PVCAM API changes for Prime camera
May 4, 2016

Theory of Operation

This document contains a summary of new features implemented for Prime camera. Please read the PVCAM User Manual to find all detail description for all features. Also check new examples from SDK to see the real usage.

User defined multiple ROIs

The ROIs are defined via the application's user interface. The position of each ROI is static and cannot change during acquisition. Once the user retrieves a pointer to a new frame, they will have an option to re-compose the frame into a list of ROIs and their corresponding metadata. For this feature, the Frame Metadata feature must be enabled, otherwise the frame re-composition may not be possible.

User defined ROIs are programmed into the camera as an array of regions via the existing **pl_exp_setup_seq/pl_exp_setup_cont** functions aimed at giving the camera the number of regions along with the coordinates of the regions to be captured.

The **pl_exp_setup_seq/pl_exp_setup_cont** functions trigger logic that first verifies if a particular camera supports the new modes, and an error is returned if the capability is not supported. This can also be checked in advance via the parameter PARAM_ROI_COUNT and ATTR_MAX. Furthermore, if an unsupported collection of ROIs is received by any camera, an error shall be reported by the camera system.

For multiple ROIs, the Frame Metadata feature must be enabled before setting up the acquisition with PARAM_METADATA_ENABLED parameter. Otherwise, an error shall be returned from the **pl_exp_setup_seq/pl_exp_setup_cont** functions.

A new frame decoding function added to the PVCAM API allows users to easily re-compose the multi-ROI frame into a displayable image. Once the user retrieves the frame buffer by calling **pl_exp_get_latest_frame**, they will pass the frame pointer to the re-composing function and retrieve either a displayable black-filled frame or a set of pointers pointing to each ROI data and metadata.

ROIs cannot be overlapping; this will be checked in the **pl_exp_setup_seq/pl_exp_setup_cont** functions. An error will be returned if any two ROIs are overlapping. However, it is noted that support for overlapping ROIs may be added in the future.

Prime Locate (centroids)

Centroids can be thought of as multiple ROIs that are chosen and generated by the camera itself. The user only specifies how many ROIs they are interested in and what should be their size. The size and number of ROIs cannot change during acquisition, however locations of the ROIs may change with each frame. Because of that an embedded frame metadata must be included for every acquired frame. The frame may be re-composed using the same re-composing function(s) as described in the Embedded Frame Metadata chapter.

The Centroids feature is enabled by PARAM_CENTROIDS_ENABLED parameter. Another two parameters, the PARAM_CENTROIDS_COUNT and PARAM_CENTROIDS_RADIUS are used to define the desired number of centroids and their size. The Embedded Frame Metadata feature must be enabled before starting the Centroids acquisition.

By calling the **pl_exp_setup_seq/pl_exp_setup_cont** functions the user only specifies a single ROI where the Centroids shall be generated. User-defined multiple ROIs cannot be combined with this feature.

Frame metadata

Metadata describes each frame acquired by the camera. It is generated by the camera, appended to the user buffer, and transferred together with the image data inside the same buffer. The metadata must include all information necessary to reconstruct the frame, along with other data-describing information such as timestamps, exposure time, or other related information.

The Frame Metadata feature is enabled by the parameter PARAM_METADATA_ENABLED. This feature is essential for the Multiple ROIs feature. However, for single ROI acquisition, the Metadata can be disabled by the user in order to instruct PVCAM to return the raw frame buffer without metadata. Since the Frame Metadata feature will be disabled by default, this will allow legacy applications to work as usually.

Frame re-composition

Once the user retrieves a pointer to the buffer containing a frame with multiple ROIs, they will need to retrieve the ROIs from the frame. PVCAM provides helper function(s) allowing the user to easily decode the frame buffer and obtain a set of pointers to each ROI data and metadata. Another function has been added to reconstruct the frame from the multi-ROI buffer. See Buffer decoding and re-composition chapter for details.

Extended metadata

The idea behind this approach is based on the following presumptions:

1) The transparent frame header and ROI header structures shall contain only the information necessary for decoding and re-composing the frame and information that is provided by every camera supporting the metadata feature. The definition (size) of these headers shall not change in future releases as it would cause binary and camera firmware compatibility issues.
2) Each header structure shall contain an "*Extended Metadata Size*" that specifies the size of an opaque structure that is located behind each header. The opaque structure contains the additional metadata. If the header is not followed by the extended metadata, the reported extended size must be zero.
3) Contents of the opaque structure are tagged. A list of supported extended metadata TAGs will be defined in a separate header file that is shared between PVCAM and camera firmware code.
4) If a new TAG needs to be added, a new PVCAM version must be released. However, running older PVCAM with new camera won't break anything, only the newly introduced TAGs will not be available. Also, existing code does not have to be recompiled when a new PVCAM DLL is loaded by the system. If the code does not require a particular TAG, it can still display all the TAGs from the frame because PVCAM DLL provides the TAG names and their types at runtime.

Extended metadata tag IDs are collected in `PL_MD_EXT_TAGS` enumeration, located either in pvcam.h.

The extended metadata feature is currently unused and there are no tag IDs declared in public header.

Switching on/off PrimeEnhance (De-noising)

The PrimeEnhance de-noising feature has been implemented in PVCAM as a post-processing feature with the feature ID PP_FEATURE_DENOISING. For more information on how to manipulate post-processing

features in PVCAM, please review the document “PVCam Postprocessing App Note.pdf” and example deployed with the PVCAM SDK.

Binning factor discovery

Binning factors entered during acquisition setup as a part of region have only two restrictions – both serial and parallel factors have to be positive 16-bit numbers and in each direction the factors must not be greater than sensor resolution. Cameras support only very small subset of possible values. Symmetrical binning factors that are power of two (1, 2, 4, 8 …) are usually well supported but applications previously had no way how to check which other values are allowed.

To enable this discover, two new read-only parameters were introduced: PARAM_BINNING_SER and PARAM_BINNING_PAR. These parameters must be implemented on each camera separately, so it is essential to ensure the ATTR_AVAIL returns TRUE before assuming the parameters are implemented.

These parameters are intended to be used together in a pair as they provide a list with all the supported binning permutations. ATTR_COUNT reports the same number of permutations for both parameters. Each parameter can and most probably will contain duplicate values but serial × parallel pairs are all unique. Also both parameters return the same string for each value.

Triggering table

The triggering table feature deals with output signals available on camera connector and has nothing in common with triggering/exposure modes. If camera supports this feature it has a one or more multiplexers connected output signals. The multiplexer can cycle the signal it is connected to (in round-robin fashion) over all or configured number of its outputs.

The configuration is do in two steps. The output signal to be configured can be selected via parameter PARAM_TRIGTAB_SIGNAL. The number of active output wires to be used for selected signal can be changed by parameter PARAM_LAST_MUXED_SIGNAL.

By default the configured number of multiplexed outputs is set to one for all signals. This has the same effect as there would be no multiplexer connected in the signal path.

Exposure time limits

Even though the exposure time is in general 32 bit unsigned number, there were always some limits. The real min. and max. values of exposure time differ for each camera and also for each exposure resolution. Until now there was no way how to get these limits. For Prime camera has been introduced new parameter PARAM_EXPOSURE_TIME which offers among the range currently set value.

Fan control

It is possible to control the fan speed e.g. to reduce vibrations during tests. See PARAM_FAN_SPEED_SETPOINT parameter.

New PVCAM structures

md_frame structure

For easier frame and metadata decoding, and thus easier integration with 3rd party software, a new structure describing the entire frame will be introduced. The structure will be used as an input or output argument for frame decoding and recomposing functions. Please refer to the Buffer decoding and re-composition chapter for further details.

This structure needs to be prepared before the acquisition with `pl_md_create_frame_struct()` or `pl_md_create_frame_struct_cont()`. After use the structure should be freed with `pl_md_release_frame_struct()`.

```
/**
This is a helper structure that is used to decode the md_frame_header. Since
the header cannot contain any pointers we need to calculate all information
using only offsets.
Please note the structure keeps only pointers to data residing in the image
buffer. Once the buffer is deleted the contents of the structure become invalid.
*/
typedef struct md_frame
{
    md_frame_header*     header;       /**< Points directly to the header within the buffer. */
    void*                extMdData;    /**< Points directly to ext/ MD data within the buffer. */
    uns16                extMdDataSize;/**< Size of the ext. MD buffer in bytes. */
    rgn_type             impliedRoi;   /**< Implied ROI calculated during decoding. */

    md_frame_roi*        roiArray;     /**< An array of ROI descriptors. */
    uns16                roiCapacity;  /**< Number of ROIs the structure can hold. */
    uns16                roiCount;     /**< Number of ROIs found during decoding. */
}
md_frame;
```

md_frame_header structure

This structure contains the same information as the current `FRAME_INFO` structure but with additional items and greater timestamp accuracy. There is no need to allocate or prepare this structure anywhere. This structure is part of the image buffer and needs to be decoded with provided functions.

```
/**
This is a frame header that is located before each frame. The size of this
structure must remain constant. The structure is generated by the camera
and should be 16-byte aligned.
*/
typedef struct md_frame_header
{                                /* TOTAL: 48 bytes */
    uns32       signature;       /* 4B (see the signature definition) */
    uns8        version;         /* 1B (must be 1 in the first release) */

    uns32       frameNr;         /* 4B (1-based, reset with each acquisition) */
    uns16       roiCount;        /* 2B (Number of ROIs in the frame, at least 1) */

    /* The final timestamp = timestampBOF * timestampResNs (in nano-seconds) */
    uns32       timestampBOF;    /* 4B (depends on resolution) */
    uns32       timestampEOF;    /* 4B (depends on resolution) */
    uns32       timestampResNs;  /* 4B (1=1ns, 1000=1us, 5000000=5ms, ...) */

    /* The final exposure time = exposureTime * exposureTimeResNs (nano-seconds) */
    uns32       exposureTime;    /* 4B (depends on resolution) */
    uns32       exposureTimeResNs;/* 4B (1=1ns, 1000=1us, 5000000=5ms, ...) */
```

```
    /* ROI timestamp resolution is stored here, no need to transfer with each ROI */
    uns32       roiTimestampResNs;/* 4B ROI timestamps resolution */

    uns8        bitDepth;         /* 1B (must be 10, 13, 14, 16, etc.) */
    uns8        colorMask;        /* 1B (corresponds to PL_COLOR_MODES) */
    uns8        flags;            /* 1B Frame flags */
    uns16       extendedMdSize;   /* 2B (must be 0 or actual ext md data size) */
    uns8        _reserved[8];
}
md_frame_header;
```

md_frame_roi structure

The md_frame_roi structure is a helper structure used to decode the md_frame_roi_header.

```
/**
This is a helper structure that is used to decode the md_frame_roi_header. Since
the header cannot contain any pointers PVCAM will calculate all information
using offsets from frame & ROI headers.
The structure must be created using the pl_md_create_frame_struct() function.
Please note the structure keeps only pointers to data residing in the image
buffer. Once the buffer is deleted the contents of the structure become invalid.
*/
typedef struct md_frame_roi
{
    md_frame_roi_header*    header;         /**< Points directly to the header within the buffer. */
    void*                   data;           /**< Points to the ROI image data. */
    uns32                   dataSize;       /**< Size of the ROI image data in bytes. */
    void*                   extMdData;      /**< Points directly to ext/ MD data within the buffer. */
    uns16                   extMdDataSize;  /**< Size of the ext. MD buffer. */
}
md_frame_roi;
```

md_frame_roi_header structure

This structure contains similar members to the frame header. This structure is part of the image buffer and need to be decoded using provided functions.

```
/**
This is a ROI header that is located before every ROI data. The size of this
structure must remain constant. The structure is generated by the camera
and should be 16-byte aligned.
*/
typedef struct md_frame_roi_header
{                               /* TOTAL: 32 bytes */
    uns16    roiNr;             /* 2B (1-based, reset with each frame) */

    /* The final timestamp = timestampBOR * roiTimestampResNs */
    uns32    timestampBOR;      /* 4B (depends on RoiTimestampResNs) */
    uns32    timestampEOR;      /* 4B (depends on RoiTimestampResNs) */

    rgn_type roi;               /* 12B (ROI coordinates and binning) */

    uns8     flags;             /* 1B ROI flags */
    uns16    extendedMdSize;    /* 2B (must be 0 or actual ext md data size in bytes) */
    uns8    _reserved[7];
}
md_frame_roi_header;
```

New PVCAM parameters

PARAM_ROI_COUNT

This parameter will be used to check whether the camera supports the user-defined multiple ROIs feature.

Attribute	Value	Description
ATTR_MIN	1	1 ROI is always supported
ATTR_MAX	N	Maximum number of ROIs supported by the camera
ATTR_COUNT	N	Maximum number of ROIs supported by the camera
ATTR_INCREMENT	1	
ATTR_DEFAULT	1	
ATTR_CURRENT	N	Currently configured number of ROIs. (after `pl_exp_setup_xxx`)
ATTR_AVAIL		TRUE if the MultiROI capability bit is enabled, FALSE by default
ATTR_ACCESS	R	Read only, the ROIs are set via the pl_exp_setup functions
ATTR_TYPE	uns16	The `pl_exp_setup` functions use uns16 for `rgn_total` as well

PARAM_METADATA_ENABLED

This parameter is used to enable/disable the frame metadata. Once enabled the image buffer will no longer contain raw data – additional frame and ROI headers will be added to the buffer.

Attribute	Value	Description
ATTR_MIN	FALSE	
ATTR_MAX	TRUE	
ATTR_COUNT	0	
ATTR_INCREMENT	FALSE	
ATTR_DEFAULT	FALSE	
ATTR_CURRENT		TRUE or FALSE, depending on the current setting
ATTR_AVAIL		TRUE if the Metadata capability bit is enabled, FALSE by default
ATTR_ACCESS	RW	Read write, this parameter is settable
ATTR_TYPE	rs_bool	

PARAM_FRAME_BUFFER_SIZE

This parameter is used to get the recommended circular buffer size in bytes. With PVCAM 3.1 a full advantage of circular buffer can be taken. With very fast acquisition (1000FPS and more) it is recommended to use large circular buffer size to avoid dropped frames. Use the ATTR_DEFAULT to get the recommended size for the actual acquisition (becomes available after pl_exp_setup_xxx() is called).

Attribute	Value	Description
ATTR_MIN		Minimum allowed buffer size in bytes
ATTR_MAX		Maximum supported size of the buffer in bytes
ATTR_COUNT	0	
ATTR_INCREMENT		Size of a single frame in bytes
ATTR_DEFAULT		Recommended size of the circular buffer in bytes
ATTR_CURRENT		Currently configured size of the circular/sequence buffer in bytes
ATTR_AVAIL	TRUE	Parameter is supported for all cameras

ATTR_ACCESS	R	This parameter is used to get recommended size of buffer
ATTR_TYPE	ulong64	Used 64bit type for future, now all values fit to 32bit variable

PARAM_FAN_SPEED_SETPOINT

This parameter allows to change the fan speed. However, camera can fine tune the speed or power off itself to prevent its damage.

Attribute	Value	Description
ATTR_MIN		A value from PL_FAN_SPEEDS enum
ATTR_MAX		A value from PL_FAN_SPEEDS enum
ATTR_COUNT		Number of supported values from PL_FAN_SPEEDS enum
ATTR_INCREMENT	0	
ATTR_DEFAULT		
ATTR_CURRENT		A value from PL_FAN_SPEEDS enum
ATTR_AVAIL		TRUE if the Metadata capability bit is enabled, FALSE by default
ATTR_ACCESS	RW	Read write, this parameter is settable
ATTR_TYPE	int32	TYPE_ENUM

PARAM_CENTROIDS_ENABLED

This parameter is used to enable/disable the Prime Locate feature. Once enabled the image buffer will no longer contain raw data – additional frame and ROI headers will be added to the buffer.

Attribute	Value	Description
ATTR_MIN	FALSE	
ATTR_MAX	TRUE	
ATTR_COUNT	0	
ATTR_INCREMENT	TRUE	
ATTR_DEFAULT	FALSE	
ATTR_CURRENT		TRUE or FALSE, depending on the current setting
ATTR_AVAIL		TRUE if the Centroids capability bit is enabled, FALSE by default
ATTR_ACCESS	RW	Read write, this parameter is settable
ATTR_TYPE	rs_bool	

PARAM_CENTROIDS_RADIUS

This parameter will be used to get/set the size of centroids. The metadata will contains detected centroids which all have the same size and can overlap. The size of region is calculated as two times radius plus one.

Attribute	Value	Description
ATTR_MIN		
ATTR_MAX		
ATTR_COUNT		
ATTR_INCREMENT	1	
ATTR_DEFAULT		
ATTR_CURRENT	N	Currently configured radius of centroids
ATTR_AVAIL		TRUE if the Centroids capability bit is enabled, FALSE by default

ATTR_ACCESS	RW	Read write, this parameter is settable
ATTR_TYPE	uns16	

PARAM_CENTROIDS_COUNT

This parameter will be used to get/set the number of centroids. If the Prime Locate algorithm is unable to detect so many centroids, either the last found centroid data is copied to all remaining centroids or those remaining centroids have set flag PL_MD_ROI_FLAG_INVALID. Invalid regions are skipped in `pl_md_frame_recompose` function.

Attribute	Value	Description
ATTR_MIN		
ATTR_MAX		
ATTR_COUNT		
ATTR_INCREMENT	1	
ATTR_DEFAULT		
ATTR_CURRENT	N	Currently configured number of centroids
ATTR_AVAIL		TRUE if the Centroids capability bit is enabled, FALSE by default
ATTR_ACCESS	RW	Read write, this parameter is settable
ATTR_TYPE	uns16	

PARAM_BINNING_SER and PARAM_BINNING_PAR

This pair of parameters provides a list of supported binning permutations. For instance if both parameters would provide three values 1, 2 and 4, the only supported binning factors would be 1×1, 2×2 and 4×4. The parameter is read-only as the setting is done via pl_exp_setup_xxx() as a part of rgn_type structure.

Attribute	Value	Description
ATTR_MIN	1	1 = no binning
ATTR_MAX		
ATTR_COUNT		Number of supported values from PL_FAN_SPEEDS enum
ATTR_INCREMENT	0	
ATTR_DEFAULT	1	No binning by default
ATTR_CURRENT		Currently configured binning
ATTR_AVAIL		TRUE if the Binning capability bit is enabled, FALSE otherwise
ATTR_ACCESS	R	This parameter is used to get serial/parallel binning factors
ATTR_TYPE	int32	TYPE_ENUM

PARAM_EXPOSURE_TIME

This parameter will be used to get current exposure time and value boundaries.

Attribute	Value	Description
ATTR_MIN		
ATTR_MAX		
ATTR_COUNT	0	There could be more valid values than fit int 32bit count variable
ATTR_INCREMENT	1	
ATTR_DEFAULT		

ATTR_CURRENT	N	Currently configured exposure time
ATTR_AVAIL	TRUE	Parameter is always available
ATTR_ACCESS	R	Read only
ATTR_TYPE	ulong64	

Buffer decoding and re-composition

Since both frame header and ROI header are transparent structures with constant size, the position of the headers and data can be calculated using simple pointer arithmetic. Helper functions have been added to the PVCAM public API to allow easier integration with 3rd party applications.

The following two functions are used to create an `md_frame` helper structure that is used for decoding the metadata-enabled frame buffer.

```
/**
This method creates an empty md_frame structure for known number of ROIs.
Use this method to prepare and pre-allocate one structure before starting
continuous acquisition. Once callback arrives fill the structure with
pl_md_frame_decode() and display the metadata.
Release the structure when not needed.
@param pFrame a pointer to frame helper structure address where the structure
              will be allocated.
@param roiCount Number of ROIs the structure should be prepared for.
@return #PV_FAIL in case of failure.
*/
rs_bool PV_DECL pl_md_create_frame_struct_cont (md_frame** pFrame, uns16 roiCount);

/**
This method creates an empty md_frame structure from an existing buffer.
Use this method when loading buffers from disk or when performance is not
critical. Do not forget to release the structure when not needed.
For continuous acquisition where the number or ROIs is known it is recommended
to use the other provided method to avoid frequent memory allocation.
@param pFrame A pointer address where the newly created structure will be stored.
@param pSrcBuf A raw frame data pointer as returned from the camera
@param srcBufSize Size of the raw frame data buffer
@return #PV_FAIL in case of failure
*/
rs_bool PV_DECL pl_md_create_frame_struct (md_frame** pFrame, void* pSrcBuf, uns32 srcBufSize);
```

The following function takes the raw buffer and its size in bytes as an input and fills in the `md_frame` structure.

```
/**
Decodes all the raw frame buffer metadata into a friendly structure.
@param pDstFrame A pre-allocated helper structure that will be filled with
                 information from the given raw buffer.
@param pSrcBuf A raw frame buffer as retrieved from PVCAM
@param srcBufSize The size of the raw frame buffer
@return #PV_FAIL in case of failure.
*/

rs_bool PV_DECL pl_md_frame_decode (md_frame* pDstFrame, void* pSrcBuf, uns32 srcBufSize);
```

The next function takes the `md_frame` structure as an input and copies all ROIs into their respective positions in the user's temporary buffer. It expects the user buffer is big enough to cover all ROIs. Destination buffer width is required by the reconstruction algorithm.

```
/**
Optional function that recomposes a multi-ROI frame into a displayable image buffer.
Every ROI will be copied into its appropriate location in the provided buffer.
```

```
Please note that the function will subtract the Implied ROI position from each ROI
position which essentially moves the entire Implied ROI to a [0, 0] position.
Use the Offset arguments to shift all ROIs back to desired positions if needed.
If you use the Implied ROI position for offset arguments the frame will be recomposed
as it appears on the full frame.
The caller is responsible for black-filling the input buffer. Usually this function
is called during live/preview mode where the destination buffer is re-used. If the
ROIs do move during acquisition it is essential to black-fill the destination buffer
before calling this function. This is not needed if the ROIs do not move.
If the ROIs move during live mode it is also recommended to use the offset arguments
and recompose the ROI to a full frame - with moving ROIs the implied ROI may change
with each frame and this may cause undesired ROI "twitching" in the displayable image.

@param pDstBuf An output buffer, the buffer must be at least the size of the implied
               ROI that is calculated during the frame decoding process. The buffer
               must be of type uns16. If offset is set the buffer must be large
               enough to allow the entire implied ROI to be shifted.
@param offX    Offset in the destination buffer, in pixels. If 0 the Implied
               ROI will be shifted to position 0 in the target buffer.
               Use (ImpliedRoi.s1 / ImplierRoi.sbin) as offset value to
               disable the shift and keep the ROIs in their absolute positions.
@param offY    Offset in the destination buffer, in pixels. If 0 the Implied
               ROI will be shifted to position 0 in the target buffer.
               Use (ImpliedRoi.p1 / ImplierRoi.pbin) as offset value to
               disable the shift and keep the ROIs in their absolute positions.
@param dstWidth  Width, in pixels of the destination image buffer. The buffer
                 must be large enough to hold the entire Implied ROI, including
                 the offsets (if used).
@param dstHeight Height, in pixels of the destination image buffer.
@param pSrcFrame A helper structure, previously decoded using the frame
                 decoding function.
@return #PV_FAIL in case of failure.
*/
rs_bool PV_DECL pl_md_frame_recompose (void* pDstBuf, uns16 offX, uns16 offY,
                                       uns16 dstWidth, uns16 dstHeight, md_frame* pSrcFrame);
```

The last function helps with extended metadata decoding and takes as input RAW metadata for a frame or ROI (located in `md_frame`) and transforms it into a metadata collection given as an output argument. Please note that extended metadata feature is not implemented in the initial release.

```
/**
Reads all the extended metadata from the given ext. metadata buffer.
@param pOutput A pre-allocated structure that will be filled with metadata
@param pExtMdPtr A pointer to the ext. MD buffer, this can be obtained from
                 the md_frame and md_frame_roi structures.
@param extMdSize Size of the ext. MD buffer, also retrievable from the helper
                 structures.
@return #PV_FAIL in case the metadata cannot be decoded.
*/
rs_bool PV_DECL pl_md_read_extended (md_ext_item_collection* pOutput, void* pExtMdPtr,
                                     uns32 extMdSize);
```

Backward compatibility

The Frame Metadata feature will be disabled by default. The user buffer will contain no additional metadata, and the structure will be the same as in older PVCAMs.