identical to *PARAM_PIX_SER_SIZE*, if there are no dead areas. Datatype: **uns16**
PARAM_PIX_SER_SIZE	This is the size of a single pixel's active area, in the serial direction, measured in nanometers. Datatype: **uns16**
PARAM_PIX_TIME	This is the actual speed for the currently selected speed choice. It returns the time for each pixel, in nanoseconds. This readout time will change as new speed choices are selected. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_PMODE	This allows the user to select the parallel clocking method. Possible values are: PMODE_NORMAL PMODE_FT PMODE_MPP PMODE_FT_MPP PMODE_ALT_NORMAL PMODE_ALT_FT PMODE_ALT_MPP PMODE_ALT_FT_MPP where FT indicates frame transfer mode, FT_MPP indicates both frame transfer and MPP mode. ALT indicates that custom parameters may be loaded. Datatype: **enum**
PARAM_POSTMASK	This is the number of masked lines at the far end of the parallel register (away from the serial register). This is the number of additional parallel shifts that need to be done after readout to clear the parallel register. Datatype: **uns16**
PARAM_POSTSCAN	This is the number of pixels to discard from the serial register after the last real data pixel. These must be read or discarded to clear the serial register. Datatype: **uns16**
PARAM_PREAMP_DELAY **Camera Dependent**	This is the number of milliseconds required for the CCD output preamp to stabilize, after it is turned on. Datatype: **uns16**
PARAM_PREAMP_OFF_CONTROL **Camera Dependent**	The exposure time limit in milliseconds above which the preamp is turned off during exposure. Datatype: **uns32**
PARAM_PREMASK	This is the number of masked lines at the near end of the parallel register, next to the serial register. 0=no mask (no normal mask). If the premask is equal to par_size, this probably indicates a frame transfer device with an ordinary mask. Accordingly, the CCD should probably be run in frame transfer mode. Datatype: **uns16**
PARAM_PRESCAN	This is the number of pixels discarded from the serial register before the first real data pixel. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_READOUT_PORT **Camera Dependent**	CCD readout port being used by the currently selected speed. Different readout ports (used for alternate speeds) flip the image in serial, parallel, or both. READOUT_PORT_MULT_GAIN READOUT_PORT_NORMAL READOUT_PORT_LOW_NOISE READOUT_PORT_HIGH_CAP Use *PARAM_READOUT_PORT* with *ATTR_COUNT* to read out the number of ports on the system. Datatype: **enum**
PARAM_READOUT_TIME **Camera Dependent**	Time it will take to read out the image from the sensor with the current camera settings, in microseconds. Settings have to be applied with pl_exp_setup_seq() or pl_exp_setup_cont() before the camera will calculate the readout time for the new settings. Datatype: **uns32**
PARAM_SER_SIZE	Defines the serial-dimension of the active area of the CCD chip. Datatype: **uns16**
PARAM_SERIAL_NUM **Camera Dependent**	This is the serial number of the camera head (not the electronics unit). Datatype: **uns16**
PARAM_SHTR_CLOSE_DELAY **Camera Dependent**	This is the shutter close delay. This is the number of milliseconds required for the shutter to close. The software default values compensate for the standard shutter that is shipped with all cameras. You only need to set this value if you are using a shutter with characteristics that differ from the standard shutter. Valid inputs are any number in the range 0 to 65535 milliseconds. Datatype: **uns16**
PARAM_SHTR_OPEN_DELAY **Camera Dependent**	This is the shutter open delay. This is the number of milliseconds required for the shutter to open. The software default values compensate for the standard shutter that is shipped with all cameras. You only need to set this value if you are using a shutter with characteristics that differ from the standard shutter. Valid inputs are any number in the range 0 to 65535 milliseconds. Datatype: **uns16**

Class 2 Parameter ID	Description
PARAM_SHTR_OPEN_MODE **Camera Dependent**	This is the shutter opening condition. See enum below for possible values. OPEN_NEVER OPEN_PRE_EXPOSURE OPEN_PRE_SEQUENCE OPEN_PRE_TRIGGER OPEN_NO_CHANGE OPEN_NEVER The shutter closes before the exposure and stays closed during the exposure. OPEN_PRE_EXPOSURE Opens each exposure. Normal mode. OPEN_PRE_SEQUENCE Opens the shutter at the start of each sequence. Useful for frame transfer and external strobe devices. OPEN_PRE_TRIGGER If using a triggered mode, this function causes the shutter to open before the external trigger is armed. If using a non-triggered mode, this function operates identical to OPEN_PRE_EXPOSURE. OPEN_NO_CHANGE Sends no signals to open or close the shutter. Useful for frame transfer when you want to open the shutter and leave it open (see pl_exp_abort). For detailed scripts, see "*Exposure Loops*" in the PVCAM introduction. Datatype: **enum**
PARAM_SHTR_STATUS **Camera Dependent**	This is the current state of the camera shutter. SHTR_FAULT SHTR_OPENING SHTR_OPEN SHTR_CLOSING SHTR_CLOSED SHTR_UNKNOWN If the shutter is run too fast, it will overheat and trigger SHTR_FAULT. The shutter electronics will disconnect until the temperature returns to a suitable range. Note that although the electronics have reset the voltages to open or close the shutter, there is a lag time for the physical mechanism to respond. See also *PARAM_SHTR_OPEN_DLY* and *PARAM_SHTR_CLOSE_DLY*. Datatype: **enum**

Class 2 Parameter ID	Description
PARAM_SPDTAB_INDEX	This selects the CCD readout speed from a table of available choices. Entries are 0-based and the range of possible values is 0 to max_entries; max_entries can be determined using *PARAM_SPDTAB_INDEX* with the *ATTR_MAX* attribute. This setting relates to other speed table values, including *PARAM_BIT_DEPTH*, *PARAM_PIX_TIME*, *PARAM_READOUT_PORT* and *PARAM_GAIN_INDEX*. After setting *PARAM_SPDTAB_INDEX*, the gain setting is always reset to a value corresponding to 1x gain. To use a different gain setting, call `pl_set_param` with *PARAM_GAIN_INDEX* after setting the speed table index. Datatype: **int16**
PARAM_SUMMING_WELL **Camera Dependent**	Checks to see if the summing well exists. When a TRUE is returned, the summing well exists. Datatype: **rs_bool**
PARAM_TEMP **Camera Dependent**	Returns the current measured temperature of the CCD in C°x 100. For example, a temperature of minus 35° would be read as -3500. Datatype: **int16**
PARAM_TEMP_SETPOINT **Camera Dependent**	Sets the desired CCD temperature in hundredths of degrees Celsius (minus 35 °C is represented as -3500). The hardware attempts to heat or cool the CCD to this temperature. The min/max allowable temperatures are given `ATTR_MIN` and `ATTR_MAX`. Settings outside this range are ignored. Note that this function only sets the desired temperature. Even if the desired temperature is in a legal range, it still may be impossible to achieve. If the ambient temperature is too high, it is difficult to get much cooling on an air-cooled camera. Datatype: **int16**
PARAM_ACTUAL_GAIN **Camera Dependent**	Gets the actual e/ADU for the current gain setting (read only). **Datatype: uns16**
PARAM_READ_NOISE **Camera Dependent**	Gets the read noise for the current speed (read only). **Datatype: uns16**

Class 2 Parameter ID	Description
PARAM_PP_INDEX **Camera Dependent**	This selects the current post-processing feature from a table of available choices. The entries are 0-based and the range of possible values is 0 to `max_entries`. `max_entries` can be determined using `PARAM_PP_INDEX` with the `ATTR_MAX` attribute. This setting relates to other post-processing tablev values, including `PARAM_PP_FEAT_NAME` and `PARAM_PP_PARAM_INDEX` **Datatype: int16**
PARAM_PP_FEAT_NAME **Camera Dependent**	This returns the name of the currently-selected post-processing feature. This is controlled by `PARAM_PP_INDEX` (read only) **Datatype: char_ptr**
PARAM_PP_PARAM_INDEX **Camera Dependent**	This selects the current post-processing parameter from a table of available choices. The entries are 0-based and the range of possible values is 0 to `max_entries`. `max_entries` can be determined using `PARAM_PP_PARAM_INDEX` with the `ATTR_MAX` attribute. This setting relates to other post-processing tablev values, including `PARAM_PP_PARAM_NAME` and `PARAM_PP_PARAM`. **Datatype: int16**
PARAM_PP_PARAM_NAME **Camera Dependent**	Gets the name of the currently-selected post-processing parameter for the currently-selected post-processing feature. (read only) **Datatype: char_ptr**
PARAM_PP_PARAM **Camera Dependent**	This sets the post-processing parameter for the currently-selected post-processing parameter in the index. **Datatype: uns32**
PARAM_SMART_STREAM_MODE **Camera Dependent**	This parameter allows the user to select between available S.MA.R.T streaming modes. Currently the only available mode is SMTMODE_ARBITRARY_ALL Datatype: **uns16**
PARAM_SMART_STREAM_MODE_ENABLED **Camera Dependent**	This parameter allows the user to retrieve or set the state of the S.M.A.R.T. streaming mode. When a TRUE is returned by the camera, S.M.A.R.T. streaming is enabled. **Datatype: rs_bool**

Class 2 Parameter ID	Description
PARAM_SMART_STREAM_EXP_PARAMS **Camera Dependent**	This parameter allows the user to set or read the current exposure parameters for S.M.A.R.T streaming. **Datatype: smart_stream_type_ptr**

Chapter 6:
Data Acquisition (Class 3)

Introduction

Class 3 defines CCD readout and specifies regions and binning factors. This class gives you complete control over exposures and exposure sequences. Camera configurations set in Class 2 must be considered when defining the functions in Class 3.

The current Class 3 functions are listed below. Although these functions have been superseded by `pl_get_param` and `pl_set_param` parameter ids, the list of these functions and their descriptions have been included for reference purposes.

List of Available Class 3 Functions

The Class 3 functions are listed below:

```
pl_exp_abort                    pl_exp_get_oldest_frame_ex
pl_exp_check_cont_status        pl_exp_setup_seq
pl_exp_check_status             pl_exp_start_cont
pl_exp_finish_seq               pl_exp_start_seq
pl_exp_get_latest_frame         pl_exp_stop_cont
pl_exp_get_oldest_frame         pl_exp_unlock_oldest_frame
pl_exp_init_seq                 pl_exp_uninit_seq
pl_exp_setup_cont               pl_io_clear_script_control
pl_exp_get_latest_frame_ex      pl_io_script_control
                                pl_exp_check_cont_status_ex
```

List of Available Class 3 Parameter IDs

The following are available Class 3 parameters used with `pl_get_param()`, `pl_set_param()`, `pl_get_enum_param()`, and `pl_enum_str_length()` functions specified in Chapter 5.

```
PARAM_BOF_EOF_CLR               PARAM_EXP_RES
PARAM_BOF_EOF_COUNT             PARAM_EXP_RES_INDEX
PARAM_BOF_EOF_ENABLE            PARAM_EXP_TIME
PARAM_EXP_MIN_TIME              PARAM_CURRENT_PVTIME
```

Defining Exposures

To define an exposure or exposure sequence, you must follow the steps below:

Define the region(s) to be collected by filling a rgn_type

Define the exposure time and mode

Configure any desired camera parameters:

- Apply the settings to the hardware by calling `pl_exp_setup_cont or pl_exp_setup_seq`
- Start the acquisition by calling `pl_exp_start_cont or pl_exp_start_seq`
- Monitor the progress of data collection by calling `pl_exp_check_cont_status or pl_exp_check_status`

Decode the multi-region pixel stream into images in a buffer by calling pl_exp_finish_seq (optional)

New Structures

To handle these tasks, a new structure is used. It is defined in the include file pvcam.h.

```
typedef struct {
uns16 s1; /*Starting pixel in the serial register          */
uns16 s2; /*Ending pixel in the serial register            */
uns16 sbin;/*Serial binning for this region               */
uns16 p1; /*Starting pixel in the parallel register       */
uns16 p2; /* Ending pixel in the parallel register        */
uns16 pbin;/* Parallel binning for this region            */
}rgn_type,
*rgn_ptr;
```

Exposure Mode Constants

The six constants below define the exposure mode:

TIMED_MODE	STROBED_MODE
VARIABLE_TIMED_MODE	BULB_MODE
TRIGGER_FIRST_MODE	FLASH_MODE

These modes describe how the exposure is controlled:

TIMED_MODE	Begins a single exposure or the first exposure of a sequence. The internal timer controls the exposure duration.
VARIABLE_TIMED_MODE	Begins a single exposure or the first exposure of a sequence. This mode ignores the exposure_time parameter in setup. Instead, you must call pl_exp_set_time to set the exposure duration before each sequence. In this mode, you can change the exposure duration between sequences, and readout in rapid succession, while maintaining the same readout parameters.
TRIGGER_FIRST_MODE	Waits for a trigger to begin a single exposure or a sequence of exposures. The exposure duration is controlled by the internal timer.
STROBED_MODE	Waits for a trigger to begin each exposure in a sequence. The exposure duration is controlled by the internal timer.
BULB_MODE	Waits for a trigger to begin each exposure in a sequence, then waits for the end of the trigger to end the exposure. This mode ignores *exposure_time* parameters in *setup*.

Note about Extended Exposure Modes:

In the past, the Exposure Mode enum values and descriptions were hard-coded in PVCAM libraries to supply slightly more information to the developer about enums that were present in PVCAM header. However, since PVCAM 3.0.1 it has been extended to not only describe those values, but also describe settings that only the camera firmware knows about, and that are not present in header files.

Because of that the newly added Exposure Modes are not defined in the PVCAM header like the constants discussed above but must be dynamically obtained from the camera in run-time.

Please refer to “*Extended Exposure Modes*” on page 26 for more details and a code example.

Class 3 Functions

PVCAM	**Class 3: Data Acquisition**	**pl_exp_finish_seq(3)**

NAME

`pl_exp_finish_seq` – finishes and cleans up after `pl_exp_start_seq`.

SYNOPSIS

```
rs_bool
   pl_exp_finish_seq(int16 hcam,void_ptr pixel_stream,int16
                     hbuf)
```

DESCRIPTION

This cleans up after an exposure started through `pl_exp_start_seq` has finished readout. If the exposure has not finished readout, this function returns with an error.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_abort(3),pl_exp_check_status(3), pl_exp_setup_seq(3),pl_exp_start_seq(3)`

NOTES

This function must also be called if acquiring in sequential mode (using `pl_exp_setup_seq` and `pl_exp_start_seq)` with callbacks notification after a frame is read out and before new exposure is started by calling `pl_exp_start_seq`.

PVCAM **Class 3: Data Acquisition** **pl_exp_get_latest_frame(3)**

NAME

`pl_exp_get_latest_frame` – returns pointer to most recent frame in circular buffer.

SYNOPSIS

```
rs_bool
   pl_exp_get_latest_frame(int16 hcam, void_ptr_ptr frame)
```

DESCRIPTION

This function returns a pointer to the most recently acquired frame in the circular buffer. *`frame`* is a pointer to the most recent frame.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, and `pl_exp_stop_cont(3)`

NOTES

If the camera in use is not able to return the latest frame for the current operating mode, this function will fail. For example, some cameras cannot return the latest frame in `CIRC_NO_OVERWRITE` mode. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM **Class 3: Data Acquisition** **pl_exp_get_latest_frame_ex(3)**

NAME

`pl_exp_get_latest_frame_ex` – returns pointer to most recent frame in circular buffer and updates values of timestamps (with precision of 0.1ms), frame counter numbers and readout time in variable of FRAME_INFO type.

SYNOPSIS

```
rs_bool
   pl_exp_get_latest_frame_ex(int16 hcam, void_ptr_ptr frame,
   PFRAME_INFO pFrameInfo)
```

DESCRIPTION

This function returns a pointer to the most recently acquired frame in the circular buffer. *`frame`* is a pointer to the most recent frame. Additionally this function updates the values in variable pointed to by *`pFrameInfo`* with the data collected at the time of frame reception by the device driver (e.g. timestamp, frame counter value).

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, and `pl_exp_stop_cont(3)`, `pl_exp_get_oldest_frame_ex(3)`, `pl_exp_check_cont_status_ex(3)`, `pl_cam_register_callback_ex2(0)`, `pl_create_frame_info_struct(2)`, `pl_release_frame_info_struct(2)`

NOTES

If the camera in use is not able to return the latest frame for the current operating mode, this function will fail. For example, some cameras cannot return the latest frame in `CIRC_NO_OVERWRITE` mode. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

Variable pointed to by *`pFrameInfo`* must be created with `pl_create_frame_info_struct(2)`.

PVCAM	**Class 3: Data Acquisition**	**pl_exp_get_oldest_frame(3)**

NAME

`pl_exp_get_oldest_frame` – locks oldest frame in circular buffer and returns pointer to that frame.

SYNOPSIS

```
rs_bool
   pl_exp_get_oldest_frame(int16 hcam, void_ptr_ptr frame)
```

DESCRIPTION

This function locks the oldest unretrieved frame in the circular buffer, and returns a pointer to that frame. *`frame`* is a pointer to the oldest unretrieved frame.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_unlock_oldest_frame(3)`, `pl_exp_stop_cont(3)`, and `pl_exp_get_oldest_frame_ex(3)`

NOTES

If the camera in use is not able to return the oldest frame for the current operating mode, this function will fail. For example, some cameras cannot return the oldest frame in `CIRC_OVERWRITE` mode. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM **Class 3: Data Acquisition** **pl_exp_get_oldest_frame_ex(3)**

NAME

`pl_exp_get_oldest_frame_ex` - locks oldest frame in circular buffer and returns pointer to that frame. Also updates frame counter value in the variable of FRAME_INFO type.

SYNOPSIS

```
rs_bool
   pl_exp_get_oldest_frame_ex(int16 hcam, void_ptr_ptr frame,
   PFRAME_INFO pFrameInfo)
```

DESCRIPTION

This function locks the oldest unretrieved frame in the circular buffer, and returns a pointer to that frame. *`frame`* is a pointer to the oldest unretrieved frame. Additionally this function updates the values in the variable pointed to by *`pFrameInfo`* with the data collected at the time of frame reception by the device driver (e.g. frame counter value).

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_unlock_oldest_frame(3)`, `pl_exp_stop_cont(3)`, `pl_exp_check_cont_status_ex(3)`, `pl_cam_register_callback_ex2(0)`, `pl_create_frame_info_struct(2)`, and `pl_release_frame_info_struct(2)`

NOTES

If the camera in use is not able to return the oldest frame for the current operating mode, this function will fail. For example, some cameras cannot return the oldest frame in `CIRC_OVERWRITE` mode. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

Variable pointed to by *`pFrameInfo`* must be created with `pl_create_frame_info_struct(2)`.

PVCAM	**Class 3: Data Acquisition**	**pl_exp_init_seq(3)**

NAME

`pl_exp_init_seq` – initializes the data collection functions.

SYNOPSIS

```
rs_bool
   pl_exp_init_seq(void)
```

DESCRIPTION

This function prepares the portion of the library associated with the exposure control for operation and must be called before any other Class 3 function.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_pvcam_init(0),pl_pvcam_uninit(0),pl_exp_uninit_seq(3)`

NOTES

You must explicitly call this function after calling `pl_pvcam_init` and before calling any other `pl_exp_` function.

PVCAM **Class 3: Data Acquisition** **pl_exp_setup_cont(3)**

NAME

`pl_exp_setup_cont` - sets circular buffer mode.

SYNOPSIS

```
rs_bool
   pl_exp_setup_cont(int16 hcam,uns16 rgn_total,rgn_const_ptr
                     rgn_array,int16 mode,uns32
                     exposure_time,uns32_ptr stream_size,
                     int16 circ_mode)
```

DESCRIPTION

This function sets the mode of operation for the circular buffer. This function uses the array of regions, exposure mode, exposure time passed in, and circular buffer mode and transmits them to the camera.

The pointer *`rgn_array`* points to *`rgn_total`* region definitions.

`mode` specifies the bitwise OR combination of the exposure mode and expose out mode. Please refer to chapter "*Extended Exposure Modes*" on page 26 for more details.

`exposure_time` specifies the exposure time in the currently selected exposure time resolution (see *`PARAM_EXP_RES`* and *`PARAM_EXP_RES_INDEX`*).

The pointer *`stream_size`* points to a variable that will be filled with number of bytes in the pixel stream.

`circ_mode` can be set to either `CIRC_OVERWRITE` or `CIRC_NO_OVERWRITE`. This function must be called before calling `pl_exp_start_cont()`.

The settings are then downloaded to the camera. If there is any problem (overlapping regions or a frame-transfer setting for a camera that lacks that capability), this function aborts and returns with a failure. `pl_error_code` indicates the definition problem.

The *`stream_size`* pointer is filled with the number of bytes of memory needed to buffer the full sequence. (It is the developer's responsibility to allocate a memory buffer for the pixel stream.)

When this function returns, the camera is ready to begin the exposure. `pl_exp_start_cont` initiates exposure and readout.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_get_oldest_frame(3)`, `pl_exp_get_latest_frame(3)`, `pl_exp_unlock_oldest_frame(3)`, and `pl_exp_stop_cont(3)`

NOTES

Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to see if the system can perform circular buffer operations. The circular buffer is passed to `pl_exp_start_cont`. The buffer is allocated by your application.

Refer to **Example 3: Circular Buffer** in "Code Examples" for two examples of code for circular buffer operation.

This page intentionally left blank.

PVCAM **Class 3: Data Acquisition** **pl_exp_setup_seq(3)**

NAME

`pl_exp_setup_seq` – prepares the camera to perform a readout.

SYNOPSIS

```
rs_bool
   pl_exp_setup_seq(int16 hcam,uns16 exp_total,uns16
                    rgn_total,rgn_const_ptr rgn_array,int16
                    mode,uns32 exposure_time,uns32_ptr
                    stream_size)
```

DESCRIPTION

This function uses the array of regions, exposure mode, and exposure time passed in and transmits them to the camera. *`exp_total`* specifies the number of images to take. The pointer *`rgn_array`* points to *`rgn_total`* region definitions, *`mode`* specifies the bitwise OR combination of exposure mode and expose out mode (see chapter "*Extended Exposure Modes*" on page 26), *`exposure_time`* specifies the exposure time in the currently selected exposure time resolution (see *`PARAM_EXP_RES`* and *`PARAM_EXP_RES_INDEX`*). The pointer *`stream_size`* points to a variable that will be filled with number of bytes in the pixel stream.

The settings are then downloaded to the camera. If there is any problem (overlapping regions or a frame-transfer setting for a camera that lacks that capability), this function aborts and returns with a failure. `pl_error_code` indicates the definition problem.

The *`stream_size`* pointer is filled with the number of bytes of memory needed to buffer the full sequence. (It is the developer's responsibility to allocate a memory buffer for the pixel stream.)

When this function returns, the camera is ready to begin the exposure. `pl_exp_start_seq` initiates exposure and readout.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure `sets pl_error_code`.

SEE ALSO

`pl_exp_abort(3),pl_exp_check_status(3), pl_exp_start_seq(3),pl_exp_finish_seq(3)`

NOTES

This function downloads new settings. After receiving the settings, the camera merely waits in an idle state. The `pl_exp_abort` command may be used to place the camera into some other state, such as continuous clearing, but this will not alter or affect the downloaded settings. Essentially, the camera is still holding the exposure sequence and waiting to start, while it clears the CCD charge.

PVCAM **Class 3: Data Acquisition** **pl_exp_start_cont(3)**

NAME

`pl_exp_start_cont` – begins continuous readout into circular buffer

SYNOPSIS

```
rs_bool
   pl_exp_start_cont(int16 hcam, void_ptr pixel_stream,uns32
                     size)
```

DESCRIPTION

This function will initiate a continuous readout from the camera into a `circ`ular buffer. *`pixel_stream`* is a pointer to the circular buffer, and *`size`* indicates the number of bytes the buffer can hold.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_get_oldest frame(3)`, `pl_exp_get_latest_frame(3)`, `pl_exp_unlock_oldest_frame(3)`, and `pl_exp_stop_cont(3)`

NOTES

If *`pixel_stream`* points to a buffer that is not an integer-multiple of the frame size for the exposure, this function will return FALSE and set an appropriate error code in `pl_error_code`. For example, a buffer size of 1000 with a frame size of 250 is OK, but a buffer size of 900 would cause a failure.

Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM **Class 3: Data Acquisition** **pl_exp_start_seq(3)**

NAME

`pl_exp_start_seq` – begins exposing, returns immediately.

SYNOPSIS

```
rs_bool
   pl_exp_start_seq(int16 hcam,void_ptr pixel_stream)
```

DESCRIPTION

This is a companion function to `pl_exp_setup_seq`. `pl_exp_setup_seq` must be called first to define the exposure and program this information into the camera. After that, `pl_exp_start_seq` may be called one or more times. Each time it is called, it starts one sequence and returns immediately (a sequence may be one or more exposures).

Progress can be monitored through `pl_exp_check_status`. The next sequence may be started as soon as the readout has finished or an abort has been performed (`pl_exp_abort`). The `hcam` parameter defines which camera is used.

The user must allocate an appropriately sized memory buffer for data collection, pointed to by *`pixel_stream`*. This buffer must be at least `stream_size` bytes, where *`stream_size`* is the value returned from `pl_exp_setup_seq`. In addition, this memory must be page-locked or similarly protected on virtual memory systems — these requirements are system specific and the responsibility of the application.

There is a special case for those users who want to use their own frame grabber (with an appropriately equipped camera). If a null pointer is passed in for *`pixel_stream`*, `pl_exp_start_seq` will assume that the user is routing the data to a frame grabber or other device under their control. Under these conditions, `pl_exp_start_seq` initiates the exposure, but does not attempt to collect incoming data.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets pl_error_code.

SEE ALSO

`pl_exp_check_status(3)`,`pl_exp_setup_seq(3`, `pl_exp_finish_seq(3)`

NOTES

Technically, this only changes the state of the CCS program. Regardless of whether the CCS is idle or continuously clearing, this forces the CCS program into the busy state. The camera settings are not altered by this command, but it does begin executing. If the CCS is idle, there is no delay and the camera will begin running immediately. If the CCS is continuously clearing, the system finishes the current parallel shift (it finishes the current single parallel row) and then begins running. This produces a delay of up to the parallel-shift time for this CCD (1–300 microseconds, depending on the CCD). If the camera has been set up with one of the `CLEAR_PRE_` clearing modes, it will also explicitly clear the CCD as its first action.

PVCAM **Class 3: Data Acquisition** **pl_exp_abort(3)**

Name

`pl_exp_abort` – stops collecting data, cleans up device driver, halts camera.

SYNOPSIS

```
rs_bool
   pl_exp_abort(int16 hcam,int16 cam_state)
```

DESCRIPTION

`pl_exp_abort` performs two functions: it stops the host device driver, and it may halt the camera (*`hcam`* specifies which camera and which device driver are being used.) Halting the camera halts *`readout`*, *`clearing`*, and all other camera activity. On the host side, data collection is controlled by a device driver. If data collection is currently enabled (the image data **active** state), this function stops collection, returns the low-level communication hardware and software to an image data **idle** state, and disables collection. In the **idle** state, any data that arrives is ignored and discarded. The **idle** state is the normal system default. On the camera side, the Camera Control Subsystem (CCS) may be in the process of collecting data, or it may be in one of several idle states (see `pl_get_param` parameter id `PARAM_CCS_STATUS`).

This function always stops the data collection software. In addition, it has the option of forcing the CCS into a new state by setting the *`cam_state`* variable to one of the following constants, which are camera dependent:

`CCS_NO_CHANGE`	Do not alter the current state of the CCS.
`CCS_HALT`	Halt all CCS activity, and put the CCS into the idle state.
`CCS_HALT_CLOSE_SHTR`	Close the shutter, then halt all CCS activity, and put the CCS into the idle state.
`CCS_CLEAR`	Put the CCS into the continuous clearing state.
`CCS_CLEAR_CLOSE_SHTR`	Close the shutter, then put the CCS into the continuous clearing state.
`CCS_OPEN_SHTR`	Open the shutter, then halt all CCS activity, and put the CCS into the idle state.
`CCS_CLEAR_OPEN_SHTR`	Open the shutter, then put the CCS into the continuous clearing state.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

Class 3 data collection functions, `pl_get_param` parameter id `PARAM_CCS_STATUS(2)`

NOTES

This may also be called outside of an exposure. It can explicitly open the shutter, close the shutter, or stop the CCS.

In the **idle** state, the system takes the least possible amount of action when image data arrives. On some systems, this involves placing the hardware in reset state, so it is inactive. On SCSI systems, the driver does not initiate any data transfers, although a buffer on the camera end may be filling up.

If the CCS is halted and the shutter is closed (`CCS_HALT_CLOSE_SHTR`), the current image remains on the CCD (although dark charge continues to accumulate). If *`clear_cycles`* is zero or the clear mode is `CLEAR_NEVER`, the image may be read off by performing a bias readout.

In frame transfer mode, you may not want to close the shutter when halting the CCS. Some frame transfer systems do not include a shutter, in which case an attempt to open or close the shutter is ignored, but does not cause an error.

PVCAM	**Class 3: Data Acquisition**	**pl_exp_stop_cont(3)**

NAME

`pl_exp_stop_cont` – stops continuous readout acquisition.

SYNOPSIS

```
rs_bool
   pl_exp_stop_cont(int16 hcam, int16 cam_state)
```

DESCRIPTION

This function halts a continuous readout acquisition into a circular buffer. *`cam_state`* defines the new state of the Camera Control Subsystem, as described in the documentation for the `pl_exp_abort()` function.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_get_oldest frame(3)`, `pl_exp_get_latest_frame(3)`, and `pl_exp_unlock_oldest_frame(3)`

NOTES

Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM **Class 3: Data Acquisition** **pl_exp_check_status(3)**

NAME

`pl_exp_check_status` – checks the status of the current exposure.

SYNOPSIS

```
rs_bool
   pl_exp_check_status(int16 hcam, int16_ptr status,
                       uns32_ptr byte_cnt)
```

DESCRIPTION

This is only useful when data collection has been set up and started, as with a call to the Class 3 functions `pl_exp_setup_seq` and `pl_exp_start_seq`. In general, Class 3 functions start an exposure then immediately return, allowing the progress to be monitored. The status gives a quick evaluation of progress. The variable status returns one of the following values:

`READOUT_NOT_ACTIVE`	The system is **idle**, no data is expected. If any arrives, it will be discarded.
`EXPOSURE_IN_PROGRESS`	The data collection routines are **active**. They are waiting for data to arrive, but none has arrived yet.
`READOUT_IN_PROGRESS`	The data collection routines are **active**. The data has started to arrive.
`READOUT_COMPLETE`	All the expected data has arrived. Data collection is complete, and the driver has returned to **idle** state.
`READOUT_FAILED`	Something went wrong. The function returns a FALSE and `pl_error_code` is set. (See Return Value below for more information.)

More detailed information is returned in *`byte_cnt`*. This reports on exactly how many bytes of data have arrived so far (divide by two to get the number of pixels). This level of feedback is unimportant to many users.

RETURN VALUE

TRUE means the status was checked successfully, FALSE indicates a bad handle, a problem communicating with the camera or driver, or some type of readout failure. Failure will set `pl_error_code`.

SEE ALSO

`pl_exp_setup_seq(3),pl_exp_start_seq(3)`

NOTES

PVCAM **Class 3: Data Acquisition** **pl_exp_check_cont_status(3)**

NAME

`pl_exp_check_cont_status` – checks the continuous readout status from the camera into a circular buffer.

SYNOPSIS

```
rs_bool
   pl_exp_check_cont_status(int16 hcam, int16_ptr
                            status,uns32_ptr byte_cnt,
                            uns32_ptr buffer_cnt)
```

DESCRIPTION

This function will return the status of a continuous readout from the camera into a circular buffer. *status* is a pointer to one of the following values:

`READOUT_NOT_ACTIVE`	The system is **idle**, no data is expected. If any arrives, it will be discarded.
`EXPOSURE_IN_PROGRESS`	The data collection routines are **active**. They are waiting for data to arrive, but none has arrived yet.
`READOUT_IN_PROGRESS`	The data collection routines are **active**. The data has started to arrive.
`FRAME_AVAILABLE`	There is at least one frame which has not yet been retrieved from the buffer.
`READOUT_FAILED`	Something went wrong. The function returns a FALSE and `pl_error_code` is set. (See Return Value below for more information.)

The *`byte_cnt`* reports on how many bytes of data have arrived so far and overflows at MAX(UNS32).

The *`buffer_cnt`* points to the number of times the buffer has been filled.

RETURN VALUE

TRUE is returned for success, FALSE for a failure. Failure will set `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_get_oldest frame(3)`, `pl_exp_get_latest_frame(3)`, `pl_exp_unlock_oldest_frame(3)`, and `pl_exp_stop_cont(3)`

NOTES

This function only returns meaningful results if a continuous readout from the camera has been initiated by a call to `pl_exp_start_cont()`. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM **Class 3: Data Acquisition** **pl_exp_check_cont_status_ex(3)**

NAME

`pl_exp_check_cont_status_ex` – checks the continuous readout status from the camera into a circular buffer.

SYNOPSIS

```
rs_bool
   pl_exp_check_cont_status_ex(int16 hcam, int16_ptr
                               status,uns32_ptr byte_cnt,
                               uns32_ptr buffer_cnt,
                               PFRAME_INFO pFrameInfo);
```

DESCRIPTION

This function will return the status of a continuous readout from the camera into a circular buffer. *status* is a pointer to one of the following values:

```
READOUT_NOT_ACTIVE  EXPOSURE_IN_PROGRESS,
READOUT_IN_PROGRESS  ACQUISITION_IN_PROGRESS,
READOUT_COMPLETE  READOUT_FAILED.
```

byte_cnt reports on how many bytes of data have arrived so far and overflows at MAX(UNS32).

buffer_cnt points to the number of times the buffer has been filled.

Values in the variable pointed to by pFrameInfo will be updated with frame counters, timestamps (with precision of 0.1ms) and readout time information assigned by device driver at the moment of frame reception.

RETURN VALUE

TRUE is returned for success, FALSE for a failure. Failure will set `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_get_oldest frame(3)`, `pl_exp_get_latest_frame(3)`, `pl_exp_unlock_oldest_frame(3)`, and `pl_exp_stop_cont(3)`, `pl_create_frame_info_struct(2)`, `pl_exp_get_latest_frame_ex(3)`, `pl_exp_get_oldest_frame_ex(3)`

NOTES

This function only returns meaningful results if a continuous readout from the camera has been initiated by a call to `pl_exp_start_cont()`. Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

Variable pointed to by *pFrameInfo* must be created with `pl_create_frame_info_struct(2)`.

PVCAM	**Class 3: Data Acquisition**	**pl_exp_uninit_seq(3)**

NAME

`pl_exp_uninit_seq` – uninitializes the data collection functions.

SYNOPSIS

```
rs_bool
   pl_exp_uninit_seq(void)
```

DESCRIPTION

This function undoes the preparations done by `pl_exp_init_seq`. After executing this function, acquisition cannot take place.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_pvcam_init(0)`,`pl_pvcam_uninit(0)`,`pl_exp_init_seq(3)`

NOTES

You must explicitly call this function before calling `pl_pvcam_uninit`.

PVCAM	**Class 3: Data Acquisition**	**pl_exp_unlock_oldest_frame(3)**

NAME

`pl_exp_unlock_oldest_frame` - makes oldest frame in circular buffer overwriteable.

SYNOPSIS

```
rs_bool
   pl_exp_unlock_oldest_frame(int16 hcam)
```

DESCRIPTION

This function unlocks the oldest frame in the circular buffer; the frame should have been locked previously by a call to `pl_exp_get_oldest_frame`.

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_exp_setup_cont(3)`, `pl_exp_start_cont(3)`, `pl_exp_check_cont_status(3)`, `pl_exp_get_oldest frame(3)`, `pl_exp_unlock_oldest_frame(3)`, and `pl_exp_stop_cont(3)`

NOTES

Failure to call this function after using the frame will cause the continuous acquisition progress to halt eventually, because the frame cannot be overwritten when it is locked.

Use the parameter id `PARAM_CIRC_BUFFER` with `pl_get_param` to check to see if the system can perform circular buffer operations.

PVCAM	**Class 3: Data Acquisition** **pl_io_clear_script_control(3)**
NAME	`pl_io_clear_script_control` – Clears the current setup for control of the available I/O lines within a camera script.
SYNOPSIS	**`rs_bool`** **`pl_io_clear_script_control(int16 hcam)`**
DESCRIPTION	This function allows the application program to clear the current setup for control of the available I/O lines within the script. This allows the user to enter a new setup for these lines.
RETURN VALUE	TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.
SEE ALSO	`pl_io_script_control(3)`
NOTES	

PVCAM **Class 3: Data Acquisition** **pl_io_script_control(3)**

NAME

`pl_io_script_control` - Defines control of an I/O line from within a camera script.

SYNOPSIS

```
rs_bool
   pl_io_script_control(int16 hcam, uns16 addr, flt64  state,
                        uns32 location)
```

DESCRIPTION

This function allows the application program to define control of the available I/O lines from within a script. This allows for more precise control of external devices. For example, the application could request that a linear stage be indexed immediately after integration, instead of waiting until after the data is read out, the shutter is closed, etc. *`addr`* specifies which I/O address to control. *`state`* specifies the desired setting for the address being controlled.

`state` has different meanings depending on the I/O type:

`IO_TYPE_TTL`	The bit pattern written to this address.
`IO_TYPE_DAC`	The value of the desired analog output written to the DAC at this address.

`location` can be set to the following values:

`SCR_PRE_OPEN SHTR`	`SCR_POST_OPEN_SHTR`
`SCR_PRE_FLASH`	`SCR_POST_FLASH`
`SCR_PRE_INTEGRATE`	`SCR_POST_INTEGRATE`
`SCR_PRE_READOUT`	`SCR_POST_READOUT`
`SCR_PRE_CLOSE_SHTR`	`SCR_POST_CLOSE_SHTR`

RETURN VALUE

TRUE for success, FALSE for a failure. Failure sets `pl_error_code`.

SEE ALSO

`pl_io_clear_script_control(3)`

NOTES

Class 3 Parameter IDs

Note: Before trying to use or retrieve more information about a parameter, it is always recommended to call an ATTR_AVAIL to see if the system supports it.

Class 3 Parameter ID	Description
PARAM_BOF_EOF_CLR **Camera Dependent**	Clears the BOF-EOF count when a `pl_set_param` is performed. This is a write-only parameter. Datatype: **rs_bool**
PARAM_BOF_EOF_COUNT **Camera Dependent**	Returns the Begin-Of-Frame and/or End-Of-Frame count. `BOF_EOF` counting is enabled and configured with `PARAM_BOF_EOF_ENABLE`. Datatype: **uns32**

PARAM_BOF_EOF_ENABLE
 	
Camera Dependent
	Enables and configures the `BOF_EOF` interrupts. Possible values are:
	
	`NO_FRAME_IRQS`
	`BEGIN_FRAME_IRQS`
	`END_FRAME_IRQS`
	`BEGIN_END_FRAME_IRQS`
	
	Datatype: **enum**

PARAM_CIRC_BUFFER	Tests to see if the hardware/software can perform circular buffer. When a TRUE is returned, the circular buffer function can be used. Datatype: **rs_bool**
PARAM_CURRENT_PVTIME	Returns value of the current PVCAM time which is counting in 0.1ms increments from the moment of opening the camera by call to **pl_cam_open.** Datatype: **long64**
PARAM_EXP_MIN_TIME	Gets the minimum effective exposure time that can be set for the camera. For example, the exposure time may be limited by the required overhead for shifting the data through the array. This minimum time will be a floating point value, in seconds. Note that the minimum exposure time returned by this function will be greater than zero; any camera can provide a minimum exposure time of zero. Datatype: **flt64**
PARAM_EXP_RES	Gets the resolution for the current resolution index, as described for *PARAM_EXP_RES_INDEX*. This value is an enumerated type, representing the resolution. Possible values are : `EXP_RES_ONE_MILLISEC` `EXP_RES_ONE_MICROSEC`.

Class 3 Parameter ID	Description
	Datatype: **enum**
PARAM_EXP_RES_INDEX	Gets and sets the index into the exposure resolution table for the camera. The table contains the resolutions supported by the camera. The value at this index is an enumerated type, representing different resolutions (such as EXP_RES_ONE_MILLISEC or EXP_RES_ONE_MICROSEC). The number of supported resolutions can be obtained by using the *ATTR_COUNT* attribute with the *PARAM_EXP_RES_INDEX* parameter. Datatype: **uns16**
PARAM_EXP_TIME	This is used to examine and change the exposure time in VARIABLE_TIMED_MODE. Datatype: **uns16**

This page intentionally left blank.

Index

R

S

T

V

This page intentionally left blank.