**OpenVX Functions and Objects**

**VX_API_CALL**

In every OpenVX function and object, insert VX_API_CALL before the function name, as shown in the example below. It was omitted from the functions on this reference card to save space.

```c
<return_type> VX_API_CALL vxFunctionName([T arg1, . . . , T argN]);
```

**Vision Functions**

Vision functions in OpenVX may be graph mode or immediate mode.

- **Graph mode functions** have "Node" in the function name. They may be created and linked together, verified by the implementation, then executed as often as needed.
- **Immediate mode functions** are executed on a context immediately, if as were single node graphs, with no leaking side-effects.

In the vision functions, the parameter `graph` is the reference to the graph, and the parameter `context` is the reference to the overall context.

---

**Vision Functions**

**Absolute Difference**

```c
vx_status vx_node vxAbsDiffNode(vx_graph graph, vx_image in1,
                                 vx_image in2, vx_image out);
```

**Accumulate**

```c
vx_status vxuAccumulateImage(vx_graph graph, vx_image input,
                             vx_image output, vx_scalar alpha, vx_enum &accum,
                             vx_enum &accum_policy, vx_image &accum);```

**Accumulate Squared**

```c
vx_status vxuAccumulateSquaredImage(vx_graph graph, vx_image input,
                                     vx_scalar shift, vx_image output);```

**Accumulate Weighted**

```c
vx_status vxuAccumulateWeightedImage(vx_graph graph, vx_image input,
                                     vx_scalar alpha, vx_image output,
                                     vx_scalar shift, vx_image output);```

**Arithmetic Addition**

```c
vx_status vxuAddNode(vx_graph graph, vx_image in1,
                     vx_image in2, vx_image output, vx_enum &policy,
                     vx_image output);```

**Arithmetic Subtraction**

```c
vx_status vxuSubtractNode(vx_graph graph, vx_image in1,
                          vx_image in2, vx_image output, vx_enum &policy,
                          vx_image output);```

**Bitwise AND**

```c
vx_status vxuAndNode(vx_graph graph, vx_image in1,
                     vx_image in2, vx_image output);```

**Bitwise EXCLUSIVE OR**

```c
vx_status vxuXorNode (vx_graph graph, vx_image in1,
                     vx_image in2, vx_image output);```

**Channel Extract**

```c
vx_status vxuExtractNode (vx_graph graph, vx_image input,
                          vx_image output, vx_array channels, vx_enum &mode);
```

**Channel Combine**

```c
vx_status vxuCombineNode (vx_graph graph, vx_image in1,
                          vx_image in2, vx_image output);```

**Convert Bit Depth**

```c
vx_status vxConvolveDepth (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_scalar shift);
```

**Convert Color**

```c
vx_status vxuColorConvert (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_enum int32_shift);
```

**Convert Size**

```c
vx_status vxConvolveSize (vx_graph graph, vx_image input,
                         vx_image output, vx_enum policy, vx_uni32_size);
```

**Custom Convolution**

```c
vx_status vxConvolve3x3 (vx_graph graph, vx_image input,
                        vx_image output, vx_array filter);
```

**Dilate Image**

```c
vx_status vxuDilate3x3 (vx_graph graph, vx_image input,
                       vx_image output, vx_u8_image struct kernel);
```

**Equalize Histogram**

```c
vx_status vxuEqualizeHist (vx_graph graph, vx_image input,
                          vx_image output, vx_u8_image struct kernel);
```

**Erode Image**

```c
vx_status vxuErode3x3 (vx_graph graph, vx_image input,
                     vx_image output, vx_u8_image struct kernel);
```

**Extract Channel**

```c
vx_status vxuExtractChannel (vx_graph graph, vx_image input,
                            vx_image output, vx_array channels, vx_enum &mode);
```

**Fast Corners**

```c
vx_status vxuFastCorners (vx_graph graph, vx_image input,
                        vx_u16_image struct kernel);
```

**Fast Corners**

```c
vx_status vxuFastCorners (vx_graph graph, vx_image input,
                        vx_scalar strength_threshold, vx_u16_image struct kernel);
```

**Grows the white space in a VX_DF_IMAGE_U8 Boolean image.**

```c
vx_status vxuDilate3x3Node (vx_graph graph, vx_image input,
                           vx_image output, vx_u8_image struct kernel);
```

**Canny Edge Detector**

```c
vx_status vxuCannyEdgeDetectorNode (vx_graph graph, vx_image input,
                                    vx_image output, vx_u8_image struct kernel);
```

**Channel to extract.**

```c
vx_status vxuChannelExtractNode (vx_graph graph, vx_image input,
                                 vx_image output, vx_array channels, vx_enum &mode);
```

**Channel to extract.**

```c
vx_status vxuChannelCombineNode (vx_graph graph, vx_image input,
                                 vx_image output, vx_array channels, vx_enum &mode);
```

**Channel Combine**

```c
vx_status vxuCombineNode (vx_graph graph, vx_image input,
                         vx_image output, vx_array channels, vx_enum &mode);
```

**Convert Bit Depth**

```c
vx_status vxConvolveDepth (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_scalar shift);
```

**Convert Color**

```c
vx_status vxuColorConvert (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_enum int32_shift);
```

**Convert Size**

```c
vx_status vxConvolveSize (vx_graph graph, vx_image input,
                         vx_image output, vx_enum policy, vx_uni32_size);
```

**Custom Convolution**

```c
vx_status vxConvolve3x3 (vx_graph graph, vx_image input,
                        vx_image output, vx_array filter);
```

**Dilate Image**

```c
vx_status vxuDilate3x3 (vx_graph graph, vx_image input,
                       vx_image output, vx_u8_image struct kernel);
```

**Equalize Histogram**

```c
vx_status vxuEqualizeHist (vx_graph graph, vx_image input,
                          vx_image output, vx_u8_image struct kernel);
```

**Erode Image**

```c
vx_status vxuErode3x3 (vx_graph graph, vx_image input,
                     vx_image output, vx_u8_image struct kernel);
```

**Fast Corners**

```c
vx_status vxuFastCorners (vx_graph graph, vx_image input,
                        vx_u16_image struct kernel);
```

**Grows the white space in a VX_DF_IMAGE_U8 Boolean image.**

```c
vx_status vxuDilate3x3Node (vx_graph graph, vx_image input,
                           vx_image output, vx_u8_image struct kernel);
```

**Canny Edge Detector**

```c
vx_status vxuCannyEdgeDetectorNode (vx_graph graph, vx_image input,
                                    vx_image output, vx_u8_image struct kernel);
```

**Channel to extract.**

```c
vx_status vxuChannelExtractNode (vx_graph graph, vx_image input,
                                 vx_image output, vx_array channels, vx_enum &mode);
```

**Channel Combine**

```c
vx_status vxuCombineNode (vx_graph graph, vx_image input,
                         vx_image output, vx_array channels, vx_enum &mode);
```

**Convert Bit Depth**

```c
vx_status vxConvolveDepth (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_scalar shift);
```

**Convert Color**

```c
vx_status vxuColorConvert (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_enum int32_shift);
```

**Convert Size**

```c
vx_status vxConvolveSize (vx_graph graph, vx_image input,
                         vx_image output, vx_enum policy, vx_uni32_size);
```

**Custom Convolution**

```c
vx_status vxConvolve3x3 (vx_graph graph, vx_image input,
                        vx_image output, vx_array filter);
```

**Dilate Image**

```c
vx_status vxuDilate3x3 (vx_graph graph, vx_image input,
                       vx_image output, vx_u8_image struct kernel);
```

**Equalize Histogram**

```c
vx_status vxuEqualizeHist (vx_graph graph, vx_image input,
                          vx_image output, vx_u8_image struct kernel);
```

**Erode Image**

```c
vx_status vxuErode3x3 (vx_graph graph, vx_image input,
                     vx_image output, vx_u8_image struct kernel);
```

**Fast Corners**

```c
vx_status vxuFastCorners (vx_graph graph, vx_image input,
                        vx_u16_image struct kernel);
```

**Grows the white space in a VX_DF_IMAGE_U8 Boolean image.**

```c
vx_status vxuDilate3x3Node (vx_graph graph, vx_image input,
                           vx_image output, vx_u8_image struct kernel);
```

**Canny Edge Detector**

```c
vx_status vxuCannyEdgeDetectorNode (vx_graph graph, vx_image input,
                                    vx_image output, vx_u8_image struct kernel);
```

**Channel to extract.**

```c
vx_status vxuChannelExtractNode (vx_graph graph, vx_image input,
                                 vx_image output, vx_array channels, vx_enum &mode);
```

**Channel Combine**

```c
vx_status vxuCombineNode (vx_graph graph, vx_image input,
                         vx_image output, vx_array channels, vx_enum &mode);
```

**Convert Bit Depth**

```c
vx_status vxConvolveDepth (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_scalar shift);
```

**Convert Color**

```c
vx_status vxuColorConvert (vx_graph graph, vx_image input,
                          vx_image output, vx_enum policy, vx_enum int32_shift);
```

**Convert Size**

```c
vx_status vxConvolveSize (vx_graph graph, vx_image input,
                         vx_image output, vx_enum policy, vx_uni32_size);
```
Vision Functions (cont.)

Gaussian Filter [3.23]
Computes a Gaussian filter over a window of the input image.

vx_node vxuGaussian3x3Node (vx_graph graph, vx_image input, vx_image output);
ux_status vxuGaussian3x3 (vx_context context, vx_image input, vx_image output);
function: Computes a 2D Gaussian filter using a 3x3 kernel.

vx_node vxuGaussian5x5Node (vx_graph graph, vx_image input, vx_image output);
ux_status vxuGaussian5x5 (vx_context context, vx_image input, vx_image output);
function: Computes a 2D Gaussian filter using a 5x5 kernel.

Non-linear Filter [3.26]
Computes a non-linear filter over a window of the input image.

vx_node vxuNonLinearFilterNode (vx_graph graph, vx_uint32 mask, vx_image input, vx_image output);
ux_status vxuNonLinearFilter (vx_context context, vx_uint32 mask, vx_image input, vx_image output);
function: The non-linear function of type VX_NONLINEAR_FILTER (MEDIAN, MIN, MAX).

Output Image

vx_node vxuImageOutputNode (vx_graph graph, vx_image input, vx_image output);
ux_status vxuImageOutput (vx_context context, vx_image input, vx_image output);
function: Input the pixel to an output image of VX_DF_IMAGE_U8 format.

Magnitude [3.31]
Calculates the magnitude of the input image.

vx_node vxuMagnitudeNode (vx_graph graph, vx_image input, vx_image output);
ux_status vxuMagnitude (vx_context context, vx_image input, vx_image output);
function: Input the pixel of VX_DF_IMAGE_U8 format.

Mean and Standard Deviation [3.32]
Calculates the mean and standard deviation of the input pixels.

vx_node vxuMeanStdDevNode (vx_graph graph, vx_image input, vx_image output, vx_scalar mean, vx_scalar stddev);
ux_status vxuMeanStdDev (vx_context context, vx_image input, vx_image output, vx_scalar mean, vx_scalar stddev);
function: Input the mean and standard deviation of the pixels of VX_DF_IMAGE_U8 format.

Harris Corners [3.25]
Computes the Harris Corners of an image.

vx_node vxuHarrisCornersNode (vx_graph graph, vx_image input, vx_scalar strength, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 grad2_size, vx_array x, vx_array y, vx_matrix num_corners);
ux_status vxuHarrisCorners (vx_context context, vx_image input, vx_scalar strength, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 grad2_size, vx_array x, vx_array y, vx_matrix num_corners);
function: Input an image of VX_DF_IMAGE_U8 format. strength: The minimum threshold of type VX_TYPE_FLOAT32 with which to eliminate Harris corner scores. min_distance: The radial Euclidean distance of type VX_TYPE_FLOAT32 for non-maximum suppression. sensitivity: The scalar sensitivity threshold of type VX_TYPE_FLOAT32. grad2_size: The gradient window size to use on the input image. x, y: The coordinates of detected corners in image (type VX_TYPE_COORDINATES2D). num_corners: The array of objects of type VX_TYPE_KEYPOINT.

Histogram [3.26]
Generates a distribution from an image.

vx_node vxuHistogramNode (vx_graph graph, vx_image input, vx_int32 bin_width, vx_image output);
ux_status vxuHistogram (vx_context context, vx_image input, vx_int32 bin_width, vx_image output);
function: Input the image of VX_DF_IMAGE_U8 format. bin_width: The block size of the histogram window is used to compute the Harris corner score.

Gaussian Image Pyramid [3.27]
Computes a Gaussian pyramid of an image using a 5x5 kernel.

vx_node vxuGaussianPyramidNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 pyramid_gaussian);
ux_status vxuGaussianPyramid (vx_context context, vx_image input, vx_image output, vx_uint32 pyramid_gaussian);
function: Input the image of VX_DF_IMAGE_U8 format. pyramid_gaussian: The pyramid of type VX_TYPE_KEYPOINT.

Laplacian Image Pyramid [3.28]
Computes a Laplacian Image Pyramid from an input image.

vx_node vxuLaplacianPyramidNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 pyramid_laplacian);
ux_status vxuLaplacianPyramid (vx_context context, vx_image input, vx_image output, vx_uint32 pyramid_laplacian);
function: Input the image of VX_DF_IMAGE_U8 format. pyramid_laplacian: The Laplacian pyramid of type VX_TYPE_IMAGE_U8 format.

Reconstruction from Laplacian Image Pyramid [3.29]
Reconstructs the original image from a Laplacian Image Pyramid.

vx_node vxuLaplacianReconstructNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 pyramid_laplacian);
ux_status vxuLaplacianReconstruct (vx_context context, vx_image input, vx_image output, vx_uint32 pyramid_laplacian);
function: Input the lowest-resolution image of VX_DF_IMAGE_U8 format to reconstruct the input image from the pyramid.

Integral Image [3.30]
Computes the integral image of the input image.

vx_node vxuIntegralImageNode (vx_graph graph, vx_image input, vx_image output);
ux_status vxuIntegralImage (vx_context context, vx_image input, vx_image output);
function: Input the input image of VX_DF_IMAGE_U8 format. Output: The output image of VX_DF_IMAGE_U8 format.

Optical Flow Pyramid [3.35]
Computes optical flow over a pyramid of images.

vx_node vxuOpticalFlowPyrtXNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 kernel_size);
ux_status vxuOpticalFlowPyrtX (vx_context context, vx_image input, vx_image output, vx_uint32 kernel_size);
function: Input the image of VX_DF_IMAGE_U8 format. kernel_size: The block size of the pyramid window is used to compute the optical flow over the pyramid.

Sobel filter [3.40]
Computes the Sobel filter over an input image and output planes.

vx_node vxuSobel3x3Node (vx_graph graph, vx_image input, vx_image output, vx_uint32 window_size);
ux_status vxuSobel3x3 (vx_context context, vx_image input, vx_image output, vx_uint32 window_size);
function: Input the image of VX_DF_IMAGE_U8 format. window_size: The block size of the Sobel filter window is used to compute the Sobel filter over the input image.

Table Lookup [3.41]
Uses a LUT to convert pixel values.

vx_node vxuTableLookupNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 lut_width, vx_image output);
ux_status vxuTableLookup (vx_context context, vx_image input, vx_image output, vx_uint32 lut_width, vx_image output);
function: Input the image of VX_DF_IMAGE_U8 format. lut_width: The LUT of type VX_TYPE_UINT32 format.

Thresholding [3.42]
Produces a Boolean image by thresholding the input image.

vx_node vxuThresholdNode (vx_graph graph, vx_image input, vx_image output, vx_uint32 threshold, vx_image output);
ux_status vxuThreshold (vx_context context, vx_image input, vx_image output, vx_uint32 threshold, vx_image output);
function: Input the image of VX_DF_IMAGE_U8 format. threshold: A threshold to apply to the input image.

Warp Affine [3.43]
Performs an affine transform on an image.

vx_node vxuWarpAffineNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_image output);
ux_status vxuWarpAffine (vx_context context, vx_image input, vx_matrix matrix, vx_image output);
function: Input the image of VX_DF_IMAGE_U8 format. matrix: The affine matrix. Must be of 2x3 of type VX_TYPE_FLOAT32.

(Continued on next page)
Warp Perspective [3.44]

Performs a perspective transform on an image.

vx_node vxWarpPerspectiveNode (vx_graph graph, vx_image input, vx_matrix matrix, vx_enum type, vx_image output);

vx_status vxWarpPerspective (vx_context context, vx_image input, vx_matrix matrix, vx_enum type, vx_image output);

input: Image of VX_DF_IMAGE_U8 format.
matrix: Perspective matrix. Must be 3x3 of type VX_TYPE_FLOAT32.
type: The interpolation type.
VX_INTERPOLATION_NEAREST_NEIGHBOR, BILINEAR.

Set Status [3.45.6]

Returns status values from Object constructors if they fail.

vx_status vxGetStatus (vx_reference reference);
reference: The reference to check for construction errors.

Context Objects [3.48]

Attributes: enum vx_context_attribute_e:
VX_CONTEXT_VENDOR_ID, UNIQUE_KERNELS, MODULES, REFERENCES, IMPLEMENTATION, EXTENSIONS, SIZE, UNIQUE_KERNEL_TABLE, IMMEDIATE_BUILDER_POLICY, (CONVOLUTION, NONLINEAR_MAX_DIMENSION, OPTICAL_FLOW, MAX_WINDOW_DIMENSION)

Create a vx_context.

vx_context vxCreateContext ();

Retrieve the context from any reference from within a context.

vx_context vxGetContext (vx_reference reference);
reference: The reference from which to extract the context.

Query the context for some specific information.

vx_status vxQueryContext (vx_context context, vx_enum attribute, void *ptr, vx_size size);
attribute: Attribute to query of type vx_context_attribute_e.
ptr: Pointer to where to store the result.
size: Size of the container to which ptr points.

Release the OpenVX object context.

vx_status vxReleaseContext (vx_context *context);

Set an attribute on the context.

vx_status vxSetContextAttribute (vx_context context, vx_enum attribute, const void *ptr, vx_size size);
attribute: Attribute to set of type vx_context_attribute_e.
ptr: Pointer to the data to which to set the attribute.
size: Size in bytes of the container to which ptr points.

Sets the default target of the immediate mode.

vx_status vxSetImmediateModeTarget (vx_context context, vx_enum target_enum, const char *target_string);
target_enum: Default immediate mode target enum to be set to vx_context니다.
TARGET_MUST_STRING, VENDOR_BEGIN);
target_string: The target name ASCII string.

Array Objects [3.51]

Attributes: enum vx_array_attribute_e:
VX_ARRAY_ITEMTYPE, NUMITEMS, CAPACITY, ITEMSIZE

Add items to the Array.

vx_status vxAddArrayItems (vx_array arr, vx_size count, const void *ptr, vx_size stride);
arr: The reference to the array.
count: The total number of elements to insert.
ptr: Location from which to read the input values.
stride: The stride in bytes between elements.

Create a reference to an Array object.

vx_array vxCreateArray (vx_context context, vx_enum item_type, vx_size capacity);
context: The reference to the overall Context.
item_type: The type of objects to hold.
capacity: The maximum number of items that the array can hold.

Delay Objects [3.67]

Attributes: enum vx_delay_attribute_e:
VX_DELAY_TYPE, SLOTS

Age the internal delay ring by one.

vx_status vxStatusAgeDelay (vx_delay delay);
delay: The pointer to the delay object.

Create a Delay object.

vx_delay vxCreateDelay (vx_context context, vx_reference exemplar, vx_size slots);
exemplar: The exemplar object.
slots: The number of reference in the delay.

Retrieve a reference from a delay object.

vx_status vxGetReferenceFromDelay (vx_delay delay, vx_int32 index);
delay: The reference to the delay object.
index: The index into the delay from which to extract the reference.

Query a vx_delay object attribute.

vx_status vxQueryDelay (vx_delay delay, vx_enum attribute, void *ptr, vx_size size);
attribute: An attribute from vx_array_attribute_e.
ptr: Location at which to store the result.
size: The size in bytes of the container to which ptr points.

Release a reference of an Array object.

vx_status vxReleaseArray (vx_array *arr);
arr: The Array to release.

Unmap and commit potential changes to an array object range.

vx_status vxUnmapArrayRange (vx_array array, vx_map id *map_id);
array: The reference to the array object.
map_id: The unique map identifier returned by vxMapArrayRange.

Object: Array (Advanced) [3.64]

Registers user-defined structures to the context.

vx_enum vxRegisterStruct (vx_context context, vx_size size);
size: The size of user struct in bytes.

Distribution Objects [3.53]

Attributes: enum vx_distribution_attribute_e:
VX_DISTRIBUTION_DIMENSIONS, OFFSET, RANGE, BINS, WINDOW, SIZE

Allows the application to copy from/from a distribution object.

vx_status vxCopyDistribution (vx_distribution distribution, void *user_ptr, vx_enum usage, vx_enum user_mem_type);
distribution: The reference to the source/destination distribution object.
user_ptr: The address of the memory location at which to store the distribution.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: VX_MEMORY_TYPE_NONE, HOST.

Create a reference to a Data Distribution.

vx_distribution vxCreateDistribution (vx_context context, vx_size num_bins, vx_int32 offset, vx_int32 range);
context: The reference to the overall context.
um_bins: The number of bins in the distribution.
offset: The start offset into the range value.
rnage: Total number of consecutive values of the distribution interval.

(Continued on next page)
**Distribution Objects (cont.)**

Allows application direct access to the distribution object.

```c
vx_status vxMapDistribution(vx_distribution *distribution, const vx_map_id *map_id, void **ptr);
```

- `vx_distribution`: The reference to the distribution object.
- `const vx_map_id *map_id`: Map ID.
- `void **ptr`: Pointer to a caller-owned array in which the requested data can be accessed.

Usage:
- VX_READONLY, VX_WRITEONLY, or VX_READANDWRITE.

Returns a Boolean to indicate the state of graph verification.

```c
vx_status vxValidateGraph(vx_graph *graph);
```

- `vx_graph`: The graph to verify.

The pointer to the parent image.

```c
vx_status vxCreateImageFromHandle(vx_image *image, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The base pointer.
- `vx_size size`: The size in bytes of the container to which `ptr` points.

Create a sub-image from a single plane channel of another image.

```c
vx_status vxCreateImageFromChannel(vx_image *image, const vx_image_channel *channel);
```

- `vx_image`: The pointer to the parent image.
- `const vx_image_channel *channel`: The channel on which to wait.

Create a opaque reference to an image buffer.

```c
vx_status vxMapImagePatch(vx_image *image, void *ptr, const vx_imagepatch_addressing_t *addr, void *user_ptr, vx_enum mem_type, vx_enum user_mem_type);
```

- `vx_image`: The pointer to the image from which to extract the patch.
- `void *ptr`: The location at which to store the result.
- `const vx_imagepatch_addressing_t *addr`: The plane index from which to get the data.
- `void *user_ptr`: The address of a structure describing the layout of the user memory location pointed by `user_ptr`.
- `vx_enum mem_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image.
- `vx_enum user_mem_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Create a sub-image from a single plane channel of another image.

```c
vx_status vxCreateImageFromChannel(vx_image *image, const vx_image_channel *channel);
```

- `vx_image`: The pointer to the parent image.
- `const vx_image_channel *channel`: The channel on which to wait.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

Create a reference to an externally allocated image object.

```c
vx_status vxImageFromHandle(vx_image *image, void *ptr, const vx_image_mem_type *memory_type);
```

- `vx_image`: The pointer to the parent image.
- `void *ptr`: The location from which to read the value.
- `const vx_image_mem_type *memory_type`: The VX_DF_IMAGE (vx_df_image_e) code that represents the format of the image and the color space.

Copy a rectangular patch from/into an image object.

```c
vx_status vxImagepatchAddress1d(vx_image *image, const vx_imagepatch_addressing_1d *addr, void *ptr, vx_size size);
```

- `vx_image`: The pointer to the parent image.
- `const vx_imagepatch_addressing_1d *addr`: Address of variable where function returns a map identifier.
- `void *ptr`: The reference to the parent image.
- `vx_size size`: The size of the container to which `ptr` points.

Create an image from another image given a rectangular patch.

```c
vx_status vxCreateImageFromROI(vx_image *image, vx_rectangle_t *rect);
```

- `vx_image`: The pointer to the parent image.
- `vx_rectangle_t *rect`: The region of interest.

vx_status vxGetKernelByEnum (vx_context context, vx_enum kernel);
context: The reference to the implementation context.
kernel: The kernel to query.

vx_status vxQueryKernel (vx_kernel kernel, vx_enum attribute, void **ptr, vx_size size);
attribute: The attribute to query. From vx_kernel_attribute_e.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference to a kernel.

vx_status vxReleaseKernel (vx_kernel *kernel);

Kernel: The pointer to the kernel reference to release.

Matrix Objects [3.66] Attributes: enum vx_matrix_attribute_e: VX_MATRIX_{TYPE, ROWS, COLUMNS, SIZE, ORIGIN, PATTERN} Copy from or to a matrix data.

vx_status vxCopyMatrix (vx_matrix matrix, void *user_ptr, vx_enum usage, vx_enum user_mem_type);
matrix: The reference to the matrix object.
user_ptr: The location at which to store the resulting value.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: VX_MEMORY_TYPE_{NONE, HOST}.

Creates a reference to a matrix object.

vx_matrix vxCreateMatrix (vx_context context, vx_enum data_type, vx_size columns, vx_size rows);
context: The string of the name of the kernel to get.

Query the kernel to get information about the number of parameters, enum values, etc.

vx_status vxQueryKernel (vx_kernel kernel, vx_enum attribute, void **ptr, vx_size size);
attribute: The reference to the implementation context.
kernel: The kernel reference to query.

Release the reference to the kernel.

vx_status vxReleaseKernel (vx_kernel *kernel);

Node Objects [3.60] Attributes: enum vx_node_attribute_e: VX_NODE_{STATUS, PERFORMANCE, BORDER, LOCAL_DATA_SIZE, SIZE, PTR, PARAMETERS, IR_replace, REPLICATE_FLAGS, VALID_RECT_RESET} Queries information out of a node.

vx_status vxQueryNode (vx_node node, vx_enum attribute, void **ptr, vx_size size);
attribute: The reference to the node to query.
ptr: The location at which to store the resulting value.
size: The size in bytes of the objects to which ptr points.

Release a reference to a node object.

vx_status vxReleaseNode (vx_node *node);
node: The reference to the node to remove.

ObjectArray Objects [3.61] Attributes: enum vx_object_array_attribute_e: VX_OBJECTARRAY_{ITEMTYPE, NUMITEMS} Creates a reference to an ObjectArray of count objects.

vx_object_array vxCreateObjectArray (vx_context context, vx_reference exemplary, vx_size size_count);
context: The reference to the overall Context.
exemplary: The exemplar object that defines the metadata of the created objects in the ObjectArray.
size_count: The number of Objects to create in the ObjectArray.

Creates an opaque reference to a virtual ObjectArray.

vx_object_array vxCreateVirtualObjectArray (vx_graph graph, vx_reference exemplary, vx_size size_count);
graph: The reference to the graph in which to create the ObjectArray.
exemplary: The exemplar object that defines the type of object in the ObjectArray.
size_count: The number of Objects to create in the ObjectArray.

Retrieves the reference to the OpenVX Object.

vx_reference vxGetObjectArrayItem (vx_object_array arr, vx_uint32 index);
arr: The ObjectArray.
index: The index of the object in the ObjectArray.

Queries an attribute from the ObjectArray.

vx_status vxQueryObjectArray (vx_object_array arr, vx_enum attribute, void **ptr, vx_size size);
arr: The reference to the ObjectArray.
attribute: VX_OBJECTARRAY_{ITEMTYPE, NUMITEMS}.
ptr: The location at which to store the resulting value.
size: The size in bytes of the container to which ptr points.

Release a reference to an ObjectArray Object.

vx_status vxReleaseObjectArray (vx_object_array *arr);
arr: The pointer to the ObjectArray to release.


vx_status vxGetParameterByIndex (vx_kernel kernel, vx_uint32 index);
kern: The reference to the kernel.
index: The index of the parameter.

Retrieve a vx_parameter from a vx_node.

vx_status vxGetParameterByIndex (vx_node node, vx_uint32 index);
ode: The node from which to extract the parameter.
index: The index of the parameter.

Query a parameter to determine its meta-information.

vx_status vxQueryParameter (vx_parameter param, vx_enum attribute, void **ptr, vx_size size);
param: The reference to the parameter.
attribute: The attribute to query from vx_parameter_attribute_e.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference to a parameter object.

vx_status vxReleaseParameter (vx_parameter *param);
param: The pointer to the parameter.
Pyramid Objects [3.57]
Attributes: enum vx_pyramid_attribute_e:
  VX_PYRAMID_LEVELS, SCALE, WIDTH, HEIGHT, FORMAT)
Create a reference to a pyramid object.

vx_pyramid vxCreatePyramid (vx_context context,
    vx_size levels, vx_float32 scale, vx_uint32 width,
    vx_uint32 height, vx_df_image_format);
context: The reference to the overall context.
levels: The number of levels desired.
scale: This must be a non-zero positive value.
width: The width of the level image in pixels.
height: The height of the level image in pixels.
format: Format of all images in the pyramid or VX_DF_IMAGE_VIRT.

Create a reference to a virtual pyramid object.

vx_pyramid vxCreateVirtualPyramid (vx_graph graph,
    vx_size levels, vx_float32 scale, vx_uint32 width,
    vx_uint32 height, vx_df_image_format);
graph: The reference to the parent graph.
levels: The number of levels desired.
scale: This must be a non-zero positive value.
width: The width of the level image in pixels.
height: The height of the level image in pixels.
format: Format of all images in the pyramid or VX_DF_IMAGE_VIRT.

Remap Objects [3.58]
Attributes: enum vx_remap_attribute_e: VX_REMAP_
  [SOURCE_WIDTH, HEIGHT], [DESTINATION_WIDTH, HEIGHT])
Create a remap table object.

vx remap vxCreateRemap (vx_context context,
    vx_uint32 src_width, vx_uint32 src_height,
    vx_uint32 dst_width, vx_uint32 dst_height);
context: The reference to the overall context.
src_width, src_height: The width, height of the source image in pixels.
dst_width, dst_height: The width, height of destination image in pixels.

Retrieve the source pixel point from a destination pixel.

vx_status vxGetRemapPoint (vx_remap remap, vx_uint32 dst_x,
    vx_uint32 dst_y, vx_float32 *src_x, vx_float32 *src_y);
remap: Remap table reference.
dst_x, dst_y: The destination {x, y} coordinate.
src_x, src_y: Pointer to location where to store the source coordinates in
  float representation to allow interpolation.

Query attributes from a remap table.

vx_status vxQueryRemap (vx_remap remap, vx_enum attribute,
    void *ptr, vx_size size);
attribute: An attribute from vx_remap_attribute_e.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference to a remap table object.

vx_status vxReleaseRemap (vx_remap *table);
table: A pointer to the remap table to release.

Assign a destination pixel mapping to the source pixel.

vx_status vxSetRemapPoint (vx_remap remap, vx_uint32 dst_x,
    vx_uint32 dst_y, vx_float32 src_x, vx_float32 src_y);
table: Remap table reference.
dst_x, dst_y: The destination {x, y} coordinate.
src_x, src_y: The source {x, y} coordinate in float to allow interpolation.

Scalor Objects [3.59]
Attribute: enum vx_scalar_attribute_e: VX_SCALAR_TYPE
Allows the application to copy from/into a scalar object.

vx_status vxCopyScalar (vx_scalar src, void *user_ptr,
    vx_enum usage, vx_enum user_mem_type);
src: The reference to the overall context.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: Type of the memory referenced by the user_addr.

vx_status vxCopyScalar (vx_scalar src, void *user_ptr,
    vx_enum usage, vx_enum user_mem_type);
src: The reference to the overall context.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: Type of the memory referenced by the user_addr.

vx_status vxCreateScalar (vx_context context,
    vx_enum data_type, const void *ptr);
context: The reference to the overall context.

data_type: The type of data.
ptr: Pointer to the local value.

Retrieve the source level of the pyramid as a vx_image.

vx_status vxGetPyramidLevel (vx_pyramid pyramid,
    vx_uint32 index);
pyramid: The pyramid.
index: The index of the level, such that index is less than levels.

Query an attribute from an image pyramid.

vx_status vxQueryPyramid (vx_pyramid pyramid,
    vx_enum attribute, void **ptr, vx_size size);
attribute: The attribute to query from vx_remap_attribute_e.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference to a pyramid object.

vx_status vxReleasePyramid (vx_pyramid *pyramid);
pyramid: The pointer to the pyramid to release.

Scalar Objects [3.59]
Attribute: enum vx_scalar_attribute_e: VX_SCALAR_TYPE
Allows the application to copy from/into a scalar object.

vx_status vxCopyScalar (vx_scalar src, void *user_ptr,
    vx_enum usage, vx_enum user_mem_type);
src: The reference to the overall context.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: Type of the memory referenced by the user_addr.

vx_status vxCopyScalar (vx_scalar src, void *user_ptr,
    vx_enum usage, vx_enum user_mem_type);
src: The reference to the overall context.
usage: VX_READ_ONLY or VX_WRITE_ONLY.
user_mem_type: Type of the memory referenced by the user_addr.

vx_status vxCreateScalar (vx_context context,
    vx_enum data_type, const void *ptr);
context: The reference to the overall context.

data_type: The type of data.
ptr: Pointer to the local value.

Retrieve the source level of the pyramid as a vx_image.

vx_status vxGetPyramidLevel (vx_pyramid pyramid,
    vx_uint32 index);
pyramid: The pointer to the pyramid.
index: The index of the level, such that index is less than levels.

Query an attribute from an image pyramid.

vx_status vxQueryPyramid (vx_pyramid pyramid,
    vx_enum attribute, void **ptr, vx_size size);
attribute: The attribute to query from vx_remap_attribute_e.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference to a pyramid object.

vx_status vxReleasePyramid (vx_pyramid *pyramid);
pyramid: The pointer to the pyramid to release.

Reference Objects [3.47]
Attributes: enum vx_reference_attribute_e:
  VX_REF_ATTRIBUTE [COUNT, TYPE, NAME]
Query any reference type for basic information (count, type).

vx_status vxReferenceQuery (vx_reference ref,
    vx_enum attribute, void *ptr, vx_size size);
ref: The reference to the object to query.
attribute: The value to query.
ptr: The location at which to store the resulting value.
size: The size of the container to which ptr points.

Release a reference.

vx_status vxReferenceRelease (vx_reference *ref_ptr);
ref_ptr: The pointer to the reference to the object to release.

Increments the reference counter of an object.

vx_status vxReferenceRetain (vx_reference ref);
ref: The reference to retain.

Name a reference.

vx_status vxSetReferenceName (vx_reference ref,
    const vx_char *name);
name: NULL if not named, or a pointer to NUL-terminated name.

Hints [3.74]
Provide a generic API to give platform-specific hints.

vx_status vxSetHint (vx_reference ref,
    vx_enum hint, void *data, vx_size data_size);
ref: The reference to the object to hint at.
hint: *vx_hint_e
data: Optional vendor specific data.
data_size: Size of the data structure data.

Directives [3.75]
Provide a generic API to give platform-specific directives.

vx_status vxSetDirective (vx_reference ref,
    vx_enum directive);
directive: The directive to set.

User Kernels [3.76]
Set the signatures of the custom kernel.

vx_status vxAddParameterToKernel (vx_kernel kernel,
    vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state);
kernel: The reference to the kernel.
dir: The index of the parameter to add.
data_type: Type of parameter, from vx_type_e (on page 8).
state: Parameter state. See vx_parameter_state_e.

(Continued on next page)
Advanced Framework (cont.)

Allows users to add custom kernels to the known kernel database.

Allocates/registers user-defined kernel enumeration to a context.

Allocates/registers user-defined kernel library ID to a context.

Called after parameters have been added and kernel is ready.

Enumerators

<table>
<thead>
<tr>
<th>vx_enum_e</th>
<th>vx_kernel_e</th>
<th>vx_accessor_e</th>
<th>vx_status_e</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX_ENUM_ACCESSOR</td>
<td>VX_KERNEL</td>
<td>VX_ACCESSOR</td>
<td>VX_STATUS_LOADKERNELS</td>
</tr>
<tr>
<td>VX_ENUM_ACTION</td>
<td>VX_ACCESSOR</td>
<td>VX_ACTION</td>
<td>VX_STATUS_VALIDATEKERNEL</td>
</tr>
<tr>
<td>VX_ENUM_BORDER</td>
<td>VX_ACCESSOR</td>
<td>VX_BORDER</td>
<td>VX_STATUS_REMOVEKERNEL</td>
</tr>
<tr>
<td>VX_ENUM_BORDER_POLICY</td>
<td>VX_ACCESSOR</td>
<td>VX_BORDER_POLICY</td>
<td>VX_STATUS_INITKERNEL</td>
</tr>
<tr>
<td>VX_ENUM_CHANNEL</td>
<td>VX_ACCESSOR</td>
<td>VX_CHANNEL</td>
<td>VX_STATUS_FINALIZEKERNEL</td>
</tr>
<tr>
<td>VX_ENUM_COLOR_RANGE</td>
<td>VX_ACCESSOR</td>
<td>VX_COLOR_RANGE</td>
<td>VX_STATUS_SETKERNELATTR</td>
</tr>
<tr>
<td>VX_ENUM_COLOR_SPACE</td>
<td>VX_ACCESSOR</td>
<td>VX_COLOR_SPACE</td>
<td>VX_STATUS_SETKERNELPARAM</td>
</tr>
<tr>
<td>VX_ENUM_CONVERT_POLICY</td>
<td>VX_ACCESSOR</td>
<td>VX_CONVERT_POLICY</td>
<td>VX_STATUS_VALIDATEPARAM</td>
</tr>
<tr>
<td>VX_ENUM_DIRECTION</td>
<td>VX_ACCESSOR</td>
<td>VX_DIRECTION</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_DIRECTIVE</td>
<td>VX_ACCESSOR</td>
<td>VX_DIRECTIVE</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_GRAPH_STATE</td>
<td>VX_ACCESSOR</td>
<td>VX_GRAPH_STATE</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_HINT</td>
<td>VX_ACCESSOR</td>
<td>VX_HINT</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_INTERPOLATION</td>
<td>VX_ACCESSOR</td>
<td>VX_INTERPOLATION</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_MEMORY_TYPE</td>
<td>VX_ACCESSOR</td>
<td>VX_MEMORY_TYPE</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_NONLINEAR</td>
<td>VX_ACCESSOR</td>
<td>VX_NON_LINEAR_FILTER</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_NORM_TYPE</td>
<td>VX_ACCESSOR</td>
<td>VX_NORM</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_PARAMETER_STATE</td>
<td>VX_ACCESSOR</td>
<td>VX_PARAMETER_STATE</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_PATTERN</td>
<td>VX_ACCESSOR</td>
<td>VX_PATTERN</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_ROUND_POLICY</td>
<td>VX_ACCESSOR</td>
<td>VX_ROUND</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_TARGET</td>
<td>VX_ACCESSOR</td>
<td>VX_TARGET</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_TERM_CRITERIA</td>
<td>VX_ACCESSOR</td>
<td>VX_TERM_CRITERIA</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
<tr>
<td>VX_ENUM_THRESHOLD_TYPE</td>
<td>VX_ACCESSOR</td>
<td>VX_THRESHOLD</td>
<td>VX_STATUS_SETMETAFORMATATTRIBUTE</td>
</tr>
</tbody>
</table>

Load one or more kernels into the OpenVX context.

vX_STATUS = VX_LOADKERNELS(vX_CONTEXT, const vx_kernel *, module);

custom vx_kernel *module;

custom The reference to the implementation context.

Remove a vx_kernel from the vx_context.

vX_STATUS = VX_REMOVEKERNEL(vX_KERNEL, kernel);
Image Access Example [2.15.2]

```c
void *base_ptr = NULL;
vx_uint32 width = 640, height = 480, plane = 0;
vx_image image = vxCreateImage(context, width, height, VX_DF_IMAGE_U8);
vx_rectangle_t rect;
vx_imagepatch_addressing_t addr;
rect.start_x = rect.start_y = 0;
rect.end_x = rect.end_y = PATCH_DIM;

status = vxAccessImagePatch(image, &rect, plane, &addr, &base_ptr, VX.READ_AND_WRITE);

if (status == VX_SUCCESS) {
    vx_uint32 x, y, i, j;
    vx_uint8 pixel = 0;

    /* addressing options */
    /* use linear addressing function/macro */
    for (i = 0; i < addr.dim_y*addr.dim_y; i++) {
        vx_uint8 *ptr2 = vxFormatImagePatchAddress1d(base_ptr, i, &addr);
        *ptr2 = pixel;
    }

    /* 2d addressing option */
    for (y = 0; y < addr.dim_y; y++) {
        for (x = 0; x < addr.dim_x; x++) {
            vx_uint8 *ptr2 = vxFormatImagePatchAddress2d(base_ptr, x, y, &addr);
            *ptr2 = pixel;
        }
    }

    /* direct addressing by client. For subsampled planes, scale will change. */
    for (y = 0; y < addr.dim_y; y++) {
        j = (addr.stride_y*y + addr.scale_y)/VX_SCALEUNITY;
        for (x = 0; x < addr.dim_x; x++) {
            vx_uint8 *tmp = (vx_uint8 *)base_ptr;
            i = j + (addr.stride_x*x + addr.scale_x)/VX_SCALEUNITY;
            tmp[i] = pixel;
        }
    }

    /* commits the data back to the image. If rect were 0 or empty, it would just decrement */
    status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
}
vxReleaseImage(&image);
```

### OpenVX Types (vx_type_e) [3.45.5]

- VX_TYPE_INVALID: Invalid type value.
- VX_TYPE_CHAR: vx_char
- VX_TYPE_INT8: vx_int8
- VX_TYPE_UINT8: vx_uint8
- VX_TYPE_INT16: vx_int16
- VX_TYPE_UINT16: vx_uint16
- VX_TYPE_INT32: vx_int32
- VX_TYPE_UINT32: vx_uint32
- VX_TYPE_INT64: vx_int64
- VX_TYPE_UINT64: vx_uint64
- VX_TYPE_FLOAT32: vx_float32
- VX_TYPE_FLOAT64: vx_float64
- VX_TYPE_KERNEL: vx_kernel
- VX_TYPE_VERTEX: vx_vertex
- VX_TYPE_NODE: vx_node
- VX_TYPE_PYRAMID: vx_pyramid
- VX_TYPE_DISTRIBUTION: vx_distribution
- VX_TYPE_IMAGE: vx_image
- VX_TYPE_REMAP: vx_remap
- VX_TYPE_MERGR: vx_mergr
- VX_TYPE_vXDF_IMAGE: vx_df_image
- VX_TYPE_VX_GRAPH: vx_graph
- VX_TYPE_VX_CONTEXT: vx_context
- VX_TYPE_VX_NODE: vx_node
- VX_TYPE_VX_KERNEL: vx_kernel
- VX_TYPE_VX_PARAMETER: vx_parameter
- VX_TYPE_VX_DELAY: vx_delay
- VX_TYPE_VX_LUT: vx_lut
- VX_TYPE_VX_MATRIX: vx_matrix
- VX_TYPE_VX_CONVOLUTION: vx_convolution
- VX_TYPE_VX_SCALAR: vx_scalar
- VX_TYPE_VX_ARRAY: vx_array
- VX_TYPE_VX_IMAGE: vx_image
- VX_TYPE_VX_REMAP: vx_remap
- VX_TYPE_VX_ERRDR: vx_errdr
- VX_TYPE_VX_META_FORMAT: vx_meta_format
- VX_TYPE_VX_OBJECT_ARRAY: vx_object_array