The OpenVX™ Raw Image Extension

The Khronos® OpenVX Working Group, Editor: Jesse Villarreal

Version 1.0, Mon, 02 Oct 2023 15:00:54 +0000: Git branch information not available

Table of Contents

1. Introduction . . . 2
   1.1. Purpose . . . 2
   1.2. In Scope . . . 2
   1.3. Not in Scope . . . 2
2. Design Overview . . . 3
   2.1. Changes to the OpenVX Specification . . . 3
      2.1.1. New Functions . . . 3
      2.1.2. Extended Functions . . . 3
         vxQueryImage . . . 3
         vxMapImagePatch . . . 4
         vxCopyImagePatchWithFlags . . . 4
   2.2. Example Code . . . 5
      2.2.1. Create and Release . . . 5
      2.2.2. Query and Map/Unmap or Copy . . . 7
3. Module Documentation . . . 8
   3.1. Macros . . . 8
      3.1.1. VX_IMAGE_RAW_MAX_EXPOSURES . . . 8
      3.1.2. VX_IMAGE_IS_RAW . . . 8
      3.1.3. VX_IMAGE_RAW_EXPOSURE_INTERLEAVING . . . 9
      3.1.4. VX_IMAGE_RAW_FORMAT . . . 9
      3.1.5. VX_IMAGE_RAW_META_HEIGHT_BEFORE . . . 9
      3.1.6. VX_IMAGE_RAW_META_HEIGHT_AFTER . . . 9
      3.1.7. VX_DF_IMAGE_RAW . . . 9
      3.1.8. VX_ENUM_IMAGE_RAW_BUFFER_ACCESS . . . 9
      3.1.9. VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER . . . 9
      3.1.10. VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING . . . 10
   3.2. Typedefs . . . 10
      3.2.1. vx_image_raw_format_t . . . 10
      3.2.2. vx_image_raw_create_params_t . . . 10
   3.3. Enumerations . . . 11
      3.3.1. vx_image_raw_buffer_access_e . . . 11
      3.3.2. vx_image_raw_pixel_container_e . . . 11
      3.3.3. vx_image_raw_exposure_interleaving_e . . . 11
   3.4. Functions . . . 12
      3.4.1. vxCreateRawImage . . . 12
      3.4.2. vxCreateVirtualRawImage . . . 12
      3.4.3. vxCopyImagePatchWithFlags . . . 13
Index . . . 15

Chapter 1. Introduction

1.1. Purpose

This document details an extension to any OpenVX version from 1.1 to 1.3.1, and references some APIs and symbols that may be found in those APIs: https://www.khronos.org/registry/OpenVX/.

This extension is intended to add Raw Image support to the vx_image object in OpenVX.

1.2. In Scope

In the context of this extension, a "raw image" contains the readout of camera sensors which output pre-processed "raw" images, including, but not limited to, Bayer sensor readouts. The raw image data type, for example, may be used as an output parameter of a custom camera capture OpenVX source node.

This extension is focused on specifying the minimal set of attributes required to store/access a variety of known sensor readout information in memory. The following features were included to accommodate known readout types across a variety of raw image sensors commonly used in the industry:

- **Multi-exposure readout** - There is a class of wide dynamic range (WDR) sensors that support multiple exposures per timestamp. Some of these sensors read out each exposure on different CSI Virtual Channels, and others read out all exposures in a single CSI Virtual Channel in interleaved fashion. Data access APIs like "map" and "copy" allow the user to access data from a specific exposure while abstracting if the data was interleaved or not during read out.
    - **Separate CSI Virtual Channels** - When each exposure is read out on separate CSI Virtual Channels, then each exposure can be stored in memory in separate planes. Additionally, this mode enables each exposure to be stored as a different bit width (e.g. 16+12 mode).
    - **Single CSI Virtual Channel** - When all exposures are read out in a single CSI Virtual Channel, they are read out in either a line interleaved or pixel interleaved fashion, requiring all exposures to be stored accordingly in memory in a single buffer. The raw image object has a parameter that can be set to indicate the interleaved format of the readout of these exposures.
- **Meta Data** - Several sensor vendors offer various, vendor-specific meta information that is read out either before, or after, the pixel data. This meta data may contain side-band information such as histograms or statistics that may be used by auto exposure or auto white balance algorithms running on the application processor. The image data type created for raw images, therefore, can be configured with programmable number of lines of meta data before, and/or after, the pixel data. Data access APIs like "map" and "copy" allow the user to access the pixel data, or the meta data, accordingly.

1.3. Not in Scope

This extension does NOT specify any new kernels, nor does it specify which existing OpenVX vision kernels are expected to support raw images. These details shall be defined by specific implementations at this time.

Chapter 2. Design Overview

Instead of creating a new data type for raw images, this extension primarily extends the existing `vx_image` data type. Most of the existing image data object functions can be reused with minor extensions (outlined below), however it does require a new 'create' function to create images of raw format type, where additional information should be passed from new types and enum definitions.

2.1. Changes to the OpenVX Specification

This section outlines the new functions, extensions to existing functions, and other changes to the OpenVX Specification included in this extension.

2.1.1. New Functions

There are only three new functions added as part of this extension:

- `vxCreateRawImage`
- `vxCreateVirtualRawImage`
- `vxCopyImagePatchWithFlags`

One of the two create functions should be used to create a `vx_image` object that contains raw image data instead of using the existing image create functions.

The new copy function is meant to be a generic extension to the main spec, since the original copy image function does not have the 'flags' parameter like the image map function does. This 'flags' parameter is a generic extension field that is needed by this extension, so we are defining this new copy image function here for consideration as part of a future version of the main specification.

The API details of these functions, along with associated new data structures are found in the Module Documentation section.

2.1.2. Extended Functions

This section clarifies how to use the following `vx_image` functions with raw images:

- `vxQueryImage`
- `vxMapImagePatch`
- `vxCopyImagePatchWithFlags`

vxQueryImage

The `vxQueryImage` function can be used with the following new attribute definitions added to support raw images:

- `VX_IMAGE_IS_RAW`
- `VX_IMAGE_RAW_EXPOSURE_INTERLEAVING`
- `VX_IMAGE_RAW_FORMAT`
- `VX_IMAGE_RAW_META_HEIGHT_BEFORE`
- `VX_IMAGE_RAW_META_HEIGHT_AFTER`

Generally, if the application is not sure if a vx_image object contains a raw image, it can query the vx_image using the VX_IMAGE_IS_RAW definition. If the query returns vx_true_e, then the other raw image attributes can also be queried.

The following table lists which attributes are valid for raw images, and which attributes are valid for all other images from the main specification.

Attribute	Main Spec Images	Raw Images	Comments for Raw Images
VX_IMAGE_IS_RAW	valid	valid	Returns vx_true_e
VX_IMAGE_WIDTH	valid	valid	
VX_IMAGE_HEIGHT	valid	valid	
VX_IMAGE_FORMAT	valid	valid	Returns VX_DF_IMAGE_RAW
VX_IMAGE_PLANES	valid	valid	Returns number of exposures
VX_IMAGE_SPACE	valid	valid	Returns VX_COLOR_SPACE_NONE
VX_IMAGE_RANGE	valid		
VX_IMAGE_MEMORY_TYPE	valid	valid	Returns VX_MEMORY_TYPE_NONE
VX_IMAGE_IS_UNIFORM	valid	valid	Returns vx_false_e
VX_IMAGE_UNIFORM_VALUE	valid		
VX_IMAGE_RAW_EXPOSURE_INTERLEAVING		valid	
VX_IMAGE_RAW_FORMAT		valid	
VX_IMAGE_RAW_META_HEIGHT_BEFORE		valid	
VX_IMAGE_RAW_META_HEIGHT_AFTER		valid	

If a query is made using an attribute which is not valid for the type of image as indicated in the table above, then a VX_ERROR_NOT_SUPPORTED error shall be returned.

vxMapImagePatch

The vxMapImagePatch function can be used with the following extensions:

- The 'plane_index' parameter can refer to the exposure_index of the raw image.
- The flags parameter shall be set to one of the values from the new vx_image_raw_buffer_access_e enumeration to specify which buffer within the raw image to map.

vxCopyImagePatchWithFlags

This extension relies on The vxCopyImagePatchWithFlags function can be used with the following extensions:

- The 'image_plane_index' parameter can refer to the exposure_index of the raw image.

- The flags parameter shall be set to one of the values from the new `vx_image_raw_buffer_access_e` enumeration to specify which buffer within the raw image to map.

2.2. Example Code

This section demonstrates the usage of the new APIs in example code.

2.2.1. Create and Release

The following example shows how the Raw Image create API could be used to connect two custom user kernels together in a graph. The example also defines a data structure used within the raw image: user_custom_data_t.

```
/*
 * Utility API used to create the graph below using raw image in between two
 * user defined nodes:
 *
 * The following graph is created,
 * vxRawSensorCaptureNode -> TMP_RAW_IMAGE -> vxRawProcessingNode -> VX_IMAGE
 *
 */
static vx_graph create_graph(vx_context context,
                             vx_enum raw_image_sensor_id)
{
    vx_graph graph;
    vx_node n0, n1;
    vx_image imgYuv;
    vx_image tmp_raw_image_obj;
    vx_user_data_object tuning_user_obj;
    vx_image_raw_create_params_t create_params;
    user_tuning_params_t tuning_params;

    graph = vxCreateGraph(context);

    /* Initialize create_params data structure from sensor driver */
    sensor_get_raw_create_params(raw_image_sensor_id, &create_params);

    /* create intermediate custom raw image object */
    tmp_raw_image_obj = vxCreateRawImage(context,
                                         &create_params,
                                         sizeof(vx_image_raw_create_params_t));

    /* create first node: This is a source node connected to the capture driver on
     * csi interface. output is an image object containing raw data */
    n0 = userRawSensorCaptureNode(graph, tmp_raw_image_obj);



    /* Initialize raw processing tuning parameters from sensor driver */
    sensor_get_tuning_params(raw_image_sensor_id, &tuning_params);

    /* create tuning parameters user data object to send as input to the
     * raw processing node */
    tuning_user_obj = vxCreateUserDataObject(context,
                                       "user_tuning_params_t",
                                       sizeof(user_tuning_params_t),
                                       &tuning_params);

    /* create output image object */
    imgYuv = vxCreateImage(context, create_params.width, create_params.height, VX_DF_IMAGE_YUYV);

    /* create second node: inputs are output of first node and tuning parameters,
     * output is yuv image */
    n1 = userRawProcessingNode(graph, tmp_raw_image_obj, tuning_user_obj, imgYuv);

    vxReleaseNode(&n0);
    vxReleaseNode(&n1);
    vxReleaseImage(&tmp_raw_image_obj);
    vxReleaseImage(&imgYuv);
    vxReleaseUserDataObject(&tuning_user_obj);

    return graph;
}
```

2.2.2. Query and Map/Unmap or Copy

The map/unmap and copy functions operate in a manner almost identical to the OpenVX specification `vx_image` functions, so those can be used more or less as a reference. The main difference is that the user can specify which exposure to access in the 'plane_index' paramter; and the user can also specify which type of data to access (pixel data, or meta data) as part of the 'flags' parameter.

Chapter 3. Module Documentation

Macros

- VX_IMAGE_RAW_MAX_EXPOSURES
- VX_IMAGE_IS_RAW
- VX_IMAGE_RAW_EXPOSURE_INTERLEAVING
- VX_IMAGE_RAW_FORMAT
- VX_IMAGE_RAW_META_HEIGHT_BEFORE
- VX_IMAGE_RAW_META_HEIGHT_AFTER
- VX_DF_IMAGE_RAW
- VX_ENUM_IMAGE_RAW_BUFFER_ACCESS
- VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER
- VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING

Typedefs

- vx_image_raw_format_t
- vx_image_raw_create_params_t

Enumerations

- vx_image_raw_buffer_access_e
- vx_image_raw_pixel_container_e
- vx_image_raw_exposure_interleaving_e

Functions

- vxCreateRawImage
- vxCreateVirtualRawImage
- vxCopyImagePatchWithFlags

3.1. Macros

3.1.1. VX_IMAGE_RAW_MAX_EXPOSURES

Maximum number of RAW image exposures that can be contained in a raw image object.

```
#define VX_IMAGE_RAW_MAX_EXPOSURES (3)
```

3.1.2. VX_IMAGE_IS_RAW

Image Attribute which queries if an image was created using vxCreateRawImage API. Read-only. Use a vx_bool parameter */

```
#define VX_IMAGE_IS_RAW (VX_ATTRIBUTE_BASE(VX_ID_KHRONOS, VX_TYPE_IMAGE) + 0xA)
```

3.1.3. VX_IMAGE_RAW_EXPOSURE_INTERLEAVING

Indicates if the exposures are interleaved in memory. Read-only. Use a vx_image_raw_exposure_interleaving_e parameter.

```
#define VX_IMAGE_RAW_EXPOSURE_INTERLEAVING (VX_ATTRIBUTE_BASE(VX_ID_KHRONOS, VX_TYPE_IMAGE) + 0xB)
```

3.1.4. VX_IMAGE_RAW_FORMAT

The format of the the raw image. Read-only. Use a pointer to a vx_image_raw_format_t array.

```
#define VX_IMAGE_RAW_FORMAT (VX_ATTRIBUTE_BASE(VX_ID_KHRONOS, VX_TYPE_IMAGE) + 0xC)
```

3.1.5. VX_IMAGE_RAW_META_HEIGHT_BEFORE

The meta height at top of readout of raw image. Read-only. Use a vx_uint32 parameter.

```
#define VX_IMAGE_RAW_META_HEIGHT_BEFORE (VX_ATTRIBUTE_BASE(VX_ID_KHRONOS, VX_TYPE_IMAGE) + 0xD)
```

3.1.6. VX_IMAGE_RAW_META_HEIGHT_AFTER

The meta height at bottom of readout of raw image. Read-only. Use a vx_uint32 parameter.

```
#define VX_IMAGE_RAW_META_HEIGHT_AFTER (VX_ATTRIBUTE_BASE(VX_ID_KHRONOS, VX_TYPE_IMAGE) + 0xE)
```

3.1.7. VX_DF_IMAGE_RAW

Returned for a raw image when the vxQueryImage API is called on the VX_IMAGE_FORMAT attribute.

```
#define VX_DF_IMAGE_RAW (VX_DF_IMAGE('R','A','W','0'))
```

3.1.8. VX_ENUM_IMAGE_RAW_BUFFER_ACCESS

A vx_image_raw_buffer_access_e

```
#define VX_ENUM_IMAGE_RAW_BUFFER_ACCESS (vx_enum)0x24
```

3.1.9. VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER

A vx_image_raw_pixel_container_e

```
#define VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER (vx_enum)0x25
```

3.1.10. VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING

A vx_image_raw_exposure_interleaving_e

```
#define VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING (vx_enum)0x26
```

3.2. Typedefs

3.2.1. vx_image_raw_format_t

The raw image format structure that is part of the vx_image_raw_create_params_t structure

```
typedef struct _vx_image_raw_format_t {
    vx_uint32    pixel_container;
    vx_uint32    msb;
} vx_image_raw_format_t;
```

- *pixel_container* - Pixel Container, see vx_image_raw_pixel_container_e
- *msb* - Most significant bit in pixel container.

3.2.2. vx_image_raw_create_params_t

The raw image create params structure that is given to the vxCreateRawImage function.

```
typedef struct _vx_image_raw_create_params_t {
    vx_uint32               width;
    vx_uint32               height;
    vx_uint32               num_exposures;
    vx_uint32               exposure_interleaving;
    vx_image_raw_format_t   format[VX_IMAGE_RAW_MAX_EXPOSURES];
    vx_uint32               meta_height_before;
    vx_uint32               meta_height_after;
} vx_image_raw_create_params_t;
```

- *width* - The image width in pixels
- *height* - TThe image height in lines (not including meta rows).
- *num_exposures* - The number of exposures contained in the sensor readout for a given timestamp. Max supported is VX_IMAGE_RAW_MAX_EXPOSURES
- *exposure_interleaving* - Indicates the type of exposure interleaving, if any, in memory. see vx_image_raw_exposure_interleaving_e.
- *format* - Array of vx_image_raw_format_t structures indicating the pixel packing and bit alignment format of each exposure. If *exposure_interleaving* == VX_IMAGE_RAW_PLANAR, then the number of valid structures in this array should be equal to the value of num_exposures. If *exposure_interleaving* != VX_IMAGE_RAW_PLANAR, then the format should be the same for each exposure in a single buffer, so the number of valid structures in this array should equal 1.
- *meta_height_before* - Number of lines of meta data at top of sensor readout (before pixel data) (uses the same width as original sensor readout width)

- *meta_height_after* - Number of lines of meta data at bottom of sensor readout (after pixel data) (uses the same width as original sensor readout width)

3.3. Enumerations

3.3.1. vx_image_raw_buffer_access_e

The raw image buffer access enum.

```
enum vx_image_raw_buffer_access_e {
    VX_IMAGE_RAW_ALLOC_BUFFER = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_BUFFER_ACCESS) + 0x0,
    VX_IMAGE_RAW_PIXEL_BUFFER = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_BUFFER_ACCESS) + 0x1,
    VX_IMAGE_RAW_META_BEFORE_BUFFER = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_BUFFER_ACCESS) + 0x2,
    VX_IMAGE_RAW_META_AFTER_BUFFER = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_BUFFER_ACCESS) + 0x3,
};
```

Enumerator

- VX_IMAGE_RAW_ALLOC_BUFFER - For accessing pointer to full allocated buffer (pixel buffer + meta buffer)
- VX_IMAGE_RAW_PIXEL_BUFFER - For accessing pointer to pixel buffer only
- VX_IMAGE_RAW_META_BEFORE_BUFFER - For accessing pointer to meta buffer only
- VX_IMAGE_RAW_META_AFTER_BUFFER - For accessing pointer to meta buffer only

3.3.2. vx_image_raw_pixel_container_e

The raw image pixel container enum.

```
enum vx_image_raw_pixel_container_e {
    VX_IMAGE_RAW_16_BIT = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER) + 0x0,
    VX_IMAGE_RAW_8_BIT = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER) + 0x1,
    VX_IMAGE_RAW_P12_BIT = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER) + 0x2,
};
```

Enumerator

- VX_IMAGE_RAW_16_BIT - Two bytes per pixel in memory
- VX_IMAGE_RAW_8_BIT - One byte per pixel in memory
- VX_IMAGE_RAW_P12_BIT - Packed 12 bit mode; Three bytes per two pixels in memory.

3.3.3. vx_image_raw_exposure_interleaving_e

The raw image exposure interleaving enum.

```
enum vx_image_raw_exposure_interleaving_e {
    VX_IMAGE_RAW_PLANAR = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING) + 0x0,
    VX_IMAGE_RAW_LINE_INTERLEAVED = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING) +
0x1,
    VX_IMAGE_RAW_PIXEL_INTERLEAVED = VX_ENUM_BASE(VX_ID_KHRONOS, VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING)
+ 0x2,
};
```

Enumerator

- `VX_IMAGE_RAW_PLANAR` - Each exposure is readout and stored in separate planes; single exposure per CSI virtual channel.
- `VX_IMAGE_RAW_LINE_INTERLEAVED` - Each exposure is readout and stored in line interleaved fashion; multiple exposures share same CSI virtual channel
- `VX_IMAGE_RAW_PIXEL_INTERLEAVED` - Each exposure is readout and stored in pixel interleaved fashion; multiple exposures share same CSI virtual channel.

3.4. Functions

3.4.1. vxCreateRawImage

Creates an opaque reference to a raw sensor image (including multi-exposure and metadata)

```
vx_image vxCreateRawImage(
    vx_context                                  context,
    const vx_image_raw_create_params_t*         params,
    vx_size                                     size);
```

Not guaranteed to exist until the vx_graph containing it has been verified.

Parameters

- `[in]` *context* - The reference to the overall Context.
- `[in]` *params* - The pointer to a `vx_image_raw_create_params_t` structure
- `[in]` *size* - The size of the structure pointed to by the params pointer, in bytes.

Returns: An image reference `vx_image`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`

3.4.2. vxCreateVirtualRawImage

Creates an opaque reference to a virtual raw sensor image with no direct user access (including multi-exposure and metadata).

```
vx_image vxCreateVirtualRawImage(
    vx_graph                                    graph,
    const vx_image_raw_create_params_t*         params,
    vx_size                                     size);
```

Virtual Raw Images are useful when the Raw Image is used as internal graph edge. Virtual Raw Images are scoped within the parent graph only.

Parameters

- [in] *graph* - The reference to the parent graph.
- [in] *params* - The pointer to a `vx_image_raw_create_params_t` structure
- [in] *size* - The size of the structure pointed to by the params pointer, in bytes.

Returns: An image reference `vx_image`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`

3.4.3. vxCopyImagePatchWithFlags

Allows the application to copy a rectangular patch from/into an image object plane.

```
vx_status vxCopyImagePatchWithFlags(
    vx_image                                image,
    const vx_rectangle_t*                   image_rect,
    vx_uint32                               image_plane_index,
    const vx_imagepatch_addressing_t*       user_addr,
    void*                                   user_ptr,
    vx_enum                                 usage,
    vx_enum                                 user_mem_type,
    vx_uint32                               flags);
```

Parameters

- [in] *image* - The reference to the image object that is the source or the destination of the copy.
- [in] *image_rect* - The coordinates of the image patch. The patch must be within the bounds of the image. (*start_x*, *start_y*) gives the coordinates of the topleft pixel inside the patch, while (*end_x*, *end_y*) gives the coordinates of the bottomright element out of the patch. Must be 0 ≤ start < end ≤ number of pixels in the image dimension.
- [in] *image_plane_index* - The plane index of the image object that is the source or the destination of the patch copy.
- [in] *user_addr* - The address of a structure describing the layout of the user memory location pointed by user_ptr. In the structure, only *dim_x*, *dim_y*, *stride_x* and *stride_y* fields must be provided, other fields are ignored by the function. The layout of the user memory must follow a row major order: *stride_x* = pixel size in bytes, and *stride_y* ≥ *stride_x* * *dim_x* .
- [in] *user_ptr* - The address of the memory location where to store the requested data if the copy was requested in read mode, or from where to get the data to store into the image object if the copy was requested in write mode. The accessible memory must be large enough to contain the specified patch with the specified layout: accessible memory in bytes ≥ (*end_y* - *start_y*) * *stride_y*.
- [in] *usage* - This declares the effect of the copy with regard to the image object using the `vx_accessor_e` enumeration. For uniform images, only `VX_READ_ONLY` is supported. For other images, only `VX_READ_ONLY` and `VX_WRITE_ONLY` are supported:
  - `VX_READ_ONLY` means that data is copied from the image object into the application memory.
  - `VX_WRITE_ONLY` means that data is copied into the image object from the application memory.

- [in] *user_mem_type* - A vx_memory_type_e enumeration that specifies the memory type of the memory referenced by the user_addr.
- [in] *flags* - An integer that allows passing options to the copy operation.

Returns: A vx_status_e enumeration.

Return Values

- VX_SUCCESS - No errors; any other value indicates failure.
- VX_ERROR_OPTIMIZED_AWAY - This is a reference to a virtual image that cannot be accessed by the application.
- VX_ERROR_INVALID_REFERENCE - image is not a valid vx_image reference.
- VX_ERROR_INVALID_PARAMETERS - An other parameter is incorrect.
- VX_ERROR_NO_MEMORY - Internal memory allocation failed.

Note

If the application asks for data outside the valid region, the returned values are implementation-defined.

Note

When copying data from/to VX_DF_IMAGE_U1 images the bit offsets for pixels are preserved. It's not necessary for the coordinates of the image patch to start and end at byte boundaries. In that case, when copying data *to* an image only the pixels inside the specified image patch will be written to and when copying *from* an image the resulting padding pixels at the start and/or end of the patch are implementation-defined.

Index

R

Raw Image API
- VX_DF_IMAGE_RAW, 9
- VX_ENUM_IMAGE_RAW_BUFFER_ACCESS, 9
- VX_ENUM_IMAGE_RAW_EXPOSURE_INTERLEAVING, 10
- VX_ENUM_IMAGE_RAW_PIXEL_CONTAINER, 9
- VX_IMAGE_IS_RAW, 8
- vx_image_raw_buffer_access_e, 11
- vx_image_raw_create_params_t, 10
- VX_IMAGE_RAW_EXPOSURE_INTERLEAVING, 9
- vx_image_raw_exposure_interleaving_e, 11
- VX_IMAGE_RAW_FORMAT, 9
- vx_image_raw_format_t, 10
- VX_IMAGE_RAW_MAX_EXPOSURES, 8
- VX_IMAGE_RAW_META_HEIGHT_AFTER, 9
- VX_IMAGE_RAW_META_HEIGHT_BEFORE, 9
- vx_image_raw_pixel_container_e, 11
- vxCopyImagePatchWithFlags, 13
- vxCreateRawImage, 12
- vxCreateVirtualRawImage, 12