



The **OpenVX™** Specification

Version 1.0.1

Document Revision: r31169

Generated on Wed May 13 2015 08:41:43

Khronos Vision Working Group

Editor: Susheel Gautam
Editor: Erik Rainey

Copyright ©2014 The Khronos Group Inc.



Copyright ©2014 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be re-formatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group web-site should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

Khronos, DevU, StreamInput, glTF, WebGL, WebCL, COLLADA, OpenKODE, OpenVG, OpenVX, OpenGL ES and OpenMAX are trademarks of the Khronos Group Inc. ASTC is a trademark of ARM Holdings PLC, OpenCL is a trademark of Apple Inc. and OpenGL is a registered trademark and the OpenGL ES and OpenGL SC logos are trademarks of Silicon Graphics International used under license by Khronos. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

Contents

1	Introduction	2
1.1	Abstract	2
1.2	Purpose	2
1.3	Scope of Specification	2
1.4	Normative References	2
1.5	Version/Change History	3
1.6	Requirements Language	3
1.7	Typographical Conventions	3
1.7.1	Naming Conventions	3
1.8	Glossary and Acronyms	4
1.9	Acknowledgements	4
2	Design Overview	6
2.1	Software Landscape	6
2.2	Design Objectives	6
2.2.1	Hardware Optimizations	7
2.2.2	Hardware Limitations	7
2.3	Assumptions	7
2.3.1	Portability	7
2.3.2	Opacity	7
2.4	Object-Oriented Behaviors	7
2.5	OpenVX Framework Objects	7
2.6	OpenVX Data Objects	8
2.7	Error Objects	9
2.8	Graphs Concepts	9
2.8.1	Linking Nodes	9
2.8.2	Virtual Data Objects	9
2.8.3	Node Parameters	10
2.8.4	Graph Parameters	10
2.8.5	Execution Model	10
2.8.5	Asynchronous Mode	10
2.8.6	Graph Formalisms	10
2.8.7	Node Execution Independence	11
2.8.8	Verification	13
2.9	Callbacks	13
2.10	User Kernels	13
2.10.1	Parameter Validation	14
2.10.1	The Meta Format Object	14
2.10.1	Delta Rectangles	15
2.10.2	User Kernels Naming Conventions	15
2.11	Immediate Mode Functions	15
2.12	Base Vision Functions	15
2.12.1	Inputs	16
2.12.2	Outputs	18
2.13	Lifecycles	19
2.13.1	OpenVX Context Lifecycle	19
2.13.2	Graph Lifecycle	20

2.13.3	Data Object Lifecycle	20
	OpenVX Image Lifecycle	21
2.14	Host Memory Data Object Access Patterns	21
2.14.1	Matrix Access Example	21
2.14.2	Image Access Example	22
2.14.3	Array Access Example	23
2.15	Extending OpenVX	23
2.15.1	Extending Attributes	23
2.15.2	Vendor Custom Kernels	23
2.15.3	Vendor Custom Extensions	24
2.15.4	Hinting	24
2.15.5	Directives	24
2.16	Known Extensions to OpenVX	24
2.16.1	User Kernel Tiling	24
3	Module Documentation	25
3.1	Vision Functions	25
3.1.1	Detailed Description	25
3.2	Absolute Difference	28
3.2.1	Detailed Description	28
3.2.2	Function Documentation	28
	vxAbsDiffNode	28
	vxuAbsDiff	28
3.3	Accumulate	29
3.3.1	Detailed Description	29
3.3.2	Function Documentation	29
	vxAccumulateImageNode	29
	vxuAccumulateImage	29
3.4	Accumulate Squared	30
3.4.1	Detailed Description	30
3.4.2	Function Documentation	30
	vxAccumulateSquareImageNode	30
	vxuAccumulateSquareImage	30
3.5	Accumulate Weighted	32
3.5.1	Detailed Description	32
3.5.2	Function Documentation	32
	vxAccumulateWeightedImageNode	32
	vxuAccumulateWeightedImage	32
3.6	Arithmetic Addition	34
3.6.1	Detailed Description	34
3.6.2	Function Documentation	34
	vxAddNode	34
	vxuAdd	34
3.7	Arithmetic Subtraction	36
3.7.1	Detailed Description	36
3.7.2	Function Documentation	36
	vxSubtractNode	36
	vxuSubtract	36
3.8	Bitwise AND	38
3.8.1	Detailed Description	38
3.8.2	Function Documentation	38
	vxAndNode	38
	vxuAnd	38
3.9	Bitwise EXCLUSIVE OR	40
3.9.1	Detailed Description	40
3.9.2	Function Documentation	40
	vxXorNode	40
	vxuXor	40

3.10 Bitwise INCLUSIVE OR	42
3.10.1 Detailed Description	42
3.10.2 Function Documentation	42
vxOrNode	42
vxuOr	42
3.11 Bitwise NOT	44
3.11.1 Detailed Description	44
3.11.2 Function Documentation	44
vxNotNode	44
vxuNot	44
3.12 Box Filter	45
3.12.1 Detailed Description	45
3.12.2 Function Documentation	45
vxBox3x3Node	45
vxuBox3x3	45
3.13 Canny Edge Detector	46
3.13.1 Detailed Description	46
3.13.2 Enumeration Type Documentation	47
vx_norm_type_e	47
3.13.3 Function Documentation	47
vxCannyEdgeDetectorNode	47
vxuCannyEdgeDetector	48
3.14 Channel Combine	49
3.14.1 Detailed Description	49
3.14.2 Function Documentation	49
vxChannelCombineNode	49
vxuChannelCombine	49
3.15 Channel Extract	51
3.15.1 Detailed Description	51
3.15.2 Function Documentation	51
vxChannelExtractNode	51
vxuChannelExtract	51
3.16 Color Convert	53
3.16.1 Detailed Description	53
3.16.2 Function Documentation	55
vxColorConvertNode	55
vxuColorConvert	55
3.17 Convert Bit depth	57
3.17.1 Detailed Description	57
3.17.2 Function Documentation	57
vxConvertDepthNode	57
vxuConvertDepth	58
3.18 Custom Convolution	59
3.18.1 Detailed Description	59
3.18.2 Function Documentation	59
vxConvolveNode	59
vxuConvolve	60
3.19 Dilate Image	61
3.19.1 Detailed Description	61
3.19.2 Function Documentation	61
vxDilate3x3Node	61
vxuDilate3x3	61
3.20 Equalize Histogram	62
3.20.1 Detailed Description	62
3.20.2 Function Documentation	62
vxEqualizeHistNode	62
vxuEqualizeHist	62
3.21 Erode Image	63

3.21.1	Detailed Description	63
3.21.2	Function Documentation	63
vxErode3x3Node		63
vxuErode3x3		63
3.22	Fast Corners	64
3.22.1	Detailed Description	64
3.22.2	Segment Test Detector	64
3.22.3	Function Documentation	65
vxFastCornersNode		65
vxuFastCorners		65
3.23	Gaussian Filter	67
3.23.1	Detailed Description	67
3.23.2	Function Documentation	67
vxGaussian3x3Node		67
vxuGaussian3x3		67
3.24	Harris Corners	68
3.24.1	Detailed Description	68
3.24.2	Function Documentation	69
vxHarrisCornersNode		69
vxuHarrisCorners		70
3.25	Histogram	71
3.25.1	Detailed Description	71
3.25.2	Function Documentation	71
vxHistogramNode		71
vxuHistogram		71
3.26	Gaussian Image Pyramid	72
3.26.1	Detailed Description	72
3.26.2	Function Documentation	72
vxGaussianPyramidNode		72
vxuGaussianPyramid		72
3.27	Integral Image	74
3.27.1	Detailed Description	74
3.27.2	Function Documentation	74
vxIntegralImageNode		74
vxuIntegralImage		74
3.28	Magnitude	76
3.28.1	Detailed Description	76
3.28.2	Function Documentation	76
vxMagnitudeNode		76
vxuMagnitude		76
3.29	Mean and Standard Deviation	78
3.29.1	Detailed Description	78
3.29.2	Function Documentation	78
vxMeanStdDevNode		78
vxuMeanStdDev		78
3.30	Median Filter	80
3.30.1	Detailed Description	80
3.30.2	Function Documentation	80
vxMedian3x3Node		80
vxuMedian3x3		80
3.31	Min, Max Location	81
3.31.1	Detailed Description	81
3.31.2	Function Documentation	81
vxMinMaxLocNode		81
vxuMinMaxLoc		81
3.32	Optical Flow Pyramid (LK)	83
3.32.1	Detailed Description	83
3.32.2	Function Documentation	84

vxOpticalFlowPyrLKNode	84
vxuOpticalFlowPyrLK	85
3.33 Phase	87
3.33.1 Detailed Description	87
3.33.2 Function Documentation	87
vxPhaseNode	87
vxuPhase	87
3.34 Pixel-wise Multiplication	89
3.34.1 Detailed Description	89
3.34.2 Function Documentation	89
vxMultiplyNode	89
vxuMultiply	90
3.35 Remap	91
3.35.1 Detailed Description	91
3.35.2 Function Documentation	91
vxRemapNode	91
vxuRemap	91
3.36 Scale Image	93
3.36.1 Detailed Description	93
3.36.2 Function Documentation	95
vxScaleImageNode	95
vxHalfScaleGaussianNode	95
vxuScaleImage	96
vxuHalfScaleGaussian	96
3.37 Sobel 3x3	97
3.37.1 Detailed Description	97
3.37.2 Function Documentation	97
vxSobel3x3Node	97
vxuSobel3x3	97
3.38 TableLookup	99
3.38.1 Detailed Description	99
3.38.2 Function Documentation	99
vxTableLookupNode	99
vxuTableLookup	99
3.39 Thresholding	100
3.39.1 Detailed Description	100
3.39.2 Function Documentation	100
vxThresholdNode	100
vxuThreshold	100
3.40 Warp Affine	102
3.40.1 Detailed Description	102
3.40.2 Function Documentation	102
vxWarpAffineNode	102
vxuWarpAffine	103
3.41 Warp Perspective	104
3.41.1 Detailed Description	104
3.41.2 Function Documentation	104
vxWarpPerspectiveNode	104
vxuWarpPerspective	105
3.42 Basic Features	106
3.42.1 Detailed Description	106
3.42.2 Data Structure Documentation	111
struct vx_coordinates2d_t	111
struct vx_coordinates3d_t	111
struct vx_delta_rectangle_t	112
struct vx_keypoint_t	112
struct vx_rectangle_t	112
3.42.3 Macro Definition Documentation	112

VX_VERSION_MAJOR	112
VX_VERSION_MINOR	112
VX_VERSION	113
VX_TYPE_MASK	113
VX_DF_IMAGE	113
VX_ENUM_BASE	113
VX_FMT_REF	113
VX_FMT_SIZE	113
VX_SCALE_UNITY	113
3.42.4 Typedef Documentation	113
vx_enum	113
vx_status	113
3.42.5 Enumeration Type Documentation	114
vx_bool	114
vx_type_e	114
vx_status_e	115
vx_enum_e	116
vx_convert_policy_e	117
vx_df_image_e	117
vx_channel_e	118
vx_interpolation_type_e	118
vx_vendor_id_e	119
3.42.6 Function Documentation	120
vxGetStatus	120
3.43 Objects	121
3.43.1 Detailed Description	121
3.44 Object: Reference	122
3.44.1 Detailed Description	122
3.44.2 Typedef Documentation	122
vx_reference	122
3.44.3 Enumeration Type Documentation	122
vx_reference_attribute_e	122
3.44.4 Function Documentation	122
vxQueryReference	122
3.45 Object: Context	124
3.45.1 Detailed Description	124
3.45.2 Typedef Documentation	125
vx_context	125
3.45.3 Enumeration Type Documentation	125
vx_context_attribute_e	125
vx_import_type_e	126
vx_termination_criteria_e	126
vx_accessor_e	127
vx_round_policy_e	127
3.45.4 Function Documentation	127
vxCreateContext	127
vxReleaseContext	128
vxGetContext	129
vxQueryContext	129
vxSetContextAttribute	130
3.46 Object: Graph	131
3.46.1 Detailed Description	131
3.46.2 Typedef Documentation	132
vx_graph	132
3.46.3 Enumeration Type Documentation	132
vx_graph_attribute_e	132
3.46.4 Function Documentation	132
vxCreateGraph	132

vxReleaseGraph	132
vxVerifyGraph	133
vxProcessGraph	133
vxScheduleGraph	134
vxWaitGraph	134
vxQueryGraph	135
vxSetGraphAttribute	135
vxIsGraphVerified	135
3.47 Object: Node	137
3.47.1 Detailed Description	137
3.47.2 Typedef Documentation	137
vx_node	137
3.47.3 Enumeration Type Documentation	138
vx_node_attribute_e	138
3.47.4 Function Documentation	138
vxQueryNode	138
vxSetNodeAttribute	138
vxReleaseNode	139
vxRemoveNode	139
3.48 Object: Array	140
3.48.1 Detailed Description	140
3.48.2 Macro Definition Documentation	141
vxFormatArrayPointer	141
vxArrayItem	141
3.48.3 Enumeration Type Documentation	141
vx_array_attribute_e	141
3.48.4 Function Documentation	141
vxCreateArray	141
vxCreateVirtualArray	142
vxReleaseArray	142
vxQueryArray	143
vxAddArrayItems	143
vxTruncateArray	144
vxAccessArrayRange	144
vxCommitArrayRange	145
3.49 Object: Convolution	147
3.49.1 Detailed Description	147
3.49.2 Enumeration Type Documentation	147
vx_convolution_attribute_e	147
3.49.3 Function Documentation	148
vxCreateConvolution	148
vxReleaseConvolution	148
vxQueryConvolution	148
vxSetConvolutionAttribute	149
vxReadConvolutionCoefficients	149
vxWriteConvolutionCoefficients	149
3.50 Object: Distribution	150
3.50.1 Detailed Description	150
3.50.2 Enumeration Type Documentation	150
vx_distribution_attribute_e	150
3.50.3 Function Documentation	151
vxCreateDistribution	151
vxReleaseDistribution	151
vxQueryDistribution	151
vxAccessDistribution	152
vxCommitDistribution	152
3.51 Object: Image	153
3.51.1 Detailed Description	153

3.51.2	Data Structure Documentation	154
	struct vx_imagepatch_addressing_t	154
3.51.3	Typedef Documentation	156
	vx_image	156
3.51.4	Enumeration Type Documentation	156
	vx_image_attribute_e	156
	vx_color_space_e	157
	vx_channel_range_e	157
3.51.5	Function Documentation	157
	vxCreateImage	157
	vxCreateImageFromROI	157
	vxCreateUniformImage	158
	vxCreateVirtualImage	158
	vxCreateImageFromHandle	159
	vxQueryImage	159
	vxSetImageAttribute	160
	vxReleaseImage	160
	vxComputeImagePatchSize	161
	vxAccessImagePatch	161
	vxCommitImagePatch	164
	vxFormatImagePatchAddress1d	166
	vxFormatImagePatchAddress2d	168
	vxGetValidRegionImage	169
3.52	Object: LUT	171
3.52.1	Detailed Description	171
3.52.2	Enumeration Type Documentation	171
	vx_lut_attribute_e	171
3.52.3	Function Documentation	171
	vxCreateLUT	171
	vxReleaseLUT	172
	vxQueryLUT	172
	vxAccessLUT	172
	vxCommitLUT	174
3.53	Object: Matrix	175
3.53.1	Detailed Description	175
3.53.2	Enumeration Type Documentation	175
	vx_matrix_attribute_e	175
3.53.3	Function Documentation	175
	vxCreateMatrix	175
	vxReleaseMatrix	176
	vxQueryMatrix	176
	vxReadMatrix	176
	vxWriteMatrix	177
3.54	Object: Pyramid	178
3.54.1	Detailed Description	178
3.54.2	Enumeration Type Documentation	179
	vx_pyramid_attribute_e	179
3.54.3	Function Documentation	179
	vxCreatePyramid	179
	vxCreateVirtualPyramid	179
	vxReleasePyramid	180
	vxQueryPyramid	180
	vxGetPyramidLevel	181
3.55	Object: Remap	182
3.55.1	Detailed Description	182
3.55.2	Enumeration Type Documentation	182
	vx_remap_attribute_e	182
3.55.3	Function Documentation	183

	vxCreateRemap	183
	vxReleaseRemap	184
	vxSetRemapPoint	184
	vxGetRemapPoint	184
	vxQueryRemap	185
3.56	Object: Scalar	186
3.56.1	Detailed Description	186
3.56.2	Typedef Documentation	186
	vx_scalar	186
3.56.3	Enumeration Type Documentation	186
	vx_scalar_attribute_e	186
3.56.4	Function Documentation	186
	vxCreateScalar	186
	vxReleaseScalar	187
	vxQueryScalar	187
	vxReadScalarValue	187
	vxWriteScalarValue	188
3.57	Object: Threshold	189
3.57.1	Detailed Description	189
3.57.2	Enumeration Type Documentation	189
	vx_threshold_type_e	189
	vx_threshold_attribute_e	190
3.57.3	Function Documentation	190
	vxCreateThreshold	190
	vxReleaseThreshold	190
	vxSetThresholdAttribute	191
	vxQueryThreshold	191
3.58	Administrative Features	192
3.58.1	Detailed Description	192
3.59	Advanced Objects	193
3.59.1	Detailed Description	193
3.60	Object: Array (Advanced)	194
3.60.1	Detailed Description	194
3.60.2	Function Documentation	194
	vxRegisterUserStruct	194
3.61	Object: Node (Advanced)	195
3.61.1	Detailed Description	195
3.61.2	Function Documentation	195
	vxCreateGenericNode	195
3.62	Node: Border Modes	196
3.62.1	Detailed Description	196
3.62.2	Data Structure Documentation	196
	struct vx_border_mode_t	196
3.62.3	Enumeration Type Documentation	196
	vx_border_mode_e	196
3.63	Object: Delay	197
3.63.1	Detailed Description	197
3.63.2	Typedef Documentation	197
	vx_delay	197
3.63.3	Enumeration Type Documentation	197
	vx_delay_attribute_e	197
3.63.4	Function Documentation	198
	vxQueryDelay	198
	vxReleaseDelay	199
	vxCreateDelay	199
	vxGetReferenceFromDelay	199
	vxAgeDelay	200
3.64	Object: Kernel	201

3.64.1	Detailed Description	201
3.64.2	Data Structure Documentation	203
	struct vx_kernel_info_t	203
3.64.3	Typedef Documentation	203
	vx_kernel	203
3.64.4	Enumeration Type Documentation	203
	vx_kernel_e	203
	vx_kernel_attribute_e	207
3.64.5	Function Documentation	207
	vxGetKernelByName	207
	vxGetKernelByEnum	208
	vxQueryKernel	209
	vxReleaseKernel	209
3.65	Object: Parameter	211
3.65.1	Detailed Description	211
3.65.2	Typedef Documentation	212
	vx_parameter	212
3.65.3	Enumeration Type Documentation	212
	vx_direction_e	212
	vx_parameter_attribute_e	212
	vx_parameter_state_e	212
3.65.4	Function Documentation	213
	vxGetKernelParameterByIndex	213
	vxGetParameterByIndex	214
	vxReleaseParameter	214
	vxSetParameterByIndex	214
	vxSetParameterByReference	215
	vxQueryParameter	216
3.66	Advanced Framework API	217
3.66.1	Detailed Description	217
3.67	Framework: Node Callbacks	218
3.67.1	Detailed Description	218
3.67.2	Typedef Documentation	220
	vx_action	220
	vx_nodecomplete_f	220
3.67.3	Enumeration Type Documentation	220
	vx_action_e	220
3.67.4	Function Documentation	220
	vxAssignNodeCallback	220
	vxRetrieveNodeCallback	221
3.68	Framework: Performance Measurement	222
3.68.1	Detailed Description	222
3.68.2	Data Structure Documentation	222
	struct vx_perf_t	222
3.69	Framework: Log	223
3.69.1	Detailed Description	223
3.69.2	Function Documentation	223
	vxAddLogEntry	223
	vxRegisterLogCallback	223
3.70	Framework: Hints	225
3.70.1	Detailed Description	225
3.70.2	Enumeration Type Documentation	225
	vx_hint_e	225
3.70.3	Function Documentation	225
	vxHint	225
3.71	Framework: Directives	227
3.71.1	Detailed Description	227
3.71.2	Enumeration Type Documentation	227

vx_directive_e	227
3.71.3 Function Documentation	227
vxDirective	227
3.72 Framework: User Kernels	229
3.72.1 Detailed Description	229
3.72.2 Typedef Documentation	231
vx_meta_format	231
vx_publish_kernels_f	231
vx_kernel_f	231
vx_kernel_initialize_f	231
vx_kernel_deinitialize_f	232
vx_kernel_input_validate_f	232
vx_kernel_output_validate_f	232
3.72.3 Enumeration Type Documentation	233
vx_meta_format_attribute_e	233
3.72.4 Function Documentation	233
vxLoadKernels	233
vxAddKernel	234
vxFinalizeKernel	234
vxAddParameterToKernel	234
vxRemoveKernel	235
vxSetKernelAttribute	235
vxSetMetaFormatAttribute	236
3.73 Framework: Graph Parameters	238
3.73.1 Detailed Description	238
3.73.2 Function Documentation	239
vxAddParameterToGraph	239
vxSetGraphParameterByIndex	239
vxGetGraphParameterByIndex	240

Chapter 1

Introduction

1.1 Abstract

OpenVX is a low-level programming framework domain to enable software developers to efficiently access computer vision hardware acceleration with both functional and performance portability. OpenVX has been designed to support modern hardware architectures, such as mobile and embedded SoCs as well as desktop systems. Many of these systems are parallel and heterogeneous: containing multiple processor types including multi-core CPUs, DSP subsystems, GPUs, dedicated vision computing fabrics as well as hardwired functionality. Additionally, vision system memory hierarchies can often be complex, distributed, and not fully coherent. OpenVX is designed to maximize functional and performance portability across these diverse hardware platforms, providing a computer vision framework that efficiently addresses current and future hardware architectures with minimal impact on applications.

OpenVX contains:

- a library of predefined and customizable vision functions,
- a graph-based execution model to combine function enabling both task and data-independent execution, and;
- a set of memory objects that abstract the physical memory.

OpenVX defines a C Application Programming Interface (API) for building, verifying, and coordinating graph execution, as well as for accessing memory objects. The graph abstraction enables OpenVX implementers to optimize the execution of the graph for the underlying acceleration architecture.

OpenVX also defines the `vxu` utility library, which exposes each OpenVX predefined function as a directly callable C function, without the need for first creating a graph. Applications built using the `vxu` library do not benefit from the optimizations enabled by graphs; however, the `vxu` library can be useful as the simplest way to use OpenVX and as first step in porting existing vision applications.

As the computer vision domain is still rapidly evolving, OpenVX provides an extensibility mechanism to enable developer-defined functions to be added to the application graph.

1.2 Purpose

The purpose of this document is to detail the Application Programming Interface (API) for OpenVX.

1.3 Scope of Specification

The document contains the definition of the OpenVX API. The conformance tests that are used to determine whether an implementation is consistent to this specification are defined separately.

1.4 Normative References

The section “Module Documentation” forms the normative part of the specification. Each API definition provided in that chapter has certain preconditions and post conditions specified that are normative. If these normative conditions are not met, the behavior of the function is undefined.

1.5 Version/Change History

- OpenVX 1.0 Provisional - November, 2013
- OpenVX 1.0 Provisional V2 - June, 2014
- OpenVX 1.0 - September 2014
- OpenVX 1.0.1 - April 2015

1.6 Requirements Language

In this specification, the words *shall* or *must* express a requirement that is binding, *should* expresses design goals or recommended actions, and *may* expresses an allowed behavior.

1.7 Typographical Conventions

The following typographical conventions are used in this specification.

- **Bold** words indicate warnings or strongly communicated concepts that are intended to draw attention to the text.
- *Monospace* words signify an API element (i.e., class, function, structure) or a filename.
- *Italics* denote an emphasis on a particular concept, an abstraction of a concept, or signify an argument, parameter, or member.
- Throughout this specification, code examples given to highlight a particular issue use the format as shown below:
- ```
/* Example Code Section */
int main(int argc, char *argv[])
{
 return 0;
}
```
- Some “mscgen” message diagrams are included in this specification. The graphical conventions for this tool can be found on its website.

See also

<http://www.mcternan.me.uk/mscgen/>

### 1.7.1 Naming Conventions

The following naming conventions are used in this specification.

- Opaque objects and atomics are named as `vx_object`, e.g., `vx_image` or `vx_uint8`, with an underscore separating the object name from the “vx” prefix.
- Defined Structures are named as `vx_struct_t`, e.g., `vx_imagepatch_addressing_t`, with underscores separating the structure from the “vx” prefix and a “t” to denote that it is a structure.
- Defined Enumerations are named as `vx_enum_e`, e.g., `vx_type_e`, with underscores separating the enumeration from the “vx” prefix and an “e” to denote that it is an enumerated value.
- Application Programming Interfaces are named `vxSomeFunction()` using camel case, starting with lower-case, and no underscores, e.g., `vxCreateContext()`.
- Vision functions also have a naming convention that follows a lower-case, inverse dotted hierarchy similar to Java Packages, e.g.,

```
"org.khronos.openvx.color_convert".
```

This minimizes the possibility of name collisions and promotes sorting and readability when querying the namespace of available vision functions. Each vision function should have a unique dotted name of the style: *tld.vendor.library.function*. The hierarchy of such vision function namespaces is undefined outside the subdomain “org.khronos”, but they do follow existing international standards. For OpenVX-specified vision functions, the “function” section of the unique name does not use camel case and uses underscores to separate words.

## 1.8 Glossary and Acronyms

- Atomic: The specification mentions *atomics*, which means a C primitive data type. Usages that have additional wording, such as *atomic operations* do not carry this meaning.
- API: Application Programming Interface that specifies how a software component interacts with another.
- Framework: A generic software abstraction in which users can override behaviors to produce application-specific functionality.
- Engine: A purpose-specific software abstraction that is tunable by users.
- Run-time: The execution phase of a program.
- Kernel: OpenVX uses the term *kernel* to mean an abstract *computer vision function*, not an Operating System kernel. Kernel may also refer to a set of convolution coefficients in some computer vision literature (e.g., the Sobel “kernel”). OpenVX does not use this meaning. OpenCL uses kernel (specifically `cl_kernel`) to qualify a function written in “CL” which the OpenCL may invoke directly. This is close to the meaning OpenVX uses; however, OpenVX does not define a language.

## 1.9 Acknowledgements

This specification would not be possible without the contributions from this partial list of the following individuals from the Khronos Working Group and the companies that they represented at the time:

- Erik Rainey - Amazon
- Susheel Gautam - QUALCOMM
- Victor Erukhimov - Itseez
- Doug Knisely - QUALCOMM
- Frank Brill - Samsung
- Kari Pulli - NVIDIA
- Thierry Lepley - NVIDIA
- Neil Trevett - NVIDIA
- Tomer Schwartz - Broadcom Corporation
- Shervin Emami - NVIDIA
- Olivier Pothier - STMicroelectronics International NV
- Andy Kuzma - Intel
- Mostafa Hagog - Intel
- Shorin Kyo - Huawei
- Renato Grottesi - ARM Limited
- Dave Schreiner - ARM Limited
- Chris Tseng - Texas Instruments, Inc.



- Daniel Laroche - CogniVue Corporation
- Andrew Garrard - Samsung Electronics
- Tomer Yanir - Samsung Electronics
- Erez Natan - Samsung Electronics
- Chang-Hyo Yu - Samsung Electronics
- Hans-Peter Nilsson - Axis Communications
- Stephen Neuendorffer - Xilinx, Inc.
- Amit Shoham - BDTi
- Paul Buxton - Imagination Technologies
- Yuki Kobayashi - Renesas Electronics
- Cormac Brick - Movidius Ltd
- Mikael Bourges-Sevenier - Aptina Imaging Corporation
- Tao Zhang - QUALCOMM
- Jesse Villareal - Texas Instruments, Inc.
- Vadim Pisarevsky - Itseez
- Andrey Kamaev - Itseez
- Vlad Vinogradov - Itseez
- Roman Donchenko - Itseez
- Alexander Alekhin - Itseez
- Radha Giduthuri - AMD
- Xin Wang - Vivante Corporation
- Anshu Arya - MulticoreWare

## Chapter 2

# Design Overview

### 2.1 Software Landscape

OpenVX is intended to be used either directly by applications or as the acceleration layer for higher-level vision frameworks, engines or platform APIs.

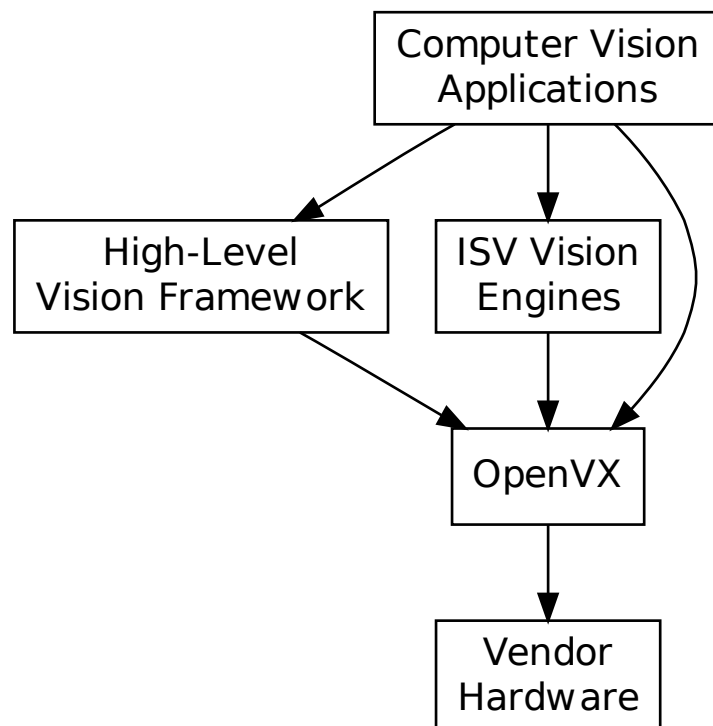


Figure 2.1: OpenVX Usage Overview

### 2.2 Design Objectives

OpenVX is designed as a framework of standardized computer vision functions able to run on a wide variety of platforms and potentially to be accelerated by a vendor's implementation on that platform. OpenVX can improve the

performance and efficiency of vision applications by providing an abstraction for commonly-used vision functions and an abstraction for aggregations of functions (a “graph”), thereby providing the implementer the opportunity to minimize the run-time overhead.

The functions in OpenVX 1.0 are intended to cover common functionality required by many vision applications.

### 2.2.1 Hardware Optimizations

This specification makes no statements as to which acceleration methodology or techniques may be used in its implementation. Vendors may choose any number of implementation methods such as parallelism and/or specialized hardware offload techniques.

This specification also makes no statement or requirements on a “level of performance” as this may vary significantly across platforms and use cases.

### 2.2.2 Hardware Limitations

The OpenVX 1.0 focuses on vision functions that can be significantly accelerated by diverse hardware. Future versions of this specification may adopt additional vision functions into the core standard when hardware acceleration for those functions becomes practical.

## 2.3 Assumptions

### 2.3.1 Portability

OpenVX 1.0 has been designed to maximize functional and performance portability wherever possible, while recognizing that the API is intended to be used on a wide diversity of devices with specific constraints and properties. Tradeoffs are made for portability where possible: for example, portable Graphs constructed using this API should work on any OpenVX implementation and return similar results within the precision bounds defined by the OpenVX conformance tests.

### 2.3.2 Opaqueness

To avoid forcing hardware-specific requirements onto any particular implementation, the API is designed to be *opaque*.

OpenVX is intended to address a very broad range of devices and platforms - from deeply embedded systems to desktop machines, and even distributed computing architectures.

The range of implementations is quite discreet, and as such, the API shall only address all these spaces through opaqueness.

All data, except client-facing structures, are opaque and hidden behind a reference that may be as thin or thick as an implementation needs. Each implementation provides the standardized interfaces for accessing data that takes care of specialized hardware, platform, or allocation requirements. Memory that is *imported* or *shared* from other APIs is not subsumed by OpenVX and is still maintained and accessible by the originator.

OpenVX does not dictate any requirements on memory allocation methods or the layout of opaque memory objects and it does not dictate byte packing or alignment for structures on architectures.

## 2.4 Object-Oriented Behaviors

OpenVX objects are both strongly typed at compile-time for safety critical applications and are strongly typed at run-time for dynamic applications. Each object has its typedef'd type and its associated enumerated value in the `vx_type_e` list. Any object may be down-cast to a `vx_reference` safely to be used in functions that require this, specifically `vxQueryReference`, which can be used to get the `vx_type_e` value using an `vx_enum`.

## 2.5 OpenVX Framework Objects

This specification defines the following OpenVX framework objects.

- **Object: Context** - The OpenVX context is the object domain for all OpenVX objects. All data objects *live* in the context as well as all framework objects. The OpenVX context keeps reference counts on all objects and must do garbage collection during its deconstruction to free lost references. While multiple clients may connect to the OpenVX context, all data are private in that the references that refer to data objects are given only to the creating party. The results of calling an OpenVX function on data objects created in different contexts are undefined.
- **Object: Kernel** - A Kernel in OpenVX is the abstract representation of a computer vision function, such as a “Sobel Gradient” or “Lucas Kanade Feature Tracking”. A vision function may implement many similar or identical features from other functions, but it is still considered a single, unique kernel as long as it is named by the same string and enumeration and conforms to the results specified by OpenVX. Kernels are similar to function signatures in this regard.
- **Object: Parameter** - An abstract input, output, or bidirectional data object passed to a computer vision function. This object contains the signature of that parameter’s usage from the kernel description. This information includes:
  - *Signature Index* - The numbered index of the parameter in the signature.
  - *Object Type* - e.g. `VX_TYPE_IMAGE`, or `VX_TYPE_ARRAY`, or some other object type from `vx_type_e`.
  - *Usage Model* - e.g. `VX_INPUT`, `VX_OUTPUT`, or `VX_BIDIRECTIONAL`.
  - *Presence State* - e.g. `VX_PARAMETER_STATE_REQUIRED`, or `VX_PARAMETER_STATE_OPTIONAL`.
- **Object: Node** - A node is an instance of a kernel that will be paired with a specific set of references (the parameters). Nodes are created from and associated with a single graph only. When a `vx_parameter` is extracted from a Node, an additional attribute can be accessed:
  - *Reference* - The `vx_reference` assigned to this parameter index from the Node creation function (e.g., `vxSobel3x3Node`).
- **Object: Graph** - A set of nodes connected in a directed (only goes one-way) acyclic (does not loop back) fashion. A Graph may have sets of Nodes that are unconnected to other sets of Nodes within the same Graph. See [Graph Formalisms](#).

## 2.6 OpenVX Data Objects

Data objects are object that are processed by graphs in nodes.

- **Object: Array** An opaque array object that could be an array of primitive data types or an array of structures.
- **Object: Convolution** An opaque object that contains  $M \times N$  matrix of `vx_int16` values. Also contains a scaling factor for normalization. Used specifically with `vxuConvolve` and `vxConvolveNode`.
- **Object: Delay** An opaque object that contains a manually controlled, temporally-delayed list of objects.
- **Object: Distribution** An opaque object that contains a frequency distribution (e.g., a histogram).
- **Object: Image** An opaque image object that may be some format in `vx_df_image_e`.
- **Object: LUT** An opaque lookup table object used with `vxTableLookupNode` and `vxuTableLookup`.
- **Object: Matrix** An opaque object that contains  $M \times N$  matrix of some scalar values.
- **Object: Pyramid** An opaque object that contains multiple levels of scaled `vx_image` objects.
- **Object: Remap** An opaque object that contains the map of source points to destination points used to transform images.
- **Object: Scalar** An opaque object that contains a single primitive data type.
- **Object: Threshold** An opaque object that contains the thresholding configuration.

## 2.7 Error Objects

Error objects are specialized objects that may be returned from other object creator functions when serious platform issue occur (i.e., out of memory or out of handles). These can be checked at the time of creation of these objects, but checking also may be put-off until usage in other APIs or verification time, in which case, the implementation must return appropriate errors to indicate that an invalid object type was used.

```
vx_<object> obj = vxCreate<Object>(context, ...);
vx_status status = vxGetStatus((vx_reference)obj);
if (status == VX_SUCCESS) {
 // object is good
}
```

## 2.8 Graphs Concepts

The *graph* is the central computation concept of OpenVX. The purpose of using graphs to express the Computer Vision problem is to allow for the possibility of any implementation to maximize its optimization potential because all the operations of the graph and its dependencies are known ahead of time, before the graph is processed.

Graphs are composed of one or more *nodes* that are added to the graph through node creation functions. Graphs in OpenVX must be created ahead of processing time and verified by the implementation, after which they can be processed as many times as needed.

### 2.8.1 Linking Nodes

Graph Nodes are linked together via data dependencies with *no explicitly-stated ordering*. The same reference may be linked to other nodes. Linking has a limitation, however, in that only one node in a graph may output to any specific data object reference. That is, only a single writer of an object may exist in a given graph. This prevents indeterminate ordering from data dependencies. All writers in a graph shall produce output data before any reader of that data accesses it.

### 2.8.2 Virtual Data Objects

Graphs in OpenVX depend on data objects to link together nodes. When clients of OpenVX know that they do not need access to these *intermediate* data objects, they may be created as *virtual*. Virtual data objects can be used in the same manner as non-virtual data objects to link nodes of a graph together; however, virtual data objects are different in the following respects.

- Inaccessible - No calls to an Access/Commit API shall succeed given a reference to an object created through a *virtual* create function from a Graph *external* perspective. Calls to Access/Commit from within client-defined functions may succeed as they are Graph *internal*.
- Dimensionless or Formatless - Virtual data objects may be declared to have no dimensions or format and they may return zeros or generic values for formats when queried.
- Scoped - Virtual data objects are scoped within the Graph in which they are created; they cannot be shared outside their scope.
- Intermediates - Virtual data objects should be used only for intermediate operations within Graphs, because they are fundamentally inaccessible to clients of the API.
- Optimizations - Virtual data objects do not have to be created during Graph validation and execution and therefore may be of zero *size*.

These restrictions enable vendors the ability to optimize some aspects of the data object or its usage. Some vendors may not allocate such objects, some may create intermediate sub-objects of the object, and some may allocate the object on remote, inaccessible memories. OpenVX does not proscribe *which* optimization the vendor does, merely that it *may* happen.

### 2.8.3 Node Parameters

Parameters to node creation functions are defined as either atomic types, such as `vx_int32`, `vx_enum`, or as objects, such as `vx_scalar`, `vx_image`. The atomic variables of the Node creation functions shall be converted by the framework into `vx_scalar` references for use by the Nodes. A node parameter of type `vx_scalar` can be changed during the graph execution; whereas, a node parameter of an atomic type (`vx_int32` etc.) require at least a graph revalidation if changed. All node parameter objects may be modified by retrieving the reference to the `vx_parameter` via `vxGetParameterByIndex`, and then passing that to `vxQueryParameter` to retrieve the reference to the object.

```
vx_parameter param = vxGetParameterByIndex(node, p);
vx_reference ref;
vxQueryParameter(param, VX_PARAMETER_ATTRIBUTE_REF, &ref,
sizeof(ref));
```

If the type of the parameter is unknown, it may be retrieved with the same function.

```
vx_enum type;
vxQueryParameter(param, VX_PARAMETER_ATTRIBUTE_TYPE,
&type, sizeof(type));
/* cast the ref to the correct vx_<type>. Atomics are now vx_scalar */
```

### 2.8.4 Graph Parameters

Parameters may exist on Graphs, as well. These parameters are defined by the author of the Graph and each Graph parameter is defined as a specific parameter from a Node within the Graph using `vxAddParameterToGraph`. Graph parameters communicate to the implementation that there are specific Node parameters that may be modified by the client between Graph executions. Additionally, they are parameters that the client may set without the reference to the Node but with the reference to the Graph using `vxSetGraphParameterByIndex`. This allows for the Graph authors to construct *Graph Factories*. How these factories work falls outside the scope of this document.

See also

[Framework: Graph Parameters](#)

### 2.8.5 Execution Model

Graphs must execute in both:

- *Synchronous blocking mode* (in that `vxProcessGraph` will block until the graph has completed), and in
- *Asynchronous single-issue-per-reference mode* (via `vxScheduleGraph` and `vxWaitGraph`).

#### Asynchronous Mode

In asynchronous mode, Graphs must be single-issue-per-reference. This means that given a constructed graph reference  $G$ , it may be scheduled multiple times but only executes sequentially with respect to itself. Multiple graphs references given to the asynchronous graph interface do not have a defined behavior and may execute in parallel or in series based on the behavior or the vendor's implementation.

### 2.8.6 Graph Formalisms

To use graphs several rules must be put in place to allow deterministic execution of Graphs. The behavior of a `processGraph( G )` call is determined by the structure of the Processing Graph  $G$ . The Processing Graph is a bipartite graph consisting of a set of Nodes  $N_1 \dots N_n$  and a set of data objects  $d_1 \dots d_i$ . Each edge  $( N_x, D_y )$  in the graph represents a data object  $D_y$  that is written by Node  $N_x$  and each edge  $( D_x, N_y )$  represents a data object  $D_x$  that is read by Node  $N_y$ . Each edge  $e$  has a name  $Name( e )$ , which gives the parameter name of the node that references the corresponding data object. Each Node Parameter also as a type  $Type( node, name )$  in  $\{ INPUT, OUTPUT, INOUT \}$ . Some data objects are *Virtual*, and some data objects are *Delay*. Delay data objects are just collections of data objects with indexing (like an image list) and known linking points in a graph. A node may be classified as a *head node*, which has no backward dependency. Alternatively, a node may be a *dependent node*, which has a backward dependency to the head node. In addition, the Processing Graph has several restrictions:

1. *Output typing* - Every output edge  $(N_x, D_y)$  requires  $\text{Type}(N_x, \text{Name}(N_x, D_y))$  in  $\{\text{OUTPUT}, \text{IN} \leftrightarrow \text{OUT}\}$
2. *Input typing* - Every input edge  $(N_x, D_y)$  requires  $\text{Type}(N_y, \text{Name}(D_x, N_y))$  in  $\{\text{INPUT}\}$  or  $\{\text{IN} \leftrightarrow \text{OUT}\}$
3. *Single Writer* - Every data object is the target of at most one output edge.
4. *Broken Cycles* - Every cycle in  $G$  must contain at least input edge  $(D_x, N_y)$  where  $D_x$  is Delay.
5. *Virtual images must have a source* - If  $D_y$  is Virtual, then there is at least one output edge that writes  $D_y(N_x, D_y)$
6. *Bidirectional data objects shall not be virtual* - If  $\text{Type}(N_x, \text{Name}(N_x, D_y))$  is INOUT implies  $D_y$  is non-Virtual.
7. *Delay data objects shall not be virtual* - If  $D_x$  is Delay then it shall not be Virtual.
8. *A uniform image cannot be output or bidirectional.*

The execution of each node in a graph consists of an atomic operation (sometimes referred to as firing) that consumes data representing each input data object, processes it, and produces data representing each output data object. A node may execute when all of its input edges are marked present. Before the graph executes, the following initial marking is used:

- All input edges  $(D_x, N_y)$  from non-Virtual objects  $D_x$  are marked (parameters must be set).
- All input edges  $(D_x, N_y)$  with an output edge  $(N_z, D_x)$  are unmarked.
- All input edges  $(D_x, N_y)$  where  $D_x$  is a Delay data object are marked.

Processing a node results in unmarking all the corresponding input edges and marking all its output edges; marking an output edge  $(N_x, D_y)$  where  $D_y$  is not a Delay results in marking all of the input edges  $(D_y, N_z)$ . Following these rules, it is possible to statically schedule the nodes in a graph as follows: Construct a precedence graph  $P$ , including all the nodes  $N_1 \dots N_x$ , and an edge  $(N_x, N_z)$  for every pair of edges  $(N_x, D_y)$  and  $(D_y, N_z)$  where  $D_y$  is not a Delay. Then unconditionally fire each node according to any topological sort of  $P$ .

The following assertions should be verified:

- $P$  is a Directed Acyclic Graph (DAG), implied by 4 and the way it is constructed.
- Every data object has a value when it is executed, implied by 5, 6, 7, and the marking.
- Execution is deterministic if the nodes are deterministic, implied by 3, 4, and the marking.
- Every node completes its execution exactly once.

The execution model described here just acts as a formalism. For example, independent processing is allowed across multiple depended and depending nodes and edges, provided that the result is invariant with the execution model described here.

## 2.8.7 Node Execution Independence

In the following example a client computes the gradient magnitude and gradient phase from a blurred input image. The `vxMagnitudeNode` and `vxPhaseNode` are independently computed, in that each does not depend on the output of the other. OpenVX does not mandate that they are run simultaneously or in parallel, but it could be implemented this way by the OpenVX vendor.

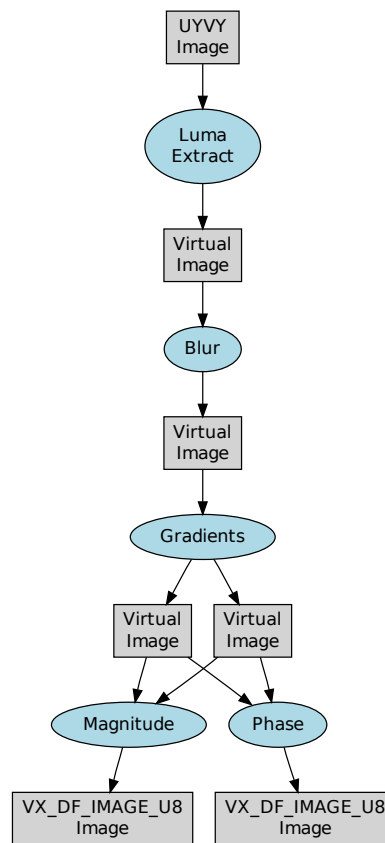


Figure 2.2: A simple graph with some independent nodes.

The code to construct such a graph can be seen below.

```

vx_context context = vxCreateContext();
vx_image images[] = {
 vxCreateImage(context, 640, 480, VX_DF_IMAGE_UYVY),
 vxCreateImage(context, 640, 480, VX_DF_IMAGE_U8),
 vxCreateImage(context, 640, 480, VX_DF_IMAGE_U8),
};

vx_graph graph = vxCreateGraph(context);
vx_image virts[] = {
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
};

vxChannelExtractNode(graph, images[0], VX_CHANNEL_Y, virts[0]),
vxGaussian3x3Node(graph, virts[0], virts[1]),
vxSobel3x3Node(graph, virts[1], virts[2], virts[3]),
vxMagnitudeNode(graph, virts[2], virts[3], images[1]),
vxPhaseNode(graph, virts[2], virts[3], images[2]),

status = vxVerifyGraph(graph);
if (status == VX_SUCCESS)
{
 status = vxProcessGraph(graph);
}
vxReleaseContext(&context); /* this will release everything */

```



### 2.8.8 Verification

Graphs within OpenVX must go through a rigorous validation process before execution to satisfy the design concept of eliminating run-time overhead (parameter checking) that guarantees safe execution of the graph. OpenVX must check for (but is not limited to) these conditions:

- Parameters To Nodes:
  - Each required parameter is given to the node (`vx_parameter_state_e`). Optional parameters may not be present and therefore are not checked when absent. If present, they are checked.
  - Each parameter given to a node must be of the right direction (a value from `vx_direction_e`).
  - Each parameter given to a node must be of the right object type (from the object range of `vx_type_e`).
  - Each parameter attribute or value must be verified. In the case of a scalar value, it may need to be range checked (e.g.,  $0.5 \leq k \leq 1.0$ ). The implementation is not required to do run-time range checking of scalar values. If the value of the scalar changes at run time to go outside the range, the results are undefined. The rationale is that the potential performance hit for run-time range checking is too large to be enforced. It will still be checked at graph verification time as a time-zero sanity check. If the scalar is an output parameter of another node, it must be initialized to a legal value. In the case of `vxScaleImageNode`, the relation of the input image dimensions to the output image dimensions determines the scaling factor. These values or attributes of data objects must be checked for compatibility on each platform.
  - Graph Connectivity - the `vx_graph` must be a Directed Acyclic Graph (DAG). No cycles or feedback is allowed. The `vx_delay` object has been designed to explicitly address feedback between Graph executions.
  - Resolution of Virtual Data Objects - Any changes to Virtual data objects from unspecified to specific format or dimensions, as well as the related creation of objects of specific type that are observable at processing time, takes place at Verification time.

## 2.9 Callbacks

Callbacks are a method to control graph flow and to make decisions based on completed work. The `vxAssignNodeCallback` call takes as a parameter a callback function. This function will be called after the execution of the particular node, but prior to the completion of the graph. If nodes are arranged into independent sets, the order of the callbacks is unspecified. Nodes that are arranged in a serial fashion due to data dependencies perform callbacks in order. The callback function may use the node reference first to extract parameters from the node, and then extract the data references. Data outputs of Nodes with callbacks shall be available (via Access/Commit methods) when the callback is called.

## 2.10 User Kernels

OpenVX supports the concept of client-defined functions that shall be executed as Nodes from inside the Graph or are Graph internal. The purpose of this paradigm is to:

- Further exploit independent operation of nodes within the OpenVX platform.
- Allow componentized functions to be reused elsewhere in OpenVX.
- Formalize strict verification requirements (i.e., Contract Programming).

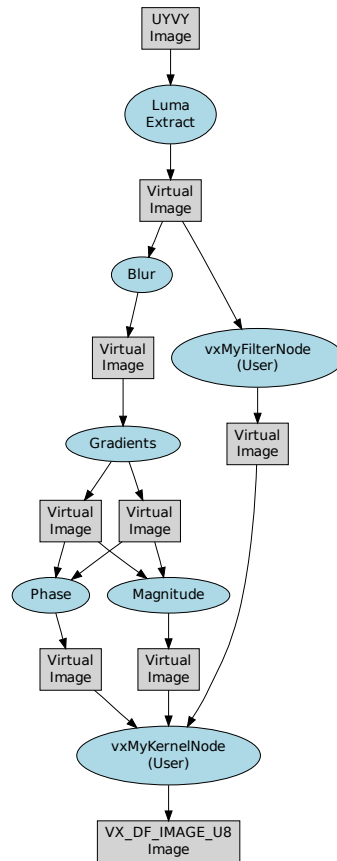


Figure 2.3: A graph with User Kernel nodes which are independent of the “base” nodes.

*In this example, to execute client-supplied functions, the graph does not have to be halted and then resumed. These nodes shall be executed in an independent fashion with respect to independent base nodes within OpenVX. This allows implementations to further minimize execution time if hardware to exploit this property exists.*

### 2.10.1 Parameter Validation

*User Kernels must aid in the Graph Verification effort by providing explicit validation functions for each vision function they implement. Each parameter passed to the instantiated Node of a User Kernel is validated using the client-supplied validation functions. The client must check these attributes and/or values of each parameter:*

- *Each attribute or value of the parameter must be checked. For example, the size of array, or the value of a scalar to be within a range, or a dimensionality constraint of an image such as width divisibility. (Some implementations may have restrictions, such as an image width be evenly divisible by some fixed number).*
- *If the output parameters depend on attributes or values from input parameters, those relationships must be checked (within the output validator).*

*Input validators execute before output validators. This allows any or all inputs to be used as dependents of output parameter validation.*

### The Meta Format Object

*The Meta Format Object is an opaque object used to collect requirements about the output parameter, which then the OpenVX implementation will check. The Client must manually set relevant object attributes to be checked against output parameters, such as dimensionality, format, scaling, etc.*

## Delta Rectangles

There is a special case with `vx_image` output parameters where the User Kernel output validation function can specify a positional and/or size-related change of the valid region of the output image relative to the input image during verification time. This is intended to give the optimizer more information about memory usage, and could lead to better outcomes or different strategies. Delta rectangles (specified using the `vx_delta_rectangle_t` parameter) are used to update a valid region for the user kernels with a call to `vxSetMetaFormatAttribute` from the output validator.

For example, for a 5x5 box filter where 2 border pixels of the output are lost (invalid), and with no center shift, use:

```
vx_delta_rectangle_t delta = {2, 2, -2, -2};
```

For the same 5x5 box filter, except with a center-shift into the upper-left corner:

```
vx_delta_rectangle_t delta = {0, 0, -4, -4};
```

If this attribute has not been set prior to graph verification, the graph manager must determine the new valid region based on `vxCommitImagePatch` calls during the execution time.

```
// User Kernel Function is a 5x5 filter with no explicit border handling code.
// So, 2 pixels on each edge of the image will be lost.
vx_delta_rectangle_t delta = {2, 2, -2, -2};
vxSetMetaFormatAttribute(meta,
 VX_META_FORMAT_ATTRIBUTE_DELTA_RECTANGLE, &delta, sizeof(delta));
```

### 2.10.2 User Kernels Naming Conventions

User Kernels must be exported with a unique name (see [Naming Conventions](#) for information on OpenVX conventions) and a unique enumeration. Clients of OpenVX may use either the name or enumeration to retrieve a kernel, so collisions due to non-unique names will cause problems. The kernel enumerations may be extended by following this example:

```
#define VX_KERNEL_NAME_KHR_XYZ "org.khronos.example.xyz"

#define VX_LIBRARY_XYZ (0x3) // assigned from Khronos, vendors control their own

enum vx_kernel_xyz_ext_e {
 VX_KERNEL_KHR_XYZ = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_XYZ) + 0x0,
 // up to 0xFFFF kernel enums can be created.
};
```

Each vendor of a vision function or an implementation must apply to Khronos to get a unique identifier (up to a limit of  $2^{12} - 1$  vendors). Until they obtain a unique ID vendors must use `VX_ID_DEFAULT`.

To construct a kernel enumeration, a vendor must have both their ID and a library ID. The library ID's are completely vendor defined (however when using the `VX_ID_DEFAULT` ID, many libraries may collide in namespace).

Once both are defined, a kernel enumeration may be constructed using the `VX_KERNEL_BASE` macro and an offset. (The offset is optional, but very helpful for long enumerations.)

## 2.11 Immediate Mode Functions

OpenVX also contains an interface defined within `<VX/vxu.h>` that allows for immediate execution of vision functions. These interfaces are prefixed with `vxu` to distinguish them from the Node interfaces, which are of the form `vx<Name>Node`. Each of these interfaces replicates a Node interface with some exceptions. Immediate mode functions are defined to behave as Single Node Graphs, which have no leaking side-effects (e.g., no Log entries) within the Graph Framework after the function returns. The following tables refer to both the Immediate Mode and Graph Mode vision functions. The Module documentation for each vision function draws a distinction on each API by noting that it is either an immediate mode function with the tag `[Immediate]` or it is a Graph mode function by the tag `[Graph]`.

## 2.12 Base Vision Functions

OpenVX comes with a standard or base set of vision functions. The following table lists the supported set of vision functions, their input types (first table) and output types (second table), and the version of OpenVX in which they are supported.

### **2.12.1 Inputs**

| Vision Function            | U8  | U16 | S16 | S32 | U32 | F32 | color |
|----------------------------|-----|-----|-----|-----|-----|-----|-------|
| <i>AbsDiff</i>             | 1.0 |     | 1.0 |     |     |     |       |
| <i>Accumulate</i>          | 1.0 |     |     |     |     |     |       |
| <i>Accumulate↔Squared</i>  | 1.0 |     |     |     |     |     |       |
| <i>Accumulate↔Weighted</i> | 1.0 |     |     |     |     |     |       |
| <i>Add</i>                 | 1.0 |     | 1.0 |     |     |     |       |
| <i>And</i>                 | 1.0 |     |     |     |     |     |       |
| <i>Box3x3</i>              | 1.0 |     |     |     |     |     |       |
| <i>Canny↔Edge↔Detector</i> | 1.0 |     |     |     |     |     |       |
| <i>Channel↔Combine</i>     | 1.0 |     |     |     |     |     |       |
| <i>Channel↔Extract</i>     |     |     |     |     |     |     | 1.0   |
| <i>Color↔Convert</i>       |     |     |     |     |     |     | 1.0   |
| <i>Convert↔Depth</i>       | 1.0 |     | 1.0 |     |     |     |       |
| <i>Convolve</i>            | 1.0 |     |     |     |     |     |       |
| <i>Dilate3x3</i>           | 1.0 |     |     |     |     |     |       |
| <i>Equalize↔Histogram</i>  | 1.0 |     |     |     |     |     |       |
| <i>Erode3x3</i>            | 1.0 |     |     |     |     |     |       |
| <i>Fast↔Corners</i>        | 1.0 |     |     |     |     |     |       |
| <i>Gaussian3x3</i>         | 1.0 |     |     |     |     |     |       |
| <i>Harris↔Corners</i>      | 1.0 |     |     |     |     |     |       |
| <i>Half↔Scale↔Gaussian</i> | 1.0 |     |     |     |     |     |       |
| <i>Histogram</i>           | 1.0 |     |     |     |     |     |       |
| <i>Integral↔Image</i>      | 1.0 |     |     |     |     |     |       |
| <i>Table↔Lookup</i>        | 1.0 |     |     |     |     |     |       |
| <i>Magnitude</i>           |     |     | 1.0 |     |     |     |       |
| <i>MeanStd↔Dev</i>         | 1.0 |     |     |     |     |     |       |
| <i>Median3x3</i>           | 1.0 |     |     |     |     |     |       |
| <i>MinMax↔Loc</i>          | 1.0 |     | 1.0 |     |     |     |       |
| <i>Multiply</i>            | 1.0 |     | 1.0 |     |     |     |       |
| <i>Not</i>                 | 1.0 |     |     |     |     |     |       |

|                              |     |  |     |  |  |  |  |
|------------------------------|-----|--|-----|--|--|--|--|
| <i>Optical↔<br/>FlowLK</i>   | 1.0 |  |     |  |  |  |  |
| <i>Or</i>                    | 1.0 |  |     |  |  |  |  |
| <i>Phase</i>                 |     |  | 1.0 |  |  |  |  |
| <i>Gaussian↔<br/>Pyramid</i> | 1.0 |  |     |  |  |  |  |
| <i>Remap</i>                 | 1.0 |  |     |  |  |  |  |
| <i>Scale↔<br/>Image</i>      | 1.0 |  |     |  |  |  |  |
| <i>Sobel3x3</i>              | 1.0 |  |     |  |  |  |  |
| <i>Subtract</i>              | 1.0 |  | 1.0 |  |  |  |  |
| <i>Threshold</i>             | 1.0 |  |     |  |  |  |  |
| <i>WarpAffine</i>            | 1.0 |  |     |  |  |  |  |
| <i>Warp↔<br/>Perspective</i> | 1.0 |  |     |  |  |  |  |
| <i>Xor</i>                   | 1.0 |  |     |  |  |  |  |

### 2.12.2 Outputs

| <b>Vision<br/>Function</b>           | <b>U8</b> | <b>U16</b> | <b>S16</b> | <b>U32</b> | <b>S32</b> | <b>F32</b> | <b>color</b> |
|--------------------------------------|-----------|------------|------------|------------|------------|------------|--------------|
| <i>AbsDiff</i>                       | 1.0       |            | 1.0        |            |            |            |              |
| <i>Accumu-<br/>late</i>              |           |            | 1.0        |            |            |            |              |
| <i>Accumulate↔<br/>Squared</i>       |           |            | 1.0        |            |            |            |              |
| <i>Accumulate↔<br/>Weighted</i>      | 1.0       |            |            |            |            |            |              |
| <i>Add</i>                           | 1.0       |            | 1.0        |            |            |            |              |
| <i>And</i>                           | 1.0       |            |            |            |            |            |              |
| <i>Box3x3</i>                        | 1.0       |            |            |            |            |            |              |
| <i>Canny↔<br/>Edge↔<br/>Detector</i> | 1.0       |            |            |            |            |            |              |
| <i>Channel↔<br/>Combine</i>          |           |            |            |            |            |            | 1.0          |
| <i>Channel↔<br/>Extract</i>          | 1.0       |            |            |            |            |            |              |
| <i>Color↔<br/>Convert</i>            |           |            |            |            |            |            | 1.0          |
| <i>Convert↔<br/>Depth</i>            | 1.0       |            | 1.0        |            |            |            |              |
| <i>Convolve</i>                      | 1.0       |            | 1.0        |            |            |            |              |
| <i>Dilate3x3</i>                     | 1.0       |            |            |            |            |            |              |
| <i>Equalize↔<br/>Histogram</i>       | 1.0       |            |            |            |            |            |              |
| <i>Erode3x3</i>                      | 1.0       |            |            |            |            |            |              |
| <i>Fast↔<br/>Corners</i>             | 1.0       |            |            |            |            |            |              |

|                             |     |  |     |     |     |     |  |
|-----------------------------|-----|--|-----|-----|-----|-----|--|
| Gaussian3x3                 | 1.0 |  |     |     |     |     |  |
| Harris↔<br>Corners          | 1.0 |  |     |     |     |     |  |
| Half↔<br>Scale↔<br>Gaussian | 1.0 |  |     |     |     |     |  |
| Histogram                   |     |  |     |     | 1.0 |     |  |
| Integral↔<br>Image          |     |  |     | 1.0 |     |     |  |
| Table↔<br>Lookup            | 1.0 |  |     |     |     |     |  |
| Magnitude                   |     |  | 1.0 |     |     |     |  |
| MeanStd↔<br>Dev             |     |  |     |     |     | 1.0 |  |
| Median3x3                   | 1.0 |  |     |     |     |     |  |
| MinMax↔<br>Loc              | 1.0 |  | 1.0 |     | 1.0 |     |  |
| Multiply                    | 1.0 |  | 1.0 |     |     |     |  |
| Not                         | 1.0 |  |     |     |     |     |  |
| Optical↔<br>FlowLK          |     |  |     | 1.0 |     |     |  |
| Or                          | 1.0 |  |     |     |     |     |  |
| Phase                       | 1.0 |  |     |     |     |     |  |
|                             | 1.0 |  |     |     |     |     |  |
| Gaussian↔<br>Pyramid        |     |  |     |     |     |     |  |
| Remap                       | 1.0 |  |     |     |     |     |  |
| Scale↔<br>Image             | 1.0 |  |     |     |     |     |  |
| Sobel3x3                    |     |  | 1.0 |     |     |     |  |
| Subtract                    | 1.0 |  | 1.0 |     |     |     |  |
| Threshold                   | 1.0 |  |     |     |     |     |  |
| WarpAffine                  | 1.0 |  |     |     |     |     |  |
| Warp↔<br>Perspective        | 1.0 |  |     |     |     |     |  |
| Xor                         | 1.0 |  |     |     |     |     |  |

## 2.13 Lifecycles

### 2.13.1 OpenVX Context Lifecycle

The lifecycle of the context is very simple.

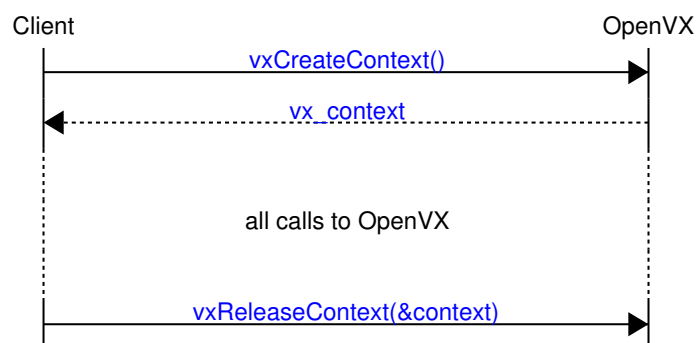


Figure 2.4: The lifecycle model for an OpenVX Context.

### 2.13.2 Graph Lifecycle

OpenVX has four main phases of graph lifecycle:

- *Construction* - Graphs are created via `vxCreateGraph`, and Nodes are connected together by data objects.
- *Verification* - The graphs are checked for consistency, correctness, and other conditions. Memory allocation may occur.
- *Execution* - The graphs are executed via `vxProcessGraph` or `vxScheduleGraph`. Between executions data may be updated by the client or some other external mechanism. The client of OpenVX may change reference of input data to a graph, but this may require the graph to be validated again by checking `vxIsGraphVerified`.
- *Deconstruction* - Graphs are released via `vxReleaseGraph`. All Nodes in the Graph are released.

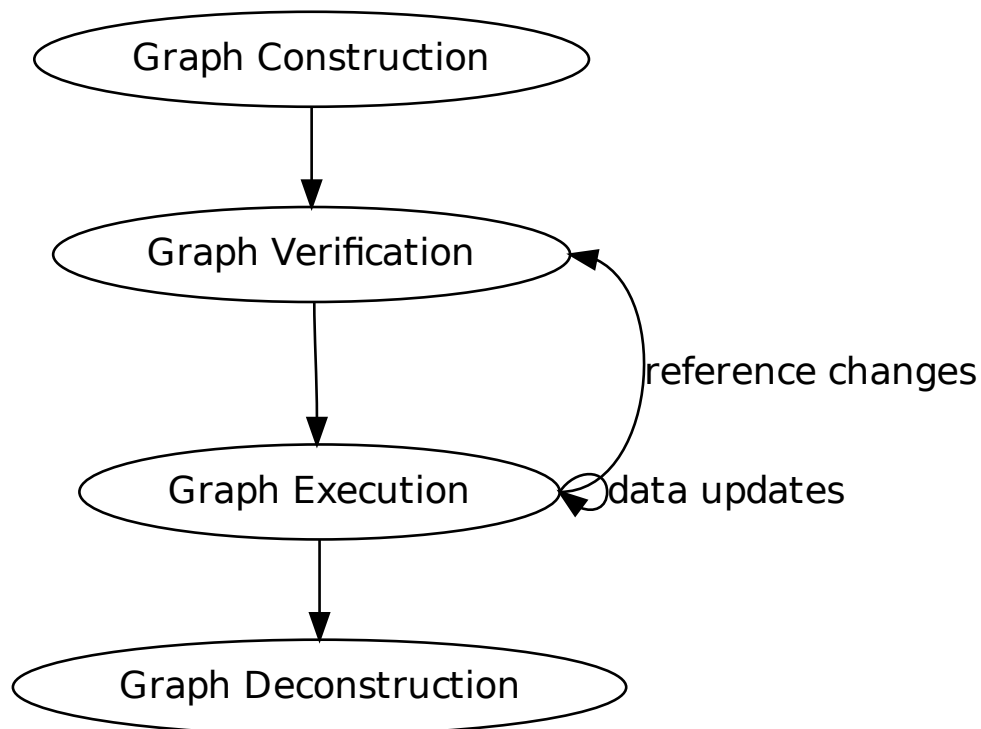


Figure 2.5: Graph Lifecycle

### 2.13.3 Data Object Lifecycle

All objects in OpenVX follow a similar lifecycle model. All objects are

- Created via `vxCreate<Object><Method>` or retrieved via `vxGet<Object><Method>` from the parent object if they are internally created.
- Used within Graphs or immediate functions as needed.
- Then objects must be released via `vxRelease<Object>` or via `vxReleaseContext` when all objects are released.



## OpenVX Image Lifecycle

This is an example of the Image Lifecycle using the OpenVX Framework API. This would also apply to other data types with changes to the types and function names.

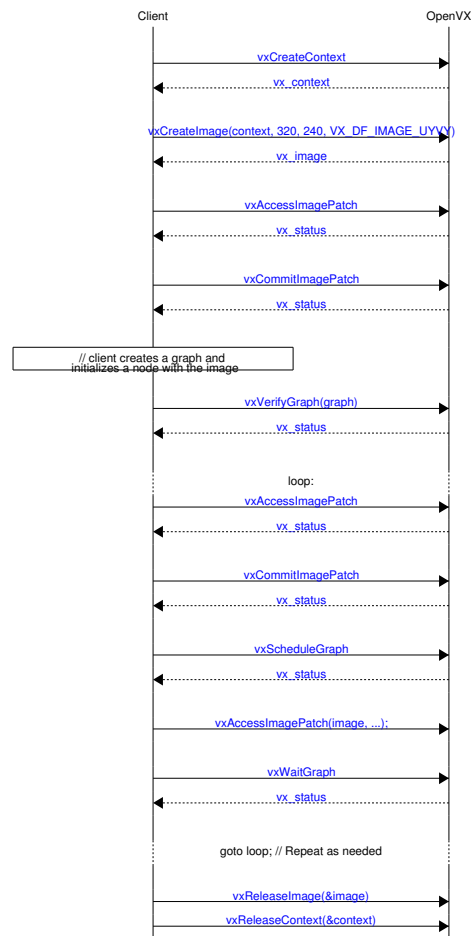


Figure 2.6: Image Object Lifecycle

## 2.14 Host Memory Data Object Access Patterns

For objects retrieved from OpenVX that are 2D in nature, such as `vx_image`, `vx_matrix`, and `vx_convolution`, the manner in which the host-side has access to these memory regions is well-defined. OpenVX uses a row-major storage (that is each unit in a column is memory-adjacent to its row adjacent unit). Two-dimensional objects are always created (using `vxCreateImage` or `vxCreateMatrix`) in width (columns) by height (rows) notation, with the arguments in that order. When accessing these structures in “C” with two-dimensional arrays of declared size, the user must therefore provide the array dimensions in the reverse of the order of the arguments to the Create function. This layout ensures row-wise storage in C on the host. A pointer could also be allocated for the matrix data and would have to be indexed in this row-major method.

### 2.14.1 Matrix Access Example

```

const vx_size columns = 3;
const vx_size rows = 4;
vx_matrix matrix = vxCreateMatrix(context,
 VX_TYPE_FLOAT32, columns, rows);
vx_status status = vxGetStatus((vx_reference)matrix);
if (status == VX_SUCCESS)
{
 vx_int32 j, i;
 #if defined(OPENVX_USE_C99)
 vx_float32 mat[rows][columns]; /* note: row major */
 #endif
}

```

```

#else
 vx_float32 *mat = (vx_float32 *)malloc(rows*columns*sizeof(
 vx_float32));
#endif
 if (vxReadMatrix(matrix, mat) == VX_SUCCESS) {
 for (j = 0; j < rows; j++)
 for (i = 0; i < columns; i++)
 mat[j][i] = (vx_float32)rand()/(vx_float32)RAND_MAX;
 }
 else
 mat[j*columns + i] = (vx_float32)rand()/(
 vx_float32)RAND_MAX;
#endif
 vxWriteMatrix(matrix, mat);
}
#endif
#if !defined(OPENVX_USE_C99)
 free(mat);
#endif
}

```

## 2.14.2 Image Access Example

*Images and Array differ slightly in how they are accessed due to more complex memory layout requirements.*

```

vx_status status = VX_SUCCESS;
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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x*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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y*addr.scale_y) /
 VX_SCALE_UNIT) +
 ((addr.stride_x*x*addr.scale_x) /
 VX_SCALE_UNIT);
 tmp[i] = pixel;
 }
 }

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

```

```

 }
}

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

```

### 2.14.3 Array Access Example

Arrays only require a single value, the stride, instead of the entire addressing structure that images need.

```

vx_size i, stride = sizeof(vx_size);
void *base = NULL;
/* access entire array at once */
vxAccessArrayRange(array, 0, num_items, &stride, &base,
VX_READ_AND_WRITE);
for (i = 0; i < num_items; i++)
{
 vxArrayItem(mystruct, base, i, stride).some_uint += i;
 vxArrayItem(mystruct, base, i, stride).some_double = 3.14f;
}
vxCommitArrayRange(array, 0, num_items, base);

```

Access/Commit pairs can also be called on individual elements of array using a method similar to this:

```

/* access each array item individually */
for (i = 0; i < num_items; i++)
{
 mystruct *myptr = NULL;
 vxAccessArrayRange(array, i, i+1, &stride, (void **)&myptr,
VX_READ_AND_WRITE);
 myptr->some_uint += 1;
 myptr->some_double = 3.14f;
 vxCommitArrayRange(array, i, i+1, (void *)myptr);
}

```

## 2.15 Extending OpenVX

Beyond *User Kernels* there are other mechanisms for vendors to extend features in OpenVX. These mechanisms are not available to User Kernels.

### 2.15.1 Extending Attributes

When extending attributes, vendors must use their assigned ID from `vx_vendor_id_e` in conjunction with the appropriate macros for creating new attributes with `VX_ATTRIBUTE_BASE`. The typical mechanism to extend a new attribute for some object type (for example a `vx_node` attribute from `VX_ID_TI`) would look like this:

```

enum {
 VX_NODE_ATTRIBUTE_TI_NEWTHING = VX_ATTRIBUTE_BASE(VX_ID_TI,
 VX_TYPE_NODE) + 0x0,
}

```

### 2.15.2 Vendor Custom Kernels

Vendors wanting to add more kernels to the base set supplied to OpenVX should provide a header of the form

```
#include <VX/vx_ext_<vendor>.h>
```

that contains definitions of each of the following.

- New Node Creation Function Prototype per function.

```

vx_node vxXYZNode(vx_graph graph, vx_image input,
 vx_uint32 value, vx_image output, vx_array temp);

```

- A new Kernel Enumeration(s) and Kernel String per function.

```
#define VX_KERNEL_NAME_KHR_XYZ "org.khronos.example.xyz"

#define VX_LIBRARY_XYZ (0x3) // assigned from Khronos, vendors control their own

enum vx_kernel_xyz_ext_e {
 VX_KERNEL_KHR_XYZ = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_XYZ) + 0x0,
 // up to 0xFFFF kernel enums can be created.
};
```

- A new VXU Function per function.

```
vx_status vxuXYZ(vx_context context, vx_image input,
 vx_uint32 value, vx_image output, vx_array temp);
```

*This should come with good documentation for each new part of the extension. Ideally, these sorts of extensions should not require linking to new objects to facilitate usage.*

### 2.15.3 Vendor Custom Extensions

Some extensions affect base vision functions and thus may be invisible to most users. In these circumstances, the vendor must report the supported extensions to the base nodes through the `VX_CONTEXT_ATTRIBUTE_EXTENSIONS` attribute on the context.

```
vx_char *tmp, *extensions = NULL;
vx_size size = 0;
vxQueryContext(context, VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE,
 &size, sizeof(size));
extensions = malloc(size);
vxQueryContext(context, VX_CONTEXT_ATTRIBUTE_EXTENSIONS,
 extensions, size);
```

Extensions in this list are dependent on the extension itself; they may or may not have a header and new kernels or framework feature or data objects. The common feature is that they are implemented and supported by the implementation vendor.

### 2.15.4 Hinting

The specification defines a Hinting API that allows Clients to feed information to the implementation for optional behavior changes. See [Framework: Hints](#). It is assumed that most of the hints will be vendor- or implementation-specific. Check with the OpenVX implementation vendor for information on vendor-specific extensions.

### 2.15.5 Directives

The specification defines a Directive API to control implementation behavior. See [Framework: Directives](#). This may allow things like disabling parallelism for debugging, enabling cache writing-through for some buffers, or any implementation-specific optimization.

## 2.16 Known Extensions to OpenVX

### 2.16.1 User Kernel Tiling

The User Kernel Tiling facility enables optimizations of the user kernels (e.g., locality of execution or parallelism) when performing computation on the image data. Modern processors have a diverse memory hierarchy that varies from relatively small but fast and expensive memory to relatively large but slow and inexpensive memory. Image data are typically too large to fit into the fast but small memory. The ability to break the image data into smaller sized units allows for optimized computation on these smaller units with fast memory access or parallel execution of a user kernel on multiple image tiles simultaneously. The OpenVX Graph Manager possesses the knowledge about the memory hierarchy of the platform and is hence in a position to break the image data into smaller units for memory optimization. Knowledge of the memory access pattern of an algorithm is key for the graph manager to enable optimizations.

The Khronos OpenVX Working Group will include this extension as part of the OpenVX 1.1 specification, contingent on community feedback.

# Chapter 3

## Module Documentation

### 3.1 Vision Functions

#### 3.1.1 Detailed Description

These are the base vision functions supported in OpenVX 1.0.

These functions were chosen as a subset of a larger pool of possible functions that fall under the following criteria:

- Applicable to Acceleration Hardware
- Very Common Usage
- Encumbrance Free

#### Modules

- [Absolute Difference](#)  
*Computes the absolute difference between two images.*
- [Accumulate](#)  
*Accumulates an input image into output image.*
- [Accumulate Squared](#)  
*Accumulates a squared value from an input image to an output image.*
- [Accumulate Weighted](#)  
*Accumulates a weighted value from an input image to an output image.*
- [Arithmetic Addition](#)  
*Performs addition between two images.*
- [Arithmetic Subtraction](#)  
*Performs subtraction between two images.*
- [Bitwise AND](#)  
*Performs a bitwise AND operation between two `VX_DF_IMAGE_U8` images.*
- [Bitwise EXCLUSIVE OR](#)  
*Performs a bitwise EXCLUSIVE OR (XOR) operation between two `VX_DF_IMAGE_U8` images.*
- [Bitwise INCLUSIVE OR](#)  
*Performs a bitwise INCLUSIVE OR operation between two `VX_DF_IMAGE_U8` images.*
- [Bitwise NOT](#)  
*Performs a bitwise NOT operation on a `VX_DF_IMAGE_U8` input image.*
- [Box Filter](#)  
*Computes a Box filter over a window of the input image.*
- [Canny Edge Detector](#)  
*Provides a Canny edge detector kernel.*
- [Channel Combine](#)

- Implements the Channel Combine Kernel.*

  - [Channel Extract](#)

*Implements the Channel Extraction Kernel.*
  - [Color Convert](#)

*Implements the Color Conversion Kernel.*
  - [Convert Bit depth](#)

*Converts image bit depth.*
  - [Custom Convolution](#)

*Convolve the input with the client supplied convolution matrix.*
  - [Dilate Image](#)

*Implements Dilation, which grows the white space in a `VX_DF_IMAGE_U8` Boolean image.*
  - [Equalize Histogram](#)

*Equalizes the histogram of a grayscale image.*
  - [Erode Image](#)

*Implements Erosion, which shrinks the white space in a `VX_DF_IMAGE_U8` Boolean image.*
  - [Fast Corners](#)

*Computes the corners in an image using a method based upon FAST9 algorithm suggested in [3] and with some updates from [4] with modifications described below.*
  - [Gaussian Filter](#)

*Computes a Gaussian filter over a window of the input image.*
  - [Harris Corners](#)

*Computes the Harris Corners of an image.*
  - [Histogram](#)

*Generates a distribution from an image.*
  - [Gaussian Image Pyramid](#)

*Computes a Gaussian Image Pyramid from an input image.*
  - [Integral Image](#)

*Computes the integral image of the input.*
  - [Magnitude](#)

*Implements the Gradient Magnitude Computation Kernel.*
  - [Mean and Standard Deviation](#)

*Computes the mean pixel value and the standard deviation of the pixels in the input image (which has a dimension width and height).*
  - [Median Filter](#)

*Computes a median pixel value over a window of the input image.*
  - [Min, Max Location](#)

*Finds the minimum and maximum values in an image and a location for each.*
  - [Optical Flow Pyramid \(LK\)](#)

*Computes the optical flow using the Lucas-Kanade method between two pyramid images.*
  - [Phase](#)

*Implements the Gradient Phase Computation Kernel.*
  - [Pixel-wise Multiplication](#)

*Performs element-wise multiplication between two images and a scalar value.*
  - [Remap](#)

*Maps output pixels in an image from input pixels in an image.*
  - [Scale Image](#)

*Implements the Image Resizing Kernel.*
  - [Sobel 3x3](#)

*Implements the Sobel Image Filter Kernel.*
  - [TableLookup](#)

*Implements the Table Lookup Image Kernel.*

- [Thresholding](#)

*Thresholds an input image and produces an output Boolean image.*

- [Warp Affine](#)

*Performs an affine transform on an image.*

- [Warp Perspective](#)

*Performs a perspective transform on an image.*

## 3.2 Absolute Difference

### 3.2.1 Detailed Description

Computes the absolute difference between two images.

Absolute Difference is computed by:

$$out(x,y) = |in_1(x,y) - in_2(x,y)|$$

When the two input parameters have type s16, the conceptual definition describing the overflow is:

```
uint16 uresult = (uint16) abs((int32) (a) - (int32) (b));
```

```
int16 result = uresult > 32767 ? 32767 : (int16) uresult;
```

### Functions

- **vx\_node** VX\_API\_CALL **vxAbsDiffNode** (**vx\_graph** graph, **vx\_image** in1, **vx\_image** in2, **vx\_image** out)  
[Graph] Creates an AbsDiff node.
- **vx\_status** VX\_API\_CALL **vxuAbsDiff** (**vx\_context** context, **vx\_image** in1, **vx\_image** in2, **vx\_image** out)  
[Immediate] Computes the absolute difference between two images.

### 3.2.2 Function Documentation

**vx\_node** VX\_API\_CALL **vxAbsDiffNode** ( **vx\_graph** graph, **vx\_image** in1, **vx\_image** in2, **vx\_image** out )

[Graph] Creates an AbsDiff node.

Parameters

|     |       |                                                                                               |
|-----|-------|-----------------------------------------------------------------------------------------------|
| in  | graph | The reference to the graph.                                                                   |
| in  | in1   | An input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.   |
| in  | in2   | An input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.   |
| out | out   | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

Return values

|         |                                                                                                                            |
|---------|----------------------------------------------------------------------------------------------------------------------------|
| vx_node | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|---------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** VX\_API\_CALL **vxuAbsDiff** ( **vx\_context** context, **vx\_image** in1, **vx\_image** in2, **vx\_image** out )

[Immediate] Computes the absolute difference between two images.

Parameters

|     |         |                                                                                               |
|-----|---------|-----------------------------------------------------------------------------------------------|
| in  | context | The reference to the overall context.                                                         |
| in  | in1     | An input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.   |
| in  | in2     | An input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.   |
| out | out     | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|            |                                                      |
|------------|------------------------------------------------------|
| VX_SUCCESS | Success                                              |
| *          | An error occurred. See <a href="#">vx_status_e</a> . |



## 3.3 Accumulate

### 3.3.1 Detailed Description

Accumulates an input image into output image.

Accumulation is computed by:

$$accum(x,y) = accum(x,y) + input(x,y)$$

The overflow policy used is [VX\\_CONVERT\\_POLICY\\_SATURATE](#).

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxAccumulateImageNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) accum)  
[Graph] Creates an accumulate node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuAccumulateImage](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) accum)  
[Immediate] Computes an accumulation.

### 3.3.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxAccumulateImageNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) accum )**

[Graph] Creates an accumulate node.

Parameters

|         |       |                                                             |
|---------|-------|-------------------------------------------------------------|
| in      | graph | The reference to the graph.                                 |
| in      | input | The input <a href="#">VX_DF_IMAGE_U8</a> image.             |
| in, out | accum | The accumulation image in <a href="#">VX_DF_IMAGE_S16</a> . |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuAccumulateImage](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) accum )**

[Immediate] Computes an accumulation.

Parameters

|         |         |                                                           |
|---------|---------|-----------------------------------------------------------|
| in      | context | The reference to the overall context.                     |
| in      | input   | The input <a href="#">VX_DF_IMAGE_U8</a> image.           |
| in, out | accum   | The accumulation image in <a href="#">VX_DF_IMAGE_S16</a> |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.4 Accumulate Squared

### 3.4.1 Detailed Description

Accumulates a squared value from an input image to an output image.

Accumulate squares is computed by:

$$accum(x,y) = saturate_{int16}(((uint16)accum(x,y) + (((uint16)input(x,y))^2) >> (shift)))$$

Where  $0 \leq shift \leq 15$

The overflow policy used is `VX_CONVERT_POLICY_SATURATE`.

### Functions

- `vx_node` `VX_API_CALL vxAccumulateSquareImageNode` (`vx_graph` graph, `vx_image` input, `vx_scalar` shift, `vx_image` accum)  
[Graph] Creates an accumulate square node.
- `vx_status` `VX_API_CALL vxuAccumulateSquareImage` (`vx_context` context, `vx_image` input, `vx_scalar` shift, `vx_image` accum)  
[Immediate] Computes a squared accumulation.

### 3.4.2 Function Documentation

**`vx_node` `VX_API_CALL vxAccumulateSquareImageNode` ( `vx_graph` graph, `vx_image` input, `vx_scalar` shift, `vx_image` accum )**

[Graph] Creates an accumulate square node.

Parameters

|         |       |                                                                                             |
|---------|-------|---------------------------------------------------------------------------------------------|
| in      | graph | The reference to the graph.                                                                 |
| in      | input | The input <code>VX_DF_IMAGE_U8</code> image.                                                |
| in      | shift | The input <code>VX_TYPE_UINT32</code> with a value in the range of $0 \leq shift \leq 15$ . |
| in, out | accum | The accumulation image in <code>VX_DF_IMAGE_S16</code> .                                    |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

**`vx_status` `VX_API_CALL vxuAccumulateSquareImage` ( `vx_context` context, `vx_image` input, `vx_scalar` shift, `vx_image` accum )**

[Immediate] Computes a squared accumulation.

Parameters

|         |         |                                                                                             |
|---------|---------|---------------------------------------------------------------------------------------------|
| in      | context | The reference to the overall context.                                                       |
| in      | input   | The input <code>VX_DF_IMAGE_U8</code> image.                                                |
| in      | shift   | A <code>VX_TYPE_UINT32</code> type, the input value with the range $0 \leq shift \leq 15$ . |
| in, out | accum   | The accumulation image in <code>VX_DF_IMAGE_S16</code>                                      |

Returns

A `vx_status_e` enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.5 Accumulate Weighted

### 3.5.1 Detailed Description

Accumulates a weighted value from an input image to an output image.

Weighted accumulation is computed by:

$$accum(x,y) = (1 - \alpha) * accum(x,y) + \alpha * input(x,y)$$

Where  $0 \leq \alpha \leq 1$  Conceptually, the rounding for this is defined as:

$$output(x,y) = uint8((1 - \alpha) * float32(int32(output(x,y))) + \alpha * float32(int32(input(x,y))))$$

### Functions

- **vx\_node** **VX\_API\_CALL** **vxAccumulateWeightedImageNode** (**vx\_graph** graph, **vx\_image** input, **vx\_scalar** alpha, **vx\_image** accum)

[Graph] Creates a weighted accumulate node.

- **vx\_status** **VX\_API\_CALL** **vxuAccumulateWeightedImage** (**vx\_context** context, **vx\_image** input, **vx\_scalar** scale, **vx\_image** accum)

[Immediate] Computes a weighted accumulation.

### 3.5.2 Function Documentation

**vx\_node** **VX\_API\_CALL** **vxAccumulateWeightedImageNode** ( **vx\_graph** graph, **vx\_image** input, **vx\_scalar** alpha, **vx\_image** accum )

[Graph] Creates a weighted accumulate node.

Parameters

|         |       |                                                                                                         |
|---------|-------|---------------------------------------------------------------------------------------------------------|
| in      | graph | The reference to the graph.                                                                             |
| in      | input | The input <b>VX_DF_IMAGE_U8</b> image.                                                                  |
| in      | alpha | The input <b>VX_TYPE_FLOAT32</b> scalar value with a value in the range of $0.0 \leq \alpha \leq 1.0$ . |
| in, out | accum | The <b>VX_DF_IMAGE_U8</b> accumulation image.                                                           |

Returns

**vx\_node**.

Return values

|                |                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------|
| <b>vx_node</b> | A node reference. Any possible errors preventing a successful creation should be checked using <b>vxGetStatus</b> |
|----------------|-------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL** **vxuAccumulateWeightedImage** ( **vx\_context** context, **vx\_image** input, **vx\_scalar** scale, **vx\_image** accum )

[Immediate] Computes a weighted accumulation.

Parameters

|         |         |                                                                                            |
|---------|---------|--------------------------------------------------------------------------------------------|
| in      | context | The reference to the overall context.                                                      |
| in      | input   | The input <b>VX_DF_IMAGE_U8</b> image.                                                     |
| in      | scale   | A <b>VX_TYPE_FLOAT32</b> type, the input value with the range $0.0 \leq \alpha \leq 1.0$ . |
| in, out | accum   | The <b>VX_DF_IMAGE_U8</b> accumulation image.                                              |

Returns

A **vx\_status\_e** enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.6 Arithmetic Addition

### 3.6.1 Detailed Description

Performs addition between two images.

Arithmetic addition is performed between the pixel values in two `VX_DF_IMAGE_U8` or `VX_DF_IMAGE_S16` images. The output image can be `VX_DF_IMAGE_U8` only if both source images are `VX_DF_IMAGE_U8` and the output image is explicitly set to `VX_DF_IMAGE_U8`. It is otherwise `VX_DF_IMAGE_S16`. If one of the input images is of type `VX_DF_IMAGE_S16`, all values are converted to `VX_DF_IMAGE_S16`. The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

$$out(x,y) = in_1(x,y) + in_2(x,y)$$

### Functions

- `vx_node` `VX_API_CALL vxAddNode` (`vx_graph` *graph*, `vx_image` *in1*, `vx_image` *in2*, `vx_enum` *policy*, `vx_image` *out*)  
[Graph] Creates an arithmetic addition node.
- `vx_status` `VX_API_CALL vxuAdd` (`vx_context` *context*, `vx_image` *in1*, `vx_image` *in2*, `vx_enum` *policy*, `vx_image` *out*)  
[Immediate] Performs arithmetic addition on pixel values in the input images.

### 3.6.2 Function Documentation

**`vx_node` `VX_API_CALL vxAddNode` ( `vx_graph` *graph*, `vx_image` *in1*, `vx_image` *in2*, `vx_enum` *policy*, `vx_image` *out* )**

[Graph] Creates an arithmetic addition node.

Parameters

|                  |               |                                                                                        |
|------------------|---------------|----------------------------------------------------------------------------------------|
| <code>in</code>  | <i>graph</i>  | The reference to the graph.                                                            |
| <code>in</code>  | <i>in1</i>    | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> .          |
| <code>in</code>  | <i>in2</i>    | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> .          |
| <code>in</code>  | <i>policy</i> | A <code>VX_TYPE_ENUM</code> of the <code>vx_convert_policy_e</code> enumeration.       |
| <code>out</code> | <i>out</i>    | The output image, a <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> image. |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

**`vx_status` `VX_API_CALL vxuAdd` ( `vx_context` *context*, `vx_image` *in1*, `vx_image` *in2*, `vx_enum` *policy*, `vx_image` *out* )**

[Immediate] Performs arithmetic addition on pixel values in the input images.

Parameters

|                 |                |                                                                            |
|-----------------|----------------|----------------------------------------------------------------------------|
| <code>in</code> | <i>context</i> | The reference to the overall context.                                      |
| <code>in</code> | <i>in1</i>     | A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image. |
| <code>in</code> | <i>in2</i>     | A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image. |
| <code>in</code> | <i>policy</i>  | A <code>vx_convert_policy_e</code> enumeration.                            |

|     |            |                                                                                               |
|-----|------------|-----------------------------------------------------------------------------------------------|
| out | <i>out</i> | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |
|-----|------------|-----------------------------------------------------------------------------------------------|

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.7 Arithmetic Subtraction

### 3.7.1 Detailed Description

Performs subtraction between two images.

Arithmetic subtraction is performed between the pixel values in two `VX_DF_IMAGE_U8` or two `VX_DF_IMAGE_S16` images. The output image can be `VX_DF_IMAGE_U8` only if both source images are `VX_DF_IMAGE_U8` and the output image is explicitly set to `VX_DF_IMAGE_U8`. It is otherwise `VX_DF_IMAGE_S16`. If one of the input images is of type `VX_DF_IMAGE_S16`, all values are converted to `VX_DF_IMAGE_S16`. The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

$$out(x,y) = in_1(x,y) - in_2(x,y)$$

### Functions

- `vx_node` `VX_API_CALL vxSubtractNode` (`vx_graph` graph, `vx_image` in1, `vx_image` in2, `vx_enum` policy, `vx_image` out)  
[Graph] Creates an arithmetic subtraction node.
- `vx_status` `VX_API_CALL vxuSubtract` (`vx_context` context, `vx_image` in1, `vx_image` in2, `vx_enum` policy, `vx_image` out)  
[Immediate] Performs arithmetic subtraction on pixel values in the input images.

### 3.7.2 Function Documentation

**`vx_node` `VX_API_CALL vxSubtractNode` ( `vx_graph` graph, `vx_image` in1, `vx_image` in2, `vx_enum` policy, `vx_image` out )**

[Graph] Creates an arithmetic subtraction node.

Parameters

|     |        |                                                                                               |
|-----|--------|-----------------------------------------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                                                   |
| in  | in1    | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> , the minuend.    |
| in  | in2    | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> , the subtrahend. |
| in  | policy | A <code>VX_TYPE_ENUM</code> of the <code>vx_convert_policy_e</code> enumeration.              |
| out | out    | The output image, a <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> image.        |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

**`vx_status` `VX_API_CALL vxuSubtract` ( `vx_context` context, `vx_image` in1, `vx_image` in2, `vx_enum` policy, `vx_image` out )**

[Immediate] Performs arithmetic subtraction on pixel values in the input images.

Parameters

|    |         |                                                                                            |
|----|---------|--------------------------------------------------------------------------------------------|
| in | context | The reference to the overall context.                                                      |
| in | in1     | A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image, the minuend.    |
| in | in2     | A <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> input image, the subtrahend. |



|     |               |                                                                                               |
|-----|---------------|-----------------------------------------------------------------------------------------------|
| in  | <i>policy</i> | A <a href="#">vx_convert_policy_e</a> enumeration.                                            |
| out | <i>out</i>    | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.8 Bitwise AND

### 3.8.1 Detailed Description

Performs a *bitwise AND* operation between two `VX_DF_IMAGE_U8` images.

Bitwise AND is computed by the following, for each bit in each pixel in the input images:

$$out(x,y) = in_1(x,y) \wedge in_2(x,y)$$

Or expressed as C code:

```
out(x,y) = in_1(x,y) & in_2(x,y)
```

### Functions

- `vx_node` `VX_API_CALL vxAndNode` (`vx_graph` `graph`, `vx_image` `in1`, `vx_image` `in2`, `vx_image` `out`)  
[Graph] Creates a bitwise AND node.
- `vx_status` `VX_API_CALL vxuAnd` (`vx_context` `context`, `vx_image` `in1`, `vx_image` `in2`, `vx_image` `out`)  
[Immediate] Computes the bitwise and between two images.

### 3.8.2 Function Documentation

`vx_node` `VX_API_CALL vxAndNode` ( `vx_graph` `graph`, `vx_image` `in1`, `vx_image` `in2`, `vx_image` `out` )

[Graph] Creates a bitwise AND node.

Parameters

|                  |                    |                                               |
|------------------|--------------------|-----------------------------------------------|
| <code>in</code>  | <code>graph</code> | The reference to the graph.                   |
| <code>in</code>  | <code>in1</code>   | A <code>VX_DF_IMAGE_U8</code> input image.    |
| <code>in</code>  | <code>in2</code>   | A <code>VX_DF_IMAGE_U8</code> input image.    |
| <code>out</code> | <code>out</code>   | The <code>VX_DF_IMAGE_U8</code> output image. |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuAnd` ( `vx_context` `context`, `vx_image` `in1`, `vx_image` `in2`, `vx_image` `out` )

[Immediate] Computes the bitwise and between two images.

Parameters

|                  |                      |                                               |
|------------------|----------------------|-----------------------------------------------|
| <code>in</code>  | <code>context</code> | The reference to the overall context.         |
| <code>in</code>  | <code>in1</code>     | A <code>VX_DF_IMAGE_U8</code> input image     |
| <code>in</code>  | <code>in2</code>     | A <code>VX_DF_IMAGE_U8</code> input image     |
| <code>out</code> | <code>out</code>     | The <code>VX_DF_IMAGE_U8</code> output image. |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.9 Bitwise EXCLUSIVE OR

### 3.9.1 Detailed Description

Performs a *bitwise EXCLUSIVE OR* (XOR) operation between two [VX\\_DF\\_IMAGE\\_U8](#) images.

Bitwise XOR is computed by the following, for each bit in each pixel in the input images:

$$out(x,y) = in_1(x,y) \oplus in_2(x,y)$$

Or expressed as C code:

```
out(x,y) = in_1(x,y) ^ in_2(x,y)
```

### Functions

- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxuXor](#) ([vx\\_context](#) context, [vx\\_image](#) in1, [vx\\_image](#) in2, [vx\\_image](#) out)  
[Immediate] Computes the bitwise exclusive-or between two images.
- [vx\\_node](#) [VX\\_API\\_CALL](#) [vxXorNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) in1, [vx\\_image](#) in2, [vx\\_image](#) out)  
[Graph] Creates a bitwise EXCLUSIVE OR node.

### 3.9.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL](#) [vxXorNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) in1, [vx\\_image](#) in2, [vx\\_image](#) out )**

[Graph] Creates a bitwise EXCLUSIVE OR node.

Parameters

|     |       |                                                  |
|-----|-------|--------------------------------------------------|
| in  | graph | The reference to the graph.                      |
| in  | in1   | A <a href="#">VX_DF_IMAGE_U8</a> input image.    |
| in  | in2   | A <a href="#">VX_DF_IMAGE_U8</a> input image.    |
| out | out   | The <a href="#">VX_DF_IMAGE_U8</a> output image. |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL](#) [vxuXor](#) ( [vx\\_context](#) context, [vx\\_image](#) in1, [vx\\_image](#) in2, [vx\\_image](#) out )**

[Immediate] Computes the bitwise exclusive-or between two images.

Parameters

|     |         |                                                  |
|-----|---------|--------------------------------------------------|
| in  | context | The reference to the overall context.            |
| in  | in1     | A <a href="#">VX_DF_IMAGE_U8</a> input image     |
| in  | in2     | A <a href="#">VX_DF_IMAGE_U8</a> input image     |
| out | out     | The <a href="#">VX_DF_IMAGE_U8</a> output image. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.10 Bitwise INCLUSIVE OR

### 3.10.1 Detailed Description

Performs a *bitwise INCLUSIVE OR* operation between two `VX_DF_IMAGE_U8` images.

Bitwise INCLUSIVE OR is computed by the following, for each bit in each pixel in the input images:

$$out(x,y) = in_1(x,y) \vee in_2(x,y)$$

Or expressed as C code:

```
out(x,y) = in_1(x,y) | in_2(x,y)
```

### Functions

- `vx_node` `VX_API_CALL vxOrNode` (`vx_graph` *graph*, `vx_image` *in1*, `vx_image` *in2*, `vx_image` *out*)  
[Graph] Creates a bitwise INCLUSIVE OR node.
- `vx_status` `VX_API_CALL vxuOr` (`vx_context` *context*, `vx_image` *in1*, `vx_image` *in2*, `vx_image` *out*)  
[Immediate] Computes the bitwise inclusive-or between two images.

### 3.10.2 Function Documentation

`vx_node` `VX_API_CALL vxOrNode` ( `vx_graph` *graph*, `vx_image` *in1*, `vx_image` *in2*, `vx_image` *out* )

[Graph] Creates a bitwise INCLUSIVE OR node.

Parameters

|                  |              |                                               |
|------------------|--------------|-----------------------------------------------|
| <code>in</code>  | <i>graph</i> | The reference to the graph.                   |
| <code>in</code>  | <i>in1</i>   | A <code>VX_DF_IMAGE_U8</code> input image.    |
| <code>in</code>  | <i>in2</i>   | A <code>VX_DF_IMAGE_U8</code> input image.    |
| <code>out</code> | <i>out</i>   | The <code>VX_DF_IMAGE_U8</code> output image. |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuOr` ( `vx_context` *context*, `vx_image` *in1*, `vx_image` *in2*, `vx_image` *out* )

[Immediate] Computes the bitwise inclusive-or between two images.

Parameters

|                  |                |                                               |
|------------------|----------------|-----------------------------------------------|
| <code>in</code>  | <i>context</i> | The reference to the overall context.         |
| <code>in</code>  | <i>in1</i>     | A <code>VX_DF_IMAGE_U8</code> input image     |
| <code>in</code>  | <i>in2</i>     | A <code>VX_DF_IMAGE_U8</code> input image     |
| <code>out</code> | <i>out</i>     | The <code>VX_DF_IMAGE_U8</code> output image. |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.11 Bitwise NOT

### 3.11.1 Detailed Description

Performs a *bitwise NOT* operation on a [VX\\_DF\\_IMAGE\\_U8](#) input image.

Bitwise NOT is computed by the following, for each bit in each pixel in the input image:

$$out(x,y) = \overline{in(x,y)}$$

Or expressed as C code:

```
out(x,y) = ~in_1(x,y)
```

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxNotNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output)  
[Graph] Creates a bitwise NOT node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuNot](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output)  
[Immediate] Computes the bitwise not of an image.

### 3.11.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxNotNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output )**

[Graph] Creates a bitwise NOT node.

Parameters

|     |        |                                                  |
|-----|--------|--------------------------------------------------|
| in  | graph  | The reference to the graph.                      |
| in  | input  | A <a href="#">VX_DF_IMAGE_U8</a> input image.    |
| out | output | The <a href="#">VX_DF_IMAGE_U8</a> output image. |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuNot](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output )**

[Immediate] Computes the bitwise not of an image.

Parameters

|     |         |                                                  |
|-----|---------|--------------------------------------------------|
| in  | context | The reference to the overall context.            |
| in  | input   | The <a href="#">VX_DF_IMAGE_U8</a> input image   |
| out | output  | The <a href="#">VX_DF_IMAGE_U8</a> output image. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |



## 3.12 Box Filter

### 3.12.1 Detailed Description

Computes a Box filter over a window of the input image.

This filter uses the following convolution matrix:

$$\mathbf{K}_{box} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \frac{1}{9}$$

### Functions

- `vx_node` `VX_API_CALL vxBox3x3Node (vx_graph graph, vx_image input, vx_image output)`  
[Graph] Creates a Box Filter Node.
- `vx_status` `VX_API_CALL vxuBox3x3 (vx_context context, vx_image input, vx_image output)`  
[Immediate] Computes a box filter on the image by a 3x3 window.

### 3.12.2 Function Documentation

`vx_node` `VX_API_CALL vxBox3x3Node ( vx_graph graph, vx_image input, vx_image output )`

[Graph] Creates a Box Filter Node.

Parameters

|     |               |                                                         |
|-----|---------------|---------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                             |
| in  | <i>input</i>  | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| out | <i>output</i> | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

`vx_node`.

Return values

|                |                                                                                                                         |
|----------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuBox3x3 ( vx_context context, vx_image input, vx_image output )`

[Immediate] Computes a box filter on the image by a 3x3 window.

Parameters

|     |                |                                                         |
|-----|----------------|---------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                   |
| in  | <i>input</i>   | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| out | <i>output</i>  | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                   |
|-------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                           |
| *                       | An error occurred. See <code>vx_status_e</code> . |

## 3.13 Canny Edge Detector

### 3.13.1 Detailed Description

Provides a Canny edge detector kernel.

This function implements an edge detection algorithm similar to that described in [2]. The main components of the algorithm are:

- Gradient magnitude and orientation computation using a noise resistant operator (Sobel).
- Non-maximum suppression of the gradient magnitude, using the gradient orientation information.
- Tracing edges in the modified gradient image using hysteresis thresholding to produce a binary result.

The details of each of these steps are described below.

- **Gradient Computation:** Conceptually, the input image is convolved with vertical and horizontal Sobel kernels of the size indicated by the *gradient\_size* parameter. The Sobel kernels used for the gradient computation shall be as shown below. The two resulting directional gradient images ( $dx$  and  $dy$ ) are then used to compute a gradient magnitude image and a gradient orientation image. The norm used to compute the gradient magnitude is indicated by the *norm\_type* parameter, so the magnitude may be  $|dx| + |dy|$  for *VX\_NORM\_L1* or  $\sqrt{dx^2 + dy^2}$  for *VX\_NORM\_L2*. The gradient orientation image is quantized into 4 values: 0, 45, 90, and 135 degrees.

- For gradient size 3:

$$\mathbf{sobel}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{sobel}_y = \text{transpose}(\mathbf{sobel}_x) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- For gradient size 5:

$$\mathbf{sobel}_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

$$\mathbf{sobel}_y = \text{transpose}(\mathbf{sobel}_x)$$

- For gradient size 7:

$$\mathbf{sobel}_x = \begin{bmatrix} -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -20 & -80 & -100 & 0 & 100 & 80 & 20 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \end{bmatrix}$$

$$\mathbf{sobel}_y = \text{transpose}(\mathbf{sobel}_x)$$

- **Non-Maximum Suppression:** This is then applied such that a pixel is retained as a potential edge pixel if and only if its magnitude is greater than or equal to the pixels in the direction perpendicular to its edge orientation. For example, if the pixel's orientation is 0 degrees, it is only retained if its gradient magnitude is larger than that of the pixels at 90 and 270 degrees to it. If a pixel is suppressed via this condition, it must not appear as an edge pixel in the final output, i.e., its value must be 0 in the final output.
- **Edge Tracing:** The final edge pixels in the output are identified via a double thresholded hysteresis procedure. All retained pixels with magnitude above the *high* threshold are marked as known edge pixels (valued 255) in the final output image. All pixels with magnitudes less than or equal to the *low* threshold must not be marked as edge pixels in the final output. For the pixels in between the thresholds, edges are traced and marked as edges (255) in the output. This can be done by starting at the known edge pixels and moving in all eight directions recursively until the gradient magnitude is less than or equal to the low threshold.

- **Caveats:** The intermediate results described above are conceptual only; so for example, the implementation may not actually construct the gradient images and non-maximum-suppressed images. Only the final binary (0 or 255 valued) output image must be computed so that it matches the result of a final image constructed as described above.

## Enumerations

- enum `vx_norm_type_e` {  
`VX_NORM_L1` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_NORM_TYPE` << 12)) + 0x0,  
`VX_NORM_L2` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_NORM_TYPE` << 12)) + 0x1 }  
*A normalization type.*

## Functions

- `vx_node` `VX_API_CALL vxCannyEdgeDetectorNode` (`vx_graph` graph, `vx_image` input, `vx_threshold` hyst, `vx_int32` gradient\_size, `vx_enum` norm\_type, `vx_image` output)  
*[Graph] Creates a Canny Edge Detection Node.*
- `vx_status` `VX_API_CALL vxuCannyEdgeDetector` (`vx_context` context, `vx_image` input, `vx_threshold` hyst, `vx_int32` gradient\_size, `vx_enum` norm\_type, `vx_image` output)  
*[Immediate] Computes Canny Edges on the input image into the output image.*

### 3.13.2 Enumeration Type Documentation

enum `vx_norm_type_e`

A normalization type.

See also

[Canny Edge Detector](#)

Enumerator

**`VX_NORM_L1`** The L1 normalization.

**`VX_NORM_L2`** The L2 normalization.

Definition at line 1136 of file `vx_types.h`.

### 3.13.3 Function Documentation

`vx_node` `VX_API_CALL vxCannyEdgeDetectorNode` ( `vx_graph` graph, `vx_image` input, `vx_threshold` hyst, `vx_int32` gradient\_size, `vx_enum` norm\_type, `vx_image` output )

[Graph] Creates a Canny Edge Detection Node.

Parameters

|     |               |                                                                                                               |
|-----|---------------|---------------------------------------------------------------------------------------------------------------|
| in  | graph         | The reference to the graph.                                                                                   |
| in  | input         | The input <code>VX_DF_IMAGE_U8</code> image.                                                                  |
| in  | hyst          | The double threshold for hysteresis.                                                                          |
| in  | gradient_size | The size of the Sobel filter window, must support at least 3, 5, and 7.                                       |
| in  | norm_type     | A flag indicating the norm used to compute the gradient, <code>VX_NORM_L1</code> or <code>VX_NORM_L2</code> . |
| out | output        | The output image in <code>VX_DF_IMAGE_U8</code> format with values either 0 or 255.                           |

Returns

`vx_node`.

## Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuCannyEdgeDetector ( vx\_context *context*, vx\_image *input*, vx\_threshold *hyst*, vx\_int32 *gradient\_size*, vx\_enum *norm\_type*, vx\_image *output* )**

[Immediate] Computes Canny Edges on the input image into the output image.

## Parameters

|     |                      |                                                                                    |
|-----|----------------------|------------------------------------------------------------------------------------|
| in  | <i>context</i>       | The reference to the overall context.                                              |
| in  | <i>input</i>         | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                    |
| in  | <i>hyst</i>          | The double threshold for hysteresis.                                               |
| in  | <i>gradient_size</i> | The size of the Sobel filter window, must support at least 3, 5 and 7.             |
| in  | <i>norm_type</i>     | A flag indicating the norm used to compute the gradient, VX_NORM_L1 or VX_NORM_L2. |
| out | <i>output</i>        | The output image in <a href="#">VX_DF_IMAGE_U8</a> format.                         |

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.14 Channel Combine

### 3.14.1 Detailed Description

Implements the Channel Combine Kernel.

This kernel takes multiple [VX\\_DF\\_IMAGE\\_U8](#) planes to recombine them into a multi-planar or interleaved format from [vx\\_df\\_image\\_e](#). The user must specify only the number of channels that are appropriate for the combining operation. If a user specifies more channels than necessary, the operation results in an error. For the case where the destination image is a format with subsampling, the input channels are expected to have been subsampled before combining (by stretching and resizing).

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxChannelCombineNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) plane0, [vx\\_image](#) plane1, [vx\\_image](#) plane2, [vx\\_image](#) plane3, [vx\\_image](#) output)  
[Graph] Creates a channel combine node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuChannelCombine](#) ([vx\\_context](#) context, [vx\\_image](#) plane0, [vx\\_image](#) plane1, [vx\\_image](#) plane2, [vx\\_image](#) plane3, [vx\\_image](#) output)  
[Immediate] Invokes an immediate Channel Combine.

### 3.14.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxChannelCombineNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) plane0, [vx\\_image](#) plane1, [vx\\_image](#) plane2, [vx\\_image](#) plane3, [vx\\_image](#) output )**

[Graph] Creates a channel combine node.

Parameters

|     |        |                                                                                          |
|-----|--------|------------------------------------------------------------------------------------------|
| in  | graph  | The graph reference.                                                                     |
| in  | plane0 | The plane that forms channel 0. Must be <a href="#">VX_DF_IMAGE_U8</a> .                 |
| in  | plane1 | The plane that forms channel 1. Must be <a href="#">VX_DF_IMAGE_U8</a> .                 |
| in  | plane2 | [optional] The plane that forms channel 2. Must be <a href="#">VX_DF_IMAGE_U8</a> .      |
| in  | plane3 | [optional] The plane that forms channel 3. Must be <a href="#">VX_DF_IMAGE_U8</a> .      |
| out | output | The output image. The format of the image must be defined, even if the image is virtual. |

See also

[VX\\_KERNEL\\_CHANNEL\\_COMBINE](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuChannelCombine](#) ( [vx\\_context](#) context, [vx\\_image](#) plane0, [vx\\_image](#) plane1, [vx\\_image](#) plane2, [vx\\_image](#) plane3, [vx\\_image](#) output )**

[Immediate] Invokes an immediate Channel Combine.

Parameters

|     |                |                                                                                     |
|-----|----------------|-------------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                               |
| in  | <i>plane0</i>  | The plane that forms channel 0. Must be <a href="#">VX_DF_IMAGE_U8</a> .            |
| in  | <i>plane1</i>  | The plane that forms channel 1. Must be <a href="#">VX_DF_IMAGE_U8</a> .            |
| in  | <i>plane2</i>  | [optional] The plane that forms channel 2. Must be <a href="#">VX_DF_IMAGE_U8</a> . |
| in  | <i>plane3</i>  | [optional] The plane that forms channel 3. Must be <a href="#">VX_DF_IMAGE_U8</a> . |
| out | <i>output</i>  | The output image.                                                                   |

#### Returns

A [vx\\_status\\_e](#) enumeration.

#### Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.15 Channel Extract

### 3.15.1 Detailed Description

Implements the Channel Extraction Kernel.

This kernel removes a single [VX\\_DF\\_IMAGE\\_U8](#) channel (plane) from a multi-planar or interleaved image format from [vx\\_df\\_image\\_e](#).

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxChannelExtractNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_enum](#) channel, [vx\\_image](#) output)  
[Graph] Creates a channel extract node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuChannelExtract](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_enum](#) channel, [vx\\_image](#) output)  
[Immediate] Invokes an immediate Channel Extract.

### 3.15.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxChannelExtractNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_enum](#) channel, [vx\\_image](#) output )**

[Graph] Creates a channel extract node.

Parameters

|     |         |                                                                                                 |
|-----|---------|-------------------------------------------------------------------------------------------------|
| in  | graph   | The reference to the graph.                                                                     |
| in  | input   | The input image. Must be one of the defined <a href="#">vx_df_image_e</a> multi-planar formats. |
| in  | channel | The <a href="#">vx_channel_e</a> channel to extract.                                            |
| out | output  | The output image. Must be <a href="#">VX_DF_IMAGE_U8</a> .                                      |

See also

[VX\\_KERNEL\\_CHANNEL\\_EXTRACT](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuChannelExtract](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_enum](#) channel, [vx\\_image](#) output )**

[Immediate] Invokes an immediate Channel Extract.

Parameters

|     |         |                                                                                                |
|-----|---------|------------------------------------------------------------------------------------------------|
| in  | context | The reference to the overall context.                                                          |
| in  | input   | The input image. Must be one of the defined <a href="#">vx_df_image_e</a> multiplanar formats. |
| in  | channel | The <a href="#">vx_channel_e</a> enumeration to extract.                                       |
| out | output  | The output image. Must be <a href="#">VX_DF_IMAGE_U8</a> .                                     |

Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |



## 3.16 Color Convert

### 3.16.1 Detailed Description

Implements the Color Conversion Kernel.

This kernel converts an image of a designated `vx_df_image_e` format to another `vx_df_image_e` format for those combinations listed in the below table, where the columns are output types and the rows are input types. The API version first supporting the conversion is also listed.

| I/O  | RGB | RGBX | NV12 | NV21 | UYVY | YUYV | IYUV | YUV4 |
|------|-----|------|------|------|------|------|------|------|
| RGB  |     | 1.0  | 1.0  |      |      |      | 1.0  | 1.0  |
| RGBX | 1.0 |      | 1.0  |      |      |      | 1.0  | 1.0  |
| NV12 | 1.0 | 1.0  |      |      |      |      | 1.0  | 1.0  |
| NV21 | 1.0 | 1.0  |      |      |      |      | 1.0  | 1.0  |
| UYVY | 1.0 | 1.0  | 1.0  |      |      |      | 1.0  |      |
| YUYV | 1.0 | 1.0  | 1.0  |      |      |      | 1.0  |      |
| IYUV | 1.0 | 1.0  | 1.0  |      |      |      |      | 1.0  |
| YUV4 |     |      |      |      |      |      |      |      |

The `vx_df_image_e` encoding, held in the `VX_IMAGE_ATTRIBUTE_FORMAT` attribute, describes the data layout. The interpretation of the colors is determined by the `VX_IMAGE_ATTRIBUTE_SPACE` (see `vx_color_space_e`) and `VX_IMAGE_ATTRIBUTE_RANGE` (see `vx_channel_range_e`) attributes of the image. OpenVX 1.0 implementations are required only to support images of `VX_COLOR_SPACE_BT709` and `VX_CHANNEL_RANGE_FULL`.

If the channel range is defined as `VX_CHANNEL_RANGE_FULL`, the conversion between the real number and integer quantizations of color channels is defined for red, green, blue, and Y as:

$$value_{real} = \frac{value_{integer}}{256.0}$$

$$value_{integer} = \max(0, \min(255, \text{floor}(value_{real} * 256.0)))$$

For the U and V channels, the conversion between real number and integer quantizations is:

$$value_{real} = \frac{(value_{integer} - 128.0)}{256.0}$$

$$value_{integer} = \max(0, \min(255, \text{floor}((value_{real} * 256.0) + 128.0)))$$

If the channel range is defined as `VX_CHANNEL_RANGE_RESTRICTED`, the conversion between the integer quantizations of color channels and the continuous representations is defined for red, green, blue, and Y as:

$$value_{real} = \frac{(value_{integer} - 16.0)}{219.0}$$

$$value_{integer} = \max(0, \min(255, \text{floor}((value_{real} * 219.0) + 16.5)))$$

For the U and V channels, the conversion between real number and integer quantizations is:

$$value_{real} = \frac{(value_{integer} - 128.0)}{224.0}$$

$$value_{integer} = \max(0, \min(255, \text{floor}((value_{real} * 224.0) + 128.5)))$$

The conversions between nonlinear-intensity Y'PbPr and R'G'B' real numbers are:

$$R' = Y' + 2(1 - K_r)Pr$$

$$B' = Y' + 2(1 - K_b)Pb$$

$$G' = Y' - \frac{2(K_r(1 - K_r)Pr + K_b(1 - K_b)Pb)}{1 - K_r - K_b}$$

$$Y' = (K_r * R') + (K_b * B') + (1 - K_r - K_b)G'$$

$$Pb = \frac{B'}{2} - \frac{(R' * K_r) + G'(1 - K_r - K_b)}{2(1 - K_b)}$$

$$Pr = \frac{R'}{2} - \frac{(B' * K_b) + G'(1 - K_r - K_b)}{2(1 - K_r)}$$

The means of reconstructing Pb and Pr values from chroma-downsampled formats is implementation-defined. In [VX\\_COLOR\\_SPACE\\_BT601\\_525](#) or [VX\\_COLOR\\_SPACE\\_BT601\\_625](#):

$$K_r = 0.299$$

$$K_b = 0.114$$

In [VX\\_COLOR\\_SPACE\\_BT709](#):

$$K_r = 0.2126$$

$$K_b = 0.0722$$

In all cases, for the purposes of conversion, these colour representations are interpreted as nonlinear in intensity, as defined by the BT.601, BT.709, and sRGB specifications. That is, the encoded colour channels are nonlinear R', G' and B', Y', Pb, and Pr.

Each channel of the R'G'B' representation can be converted to and from a linear-intensity RGB channel by these formulae:

$$value_{nonlinear} = 1.099 * value_{linear}^{0.45} - 0.099 \quad \text{for } 1 \geq value_{linear} \geq 0.018$$

$$value_{nonlinear} = 4.500 * value_{linear} \quad \text{for } 0.018 > value_{linear} \geq 0$$

$$value_{linear} = \left( \frac{value_{nonlinear} + 0.099}{1.099} \right)^{\frac{1}{0.45}} \quad \text{for } 1 \geq value_{nonlinear} > 0.081$$

$$value_{linear} = \frac{value_{nonlinear}}{4.5} \quad \text{for } 0.081 \geq value_{nonlinear} \geq 0$$

As the different color spaces have different RGB primaries, a conversion between them must transform the color coordinates into the new RGB space. Working with linear RGB values, the conversion formulae are:

$$R_{BT601\_525} = R_{BT601\_625} * 1.112302 + G_{BT601\_625} * -0.102441 + B_{BT601\_625} * -0.009860$$

$$G_{BT601\_525} = R_{BT601\_625} * -0.020497 + G_{BT601\_625} * 1.037030 + B_{BT601\_625} * -0.016533$$

$$B_{BT601\_525} = R_{BT601\_625} * 0.001704 + G_{BT601\_625} * 0.016063 + B_{BT601\_625} * 0.982233$$

$$R_{BT601\_525} = R_{BT709} * 1.065379 + G_{BT709} * -0.055401 + B_{BT709} * -0.009978$$

$$G_{BT601\_525} = R_{BT709} * -0.019633 + G_{BT709} * 1.036363 + B_{BT709} * -0.016731$$

$$B_{BT601\_525} = R_{BT709} * 0.001632 + G_{BT709} * 0.004412 + B_{BT709} * 0.993956$$

$$R_{BT601\_625} = R_{BT601\_525} * 0.900657 + G_{BT601\_525} * 0.088807 + B_{BT601\_525} * 0.010536$$

$$G_{BT601\_625} = R_{BT601\_525} * 0.017772 + G_{BT601\_525} * 0.965793 + B_{BT601\_525} * 0.016435$$

$$B_{BT601\_625} = R_{BT601\_525} * -0.001853 + G_{BT601\_525} * -0.015948 + B_{BT601\_525} * 1.017801$$

$$R_{BT601\_625} = R_{BT709} * 0.957815 + G_{BT709} * 0.042185$$

$$G_{BT601\_625} = G_{BT709}$$

$$B_{BT601\_625} = G_{BT709} * -0.011934 + B_{BT709} * 1.011934$$

$$R_{BT709} = R_{BT601\_525} * 0.939542 + G_{BT601\_525} * 0.050181 + B_{BT601\_525} * 0.010277$$

$$G_{BT709} = R_{BT601\_525} * 0.017772 + G_{BT601\_525} * 0.965793 + B_{BT601\_525} * 0.016435$$

$$B_{BT709} = R_{BT601\_525} * -0.001622 + G_{BT601\_525} * -0.004370 + B_{BT601\_525} * 1.005991$$

$$R_{BT709} = R_{BT601\_625} * 1.044043 + G_{BT601\_625} * -0.044043$$

$$G_{BT709} = G_{BT601\_625}$$

$$B_{BT709} = G_{BT601\_625} * 0.011793 + B_{BT601\_625} * 0.988207$$

A conversion between one YUV color space and another may therefore consist of the following transformations:

1. Convert quantized Y'CbCr ("YUV") to continuous, nonlinear Y'PbPr.
2. Convert continuous Y'PbPr to continuous, nonlinear R'G'B'.
3. Convert nonlinear R'G'B' to linear-intensity RGB (gamma-correction).
4. Convert linear RGB from the first color space to linear RGB in the second color space.
5. Convert linear RGB to nonlinear R'G'B' (gamma-conversion).
6. Convert nonlinear R'G'B' to Y'PbPr.
7. Convert continuous Y'PbPr to quantized Y'CbCr ("YUV").

The above formulae and constants are defined in the ITU [BT . 601](#) and [BT . 709](#) specifications. The formulae for converting between RGB primaries can be derived from the specified primary chromaticity values and the specified white point by solving for the relative intensity of the primaries.

## Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxColorConvertNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output)  
[Graph] Creates a color conversion node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuColorConvert](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output)  
[Immediate] Invokes an immediate Color Conversion.

### 3.16.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxColorConvertNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output )**

[Graph] Creates a color conversion node.

Parameters

|     |        |                                        |
|-----|--------|----------------------------------------|
| in  | graph  | The reference to the graph.            |
| in  | input  | The input image from which to convert. |
| out | output | The output image to which to convert.  |

See also

[VX\\_KERNEL\\_COLOR\\_CONVERT](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuColorConvert](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output )**

[Immediate] Invokes an immediate Color Conversion.

Parameters

|     |         |                                       |
|-----|---------|---------------------------------------|
| in  | context | The reference to the overall context. |
| in  | input   | The input image.                      |
| out | output  | The output image.                     |

Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.17 Convert Bit depth

### 3.17.1 Detailed Description

Converts image bit depth.

This kernel converts an image from some source bit-depth to another bit-depth as described by the table below. If the input value is unsigned the shift must be in zeros. If the input value is signed, the shift used must be an arithmetic shift. The columns in the table below are the output types and the rows are the input types. The API version on which conversion is supported is also listed. (An X denotes an invalid operation.)

| I/O | U8  | U16 | S16 | U32 | S32 |
|-----|-----|-----|-----|-----|-----|
| U8  | X   |     | 1.0 |     |     |
| U16 |     | X   | X   |     |     |
| S16 | 1.0 | X   | X   |     |     |
| U32 |     |     |     | X   | X   |
| S32 |     |     |     | X   | X   |

**Conversion Type** The table below identifies the conversion types for the allowed bith depth conversions.

| From | To  | Conversion Type |
|------|-----|-----------------|
| U8   | S16 | Up-conversion   |
| S16  | U8  | Down-conversion |

**Convert Policy** Down-conversions with `VX_CONVERT_POLICY_WRAP` follow this equation:

```
output(x,y) = ((uint8) (input(x,y) >> shift));
```

Down-conversions with `VX_CONVERT_POLICY_SATURATE` follow this equation:

```
int16 value = input(x,y) >> shift;
value = value < 0 ? 0 : value;
value = value > 255 ? 255 : value;
output(x,y) = (uint8) value;
```

Up-conversions ignore the policy and perform this operation:

```
output(x,y) = ((int16) input(x,y)) << shift;
```

The valid values for 'shift' are as specified below, all other values produce undefined behavior.

```
0 <= shift < 8;
```

## Functions

- `vx_node VX_API_CALL vxConvertDepthNode (vx_graph graph, vx_image input, vx_image output, vx_enum policy, vx_scalar shift)`  
[Graph] Creates a bit-depth conversion node.
- `vx_status VX_API_CALL vxuConvertDepth (vx_context context, vx_image input, vx_image output, vx_enum policy, vx_int32 shift)`  
[Immediate] Converts the input images bit-depth into the output image.

### 3.17.2 Function Documentation

**vx\_node VX\_API\_CALL vxConvertDepthNode ( vx\_graph graph, vx\_image input, vx\_image output, vx\_enum policy, vx\_scalar shift )**

[Graph] Creates a bit-depth conversion node.

Parameters

|     |               |                                                                                                            |
|-----|---------------|------------------------------------------------------------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                                                                                |
| in  | <i>input</i>  | The input image.                                                                                           |
| out | <i>output</i> | The output image.                                                                                          |
| in  | <i>policy</i> | A scalar containing a <a href="#">VX_TYPE_ENUM</a> of the <a href="#">vx_convert_policy_e</a> enumeration. |
| in  | <i>shift</i>  | A scalar containing a <a href="#">VX_TYPE_INT32</a> of the shift value.                                    |

Returns

[vx\\_node](#).

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuConvertDepth ( vx\_context *context*, vx\_image *input*, vx\_image *output*, vx\_enum *policy*, vx\_int32 *shift* )**

[Immediate] Converts the input images bit-depth into the output image.

Parameters

|     |                |                                                    |
|-----|----------------|----------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.              |
| in  | <i>input</i>   | The input image.                                   |
| out | <i>output</i>  | The output image.                                  |
| in  | <i>policy</i>  | A <a href="#">vx_convert_policy_e</a> enumeration. |
| in  | <i>shift</i>   | The shift value.                                   |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                               |
| *                 | An error occurred. See <a href="#">vx_status_e</a> .. |

## 3.18 Custom Convolution

### 3.18.1 Detailed Description

Convolves the input with the client supplied convolution matrix.

The client can supply a `vx_int16` typed convolution matrix  $C_{m,n}$ . Outputs will be in the `VX_DF_IMAGE_S16` format unless a `VX_DF_IMAGE_U8` image is explicitly provided. If values would have been out of range of U8 for `VX_DF_IMAGE_U8`, the values are clamped to 0 or 255.

$$k_0 = \frac{m}{2} \quad (3.1)$$

$$l_0 = \frac{n}{2} \quad (3.2)$$

$$sum = \sum_{k=0, l=0}^{k=m-1, l=n-1} input(x + k_0 - k, y + l_0 - l) C_{k,l} \quad (3.3)$$

Note

The above equation for this function is different than an equivalent operation suggested by the OpenCL `V Filter2D` function.

This translates into the C declaration:

```
// A horizontal Scharr gradient operator with different scale.
vx_int16 gx[3][3] = {
 { 3, 0, -3},
 {10, 0, -10},
 { 3, 0, -3},
};
vx_uint32 scale = 8;
vx_convolution scharr_x = vxCreateConvolution(context, 3, 3);
vxReadConvolutionCoefficients(scharr_x, NULL);
vxWriteConvolutionCoefficients(scharr_x, (
 vx_int16*)gx);
vxSetConvolutionAttribute(scharr_x,
 VX_CONVOLUTION_ATTRIBUTE_SCALE, &scale, sizeof(scale));
```

For `VX_DF_IMAGE_U8` output, an additional step is taken:

$$output(x,y) = \begin{cases} 0 & \text{if } sum < 0 \\ 255 & \text{if } sum/scale > 255 \\ sum/scale & \text{otherwise} \end{cases}$$

For `VX_DF_IMAGE_S16` output, the summation is simply set to the output

$$output(x,y) = sum/scale$$

The overflow policy used is `VX_CONVERT_POLICY_SATURATE`.

### Functions

- `vx_node VX_API_CALL vxConvolveNode (vx_graph graph, vx_image input, vx_convolution conv, vx_image output)`  
[Graph] Creates a custom convolution node.
- `vx_status VX_API_CALL vxuConvolve (vx_context context, vx_image input, vx_convolution matrix, vx_image output)`  
[Immediate] Computes a convolution on the input image with the supplied matrix.

### 3.18.2 Function Documentation

`vx_node VX_API_CALL vxConvolveNode ( vx_graph graph, vx_image input, vx_convolution conv, vx_image output )`

[Graph] Creates a custom convolution node.

**Parameters**

|     |               |                                                                                               |
|-----|---------------|-----------------------------------------------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                                                                   |
| in  | <i>input</i>  | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.                                     |
| in  | <i>conv</i>   | The vx_int16 convolution matrix.                                                              |
| out | <i>output</i> | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

**Returns**

[vx\\_node](#).

**Return values**

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** [VX\\_API\\_CALL](#) vxuConvolve ( *vx\_context context*, *vx\_image input*, *vx\_convolution matrix*, *vx\_image output* )

[Immediate] Computes a convolution on the input image with the supplied matrix.

**Parameters**

|     |                |                                                                                               |
|-----|----------------|-----------------------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                                         |
| in  | <i>input</i>   | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.                                     |
| in  | <i>matrix</i>  | The convolution matrix.                                                                       |
| out | <i>output</i>  | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |



## 3.19 Dilate Image

### 3.19.1 Detailed Description

Implements Dilation, which *grows* the white space in a `VX_DF_IMAGE_U8` Boolean image.

This kernel uses a 3x3 box around the output pixel used to determine value.

$$dst(x,y) = \max_{\substack{x-1 \leq x' \leq x+1 \\ y-1 \leq y' \leq y+1}} src(x',y')$$

### Functions

- `vx_node` `VX_API_CALL vxDilate3x3Node` (`vx_graph` *graph*, `vx_image` *input*, `vx_image` *output*)  
[Graph] Creates a Dilation Image Node.
- `vx_status` `VX_API_CALL vxuDilate3x3` (`vx_context` *context*, `vx_image` *input*, `vx_image` *output*)  
[Immediate] Dilates an image by a 3x3 window.

### 3.19.2 Function Documentation

`vx_node` `VX_API_CALL vxDilate3x3Node` ( `vx_graph` *graph*, `vx_image` *input*, `vx_image` *output* )

[Graph] Creates a Dilation Image Node.

Parameters

|     |               |                                                         |
|-----|---------------|---------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                             |
| in  | <i>input</i>  | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| out | <i>output</i> | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

`vx_node`.

Return values

|                |                                                                                                                         |
|----------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuDilate3x3` ( `vx_context` *context*, `vx_image` *input*, `vx_image` *output* )

[Immediate] Dilates an image by a 3x3 window.

Parameters

|     |                |                                                         |
|-----|----------------|---------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                   |
| in  | <i>input</i>   | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| out | <i>output</i>  | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                   |
|-------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                           |
| *                       | An error occurred. See <code>vx_status_e</code> . |

## 3.20 Equalize Histogram

### 3.20.1 Detailed Description

Equalizes the histogram of a grayscale image.

This kernel uses Histogram Equalization to modify the values of a grayscale image so that it will automatically have a standardized brightness and contrast.

### Functions

- **vx\_node** **VX\_API\_CALL vxEqualizeHistNode** (**vx\_graph** graph, **vx\_image** input, **vx\_image** output)  
[Graph] Creates a Histogram Equalization node.
- **vx\_status** **VX\_API\_CALL vxuEqualizeHist** (**vx\_context** context, **vx\_image** input, **vx\_image** output)  
[Immediate] Equalizes the Histogram of a grayscale image.

### 3.20.2 Function Documentation

**vx\_node** **VX\_API\_CALL vxEqualizeHistNode** ( **vx\_graph** graph, **vx\_image** input, **vx\_image** output )

[Graph] Creates a Histogram Equalization node.

Parameters

|     |        |                                                                                                           |
|-----|--------|-----------------------------------------------------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                                                               |
| in  | input  | The grayscale input image in <a href="#">VX_DF_IMAGE_U8</a> .                                             |
| out | output | The grayscale output image of type <a href="#">VX_DF_IMAGE_U8</a> with equalized brightness and contrast. |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL vxuEqualizeHist** ( **vx\_context** context, **vx\_image** input, **vx\_image** output )

[Immediate] Equalizes the Histogram of a grayscale image.

Parameters

|     |         |                                                                                                           |
|-----|---------|-----------------------------------------------------------------------------------------------------------|
| in  | context | The reference to the overall context.                                                                     |
| in  | input   | The grayscale input image in <a href="#">VX_DF_IMAGE_U8</a>                                               |
| out | output  | The grayscale output image of type <a href="#">VX_DF_IMAGE_U8</a> with equalized brightness and contrast. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.21 Erode Image

### 3.21.1 Detailed Description

Implements Erosion, which *shrinks* the white space in a [VX\\_DF\\_IMAGE\\_U8](#) Boolean image.

This kernel uses a 3x3 box around the output pixel used to determine value.

$$dst(x,y) = \min_{\substack{x-1 \leq x' \leq x+1 \\ y-1 \leq y' \leq y+1}} src(x',y')$$

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxErode3x3Node](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output)  
[Graph] Creates an Erosion Image Node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuErode3x3](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output)  
[Immediate] Erodes an image by a 3x3 window.

### 3.21.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxErode3x3Node](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output )**

[Graph] Creates an Erosion Image Node.

Parameters

|     |        |                                                            |
|-----|--------|------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                |
| in  | input  | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.  |
| out | output | The output image in <a href="#">VX_DF_IMAGE_U8</a> format. |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuErode3x3](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output )**

[Immediate] Erodes an image by a 3x3 window.

Parameters

|     |         |                                                            |
|-----|---------|------------------------------------------------------------|
| in  | context | The reference to the overall context.                      |
| in  | input   | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.  |
| out | output  | The output image in <a href="#">VX_DF_IMAGE_U8</a> format. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.22 Fast Corners

### 3.22.1 Detailed Description

Computes the corners in an image using a method based upon FAST9 algorithm suggested in [3] and with some updates from [4] with modifications described below.

It extracts corners by evaluating pixels on the Bresenham circle around a candidate point. If  $N$  contiguous pixels are brighter than the candidate point by at least a threshold value  $t$  or darker by at least  $t$ , then the candidate point is considered to be a corner. For each detected corner, its strength is computed. Optionally, a non-maxima suppression step is applied on all detected corners to remove multiple or spurious responses.

### 3.22.2 Segment Test Detector

The FAST corner detector uses the pixels on a Bresenham circle of radius 3 (16 pixels) to classify whether a candidate point  $p$  is actually a corner, given the following variables.

$I$  = input image (3.4)

$p$  = candidate point position for a corner (3.5)

$I_p$  = image intensity of the candidate point in image  $I$  (3.6)

$x$  = pixel on the Bresenham circle around the candidate point  $p$  (3.7)

$I_x$  = image intensity of the candidate point (3.8)

$t$  = intensity difference threshold for a corner (3.9)

$N$  = minimum number of contiguous pixel to detect a corner (3.10)

$S$  = set of contiguous pixel on the Bresenham circle around the candidate point (3.11)

$C_p$  = corner response at corner location  $p$  (3.12)

(3.13)

The two conditions for FAST corner detection can be expressed as:

- C1: A set of  $N$  contiguous pixels  $S$ ,  $\forall x \text{ in } S, I_x > I_p + t$
- C2: A set of  $N$  contiguous pixels  $S$ ,  $\forall x \text{ in } S, I_x < I_p - t$

So when either of these two conditions is met, the candidate  $p$  is classified as a corner.

In this version of the FAST algorithm, the minimum number of contiguous pixels  $N$  is 9 (FAST9).

The value of the intensity difference threshold *strength\_thresh.* of type `VX_TYPE_FLOAT32` must be within:

$$UINT8_{MIN} < t < UINT8_{MAX}$$

These limits are established due to the input data type `VX_DF_IMAGE_U8`.

**Corner Strength Computation** Once a corner has been detected, its strength (response, saliency, or score) shall be computed if `nonmax_suppression` is set to true, otherwise the value of strength is undefined. The corner response  $C_p$  function is defined as the largest threshold  $t$  for which the pixel  $p$  remains a corner.

**Non-maximum suppression** If the `nonmax_suppression` flag is true, a non-maxima suppression step is applied on the detected corners. The corner with coordinates  $(x, y)$  is kept if and only if

$$\begin{aligned} C_p(x, y) &\geq C_p(x-1, y-1) \text{ and } C_p(x, y) \geq C_p(x, y-1) \text{ and} \\ C_p(x, y) &\geq C_p(x+1, y-1) \text{ and } C_p(x, y) \geq C_p(x-1, y) \text{ and} \\ C_p(x, y) &> C_p(x+1, y) \text{ and } C_p(x, y) > C_p(x-1, y+1) \text{ and} \\ C_p(x, y) &> C_p(x, y+1) \text{ and } C_p(x, y) > C_p(x+1, y+1) \end{aligned}$$

See also

<http://www.edwardrosten.com/work/fast.html>

[http://en.wikipedia.org/wiki/Features\\_from\\_accelerated\\_segment\\_test](http://en.wikipedia.org/wiki/Features_from_accelerated_segment_test)

## Functions

- **vx\_node** **VX\_API\_CALL vxFastCornersNode** ( **vx\_graph** *graph*, **vx\_image** *input*, **vx\_scalar** *strength\_thresh*, **vx\_bool** *nonmax\_suppression*, **vx\_array** *corners*, **vx\_scalar** *num\_corners* )  
[Graph] Creates a FAST Corners Node.
- **vx\_status** **VX\_API\_CALL vxuFastCorners** ( **vx\_context** *context*, **vx\_image** *input*, **vx\_scalar** *strength\_thresh*, **vx\_bool** *nonmax\_suppression*, **vx\_array** *corners*, **vx\_scalar** *num\_corners* )  
[Immediate] Computes corners on an image using FAST algorithm and produces the array of feature points.

### 3.22.3 Function Documentation

**vx\_node** **VX\_API\_CALL vxFastCornersNode** ( **vx\_graph** *graph*, **vx\_image** *input*, **vx\_scalar** *strength\_thresh*, **vx\_bool** *nonmax\_suppression*, **vx\_array** *corners*, **vx\_scalar** *num\_corners* )

[Graph] Creates a FAST Corners Node.

Parameters

|     |                                 |                                                                                                                                                |
|-----|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>graph</i>                    | The reference to the graph.                                                                                                                    |
| in  | <i>input</i>                    | The input <b>VX_DF_IMAGE_U8</b> image.                                                                                                         |
| in  | <i>strength_thresh</i>          | Threshold on difference between intensity of the central pixel and pixels on Bresenham's circle of radius 3 ( <b>VX_TYPE_FLOAT32</b> scalar).  |
| in  | <i>nonmax_↔<br/>suppression</i> | If true, non-maximum suppression is applied to detected corners before being placed in the <b>vx_array</b> of <b>VX_TYPE_KEYPOINT</b> objects. |
| out | <i>corners</i>                  | Output corner <b>vx_array</b> of <b>VX_TYPE_KEYPOINT</b> .                                                                                     |
| out | <i>num_corners</i>              | The total number of detected corners in image (optional). Use a <b>VX_TYPE_↔<br/>SIZE</b> scalar.                                              |

Returns

**vx\_node**.

Return values

|                |                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <b>vxGetStatus</b> |
|----------------|-------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL vxuFastCorners** ( **vx\_context** *context*, **vx\_image** *input*, **vx\_scalar** *strength\_thresh*, **vx\_bool** *nonmax\_suppression*, **vx\_array** *corners*, **vx\_scalar** *num\_corners* )

[Immediate] Computes corners on an image using FAST algorithm and produces the array of feature points.

Parameters

|     |                                 |                                                                                                                                                |
|-----|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i>                  | The reference to the overall context.                                                                                                          |
| in  | <i>input</i>                    | The input <b>VX_DF_IMAGE_U8</b> image.                                                                                                         |
| in  | <i>strength_thresh</i>          | Threshold on difference between intensity of the central pixel and pixels on Bresenham's circle of radius 3 ( <b>VX_TYPE_FLOAT32</b> scalar)   |
| in  | <i>nonmax_↔<br/>suppression</i> | If true, non-maximum suppression is applied to detected corners before being places in the <b>vx_array</b> of <b>VX_TYPE_KEYPOINT</b> structs. |
| out | <i>corners</i>                  | Output corner <b>vx_array</b> of <b>VX_TYPE_KEYPOINT</b> .                                                                                     |
| out | <i>num_corners</i>              | The total number of detected corners in image (optional). Use a <b>VX_TYPE_↔<br/>SIZE</b> scalar.                                              |

Returns

A **vx\_status\_e** enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.23 Gaussian Filter

### 3.23.1 Detailed Description

Computes a Gaussian filter over a window of the input image.

This filter uses the following convolution matrix:

$$\mathbf{K}_{\text{gaussian}} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \frac{1}{16}$$

### Functions

- **vx\_node** **VX\_API\_CALL vxGaussian3x3Node** (**vx\_graph** graph, **vx\_image** input, **vx\_image** output)  
[Graph] Creates a Gaussian Filter Node.
- **vx\_status** **VX\_API\_CALL vxuGaussian3x3** (**vx\_context** context, **vx\_image** input, **vx\_image** output)  
[Immediate] Computes a gaussian filter on the image by a 3x3 window.

### 3.23.2 Function Documentation

**vx\_node** **VX\_API\_CALL vxGaussian3x3Node** ( **vx\_graph** graph, **vx\_image** input, **vx\_image** output )

[Graph] Creates a Gaussian Filter Node.

Parameters

|     |        |                                                            |
|-----|--------|------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                |
| in  | input  | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.  |
| out | output | The output image in <a href="#">VX_DF_IMAGE_U8</a> format. |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL vxuGaussian3x3** ( **vx\_context** context, **vx\_image** input, **vx\_image** output )

[Immediate] Computes a gaussian filter on the image by a 3x3 window.

Parameters

|     |         |                                                            |
|-----|---------|------------------------------------------------------------|
| in  | context | The reference to the overall context.                      |
| in  | input   | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.  |
| out | output  | The output image in <a href="#">VX_DF_IMAGE_U8</a> format. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.24 Harris Corners

### 3.24.1 Detailed Description

Computes the Harris Corners of an image.

The Harris Corners are computed with several parameters

$$I = \text{input image} \quad (3.14)$$

$$T_c = \text{corner strength threshold} \quad (3.15)$$

$$r = \text{euclidean radius} \quad (3.16)$$

$$k = \text{sensitivity threshold} \quad (3.17)$$

$$w = \text{window size} \quad (3.18)$$

$$b = \text{block size} \quad (3.19)$$

$$(3.20)$$

The computation to find the corner values or scores can be summarized as:

$$G_x = \text{Sobel}_x(w, I) \quad (3.21)$$

$$G_y = \text{Sobel}_y(w, I) \quad (3.22)$$

$$A = \text{window}_{G_{x,y}}(x - b/2, y - b/2, x + b/2, y + b/2) \quad (3.23)$$

$$\text{trace}(A) = \sum_x^A G_x^2 + \sum_y^A G_y^2 \quad (3.24)$$

$$\det(A) = \sum_x^A G_x^2 \sum_y^A G_y^2 - \left( \sum_x^A (G_x G_y) \right)^2 \quad (3.25)$$

$$M_c(x, y) = \det(A) - k * \text{trace}(A)^2 \quad (3.26)$$

$$V_c(x, y) = \begin{cases} M_c(x, y) & \text{if } M_c(x, y) > T_c \\ 0 & \text{otherwise} \end{cases} \quad (3.27)$$

where  $V_c$  is the thresholded corner value.

The normalized Sobel kernels used for the gradient computation shall be as shown below:

- For gradient size 3:

$$\text{Sobel}_x(\text{Normalized}) = \frac{1}{4 * 255 * b} * \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$$

$$\text{Sobel}_y(\text{Normalized}) = \frac{1}{4 * 255 * b} * \text{transpose}(\text{sobel}_x) = \frac{1}{4 * 255 * b} * \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$$

- For gradient size 5:

$$\text{Sobel}_x(\text{Normalized}) = \frac{1}{16 * 255 * b} * \begin{vmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{vmatrix}$$

$$\text{Sobel}_y(\text{Normalized}) = \frac{1}{16 * 255 * b} * \text{transpose}(\text{sobel}_x)$$

- For gradient size 7:

$$\text{Sobel}_x(\text{Normalized}) = \frac{1}{64 * 255 * b} * \begin{vmatrix} -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -20 & -80 & -100 & 0 & 100 & 80 & 20 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \end{vmatrix}$$



$$\text{Sobel}_y(\text{Normalized}) = \frac{1}{64 * 255 * b} * \text{transpose}(\text{sobel}_x)$$

$V_c$  is then non-maximally suppressed using the following algorithm:

- Filter the features using the non-maximum suppression algorithm defined for `vxFastCornersNode`.
- Create an array of features sorted by  $V_c$  in descending order:  $V_c(j) > V_c(j+1)$ .
- Initialize an empty feature set  $F = \{\}$
- For each feature  $j$  in the sorted array, while  $V_c(j) > T_c$ :
  - If there is no feature  $i$  in  $F$  such that the Euclidean distance between pixels  $i$  and  $j$  is less than  $r$ , add the feature  $j$  to the feature set  $F$ .

An implementation shall support all values of Euclidean distance  $r$  that satisfy:

```
0 <= max_dist <= 30
```

The feature set  $F$  is returned as a `vx_array` of `vx_keypoint_t` structs.

## Functions

- `vx_node VX_API_CALL vxHarrisCornersNode (vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners)`  
*[Graph] Creates a Harris Corners Node.*
- `vx_status VX_API_CALL vxuHarrisCorners (vx_context context, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners)`

*[Immediate] Computes the Harris Corners over an image and produces the array of scored points.*

### 3.24.2 Function Documentation

**`vx_node VX_API_CALL vxHarrisCornersNode ( vx_graph graph, vx_image input, vx_scalar strength_thresh, vx_scalar min_distance, vx_scalar sensitivity, vx_int32 gradient_size, vx_int32 block_size, vx_array corners, vx_scalar num_corners )`**

*[Graph] Creates a Harris Corners Node.*

Parameters

|     |                        |                                                                                                                                               |
|-----|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>graph</i>           | The reference to the graph.                                                                                                                   |
| in  | <i>input</i>           | The input <code>VX_DF_IMAGE_U8</code> image.                                                                                                  |
| in  | <i>strength_thresh</i> | The <code>VX_TYPE_FLOAT32</code> minimum threshold with which to eliminate Harris Corner scores (computed using the normalized Sobel kernel). |
| in  | <i>min_distance</i>    | The <code>VX_TYPE_FLOAT32</code> radial Euclidean distance for non-maximum suppression.                                                       |
| in  | <i>sensitivity</i>     | The <code>VX_TYPE_FLOAT32</code> scalar sensitivity threshold $k$ from the Harris-Stephens equation.                                          |
| in  | <i>gradient_size</i>   | The gradient window size to use on the input. The implementation must support at least 3, 5, and 7.                                           |
| in  | <i>block_size</i>      | The block window size used to compute the Harris Corner score. The implementation must support at least 3, 5, and 7.                          |
| out | <i>corners</i>         | The array of <code>VX_TYPE_KEYPOINT</code> objects.                                                                                           |
| out | <i>num_corners</i>     | The total number of detected corners in image (optional). Use a <code>VX_TYPE_SIZE</code> scalar.                                             |

Returns

`vx_node`.

## Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuHarrisCorners ( vx\_context context, vx\_image input, vx\_scalar strength\_thresh, vx\_scalar min\_distance, vx\_scalar sensitivity, vx\_int32 gradient\_size, vx\_int32 block\_size, vx\_array corners, vx\_scalar num\_corners )**

[Immediate] Computes the Harris Corners over an image and produces the array of scored points.

## Parameters

|     |                        |                                                                                                                                             |
|-----|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i>         | The reference to the overall context.                                                                                                       |
| in  | <i>input</i>           | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                             |
| in  | <i>strength_thresh</i> | The <a href="#">VX_TYPE_FLOAT32</a> minimum threshold which to eliminate Harris Corner scores (computed using the normalized Sobel kernel). |
| in  | <i>min_distance</i>    | The <a href="#">VX_TYPE_FLOAT32</a> radial Euclidean distance for non-maximum suppression.                                                  |
| in  | <i>sensitivity</i>     | The <a href="#">VX_TYPE_FLOAT32</a> scalar sensitivity threshold $k$ from the Harris-Stephens equation.                                     |
| in  | <i>gradient_size</i>   | The gradient window size to use on the input. The implementation must support at least 3, 5, and 7.                                         |
| in  | <i>block_size</i>      | The block window size used to compute the harris corner score. The implementation must support at least 3, 5, and 7.                        |
| out | <i>corners</i>         | The array of <a href="#">VX_TYPE_KEYPOINT</a> structs.                                                                                      |
| out | <i>num_corners</i>     | The total number of detected corners in image (optional). Use a <a href="#">VX_TYPE_SIZE</a> scalar                                         |

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.25 Histogram

### 3.25.1 Detailed Description

Generates a distribution from an image.

This kernel counts the number of occurrences of each pixel value within the window size of a pre-calculated number of bins.

### Functions

- `vx_node` `VX_API_CALL vxHistogramNode` (`vx_graph` *graph*, `vx_image` *input*, `vx_distribution` *distribution*)  
[Graph] Creates a Histogram node.
- `vx_status` `VX_API_CALL vxuHistogram` (`vx_context` *context*, `vx_image` *input*, `vx_distribution` *distribution*)  
[Immediate] Generates a distribution from an image.

### 3.25.2 Function Documentation

`vx_node` `VX_API_CALL vxHistogramNode` ( `vx_graph` *graph*, `vx_image` *input*, `vx_distribution` *distribution* )

[Graph] Creates a Histogram node.

Parameters

|     |                     |                                                  |
|-----|---------------------|--------------------------------------------------|
| in  | <i>graph</i>        | The reference to the graph.                      |
| in  | <i>input</i>        | The input image in <code>VX_DF_IMAGE_U8</code> . |
| out | <i>distribution</i> | The output distribution.                         |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuHistogram` ( `vx_context` *context*, `vx_image` *input*, `vx_distribution` *distribution* )

[Immediate] Generates a distribution from an image.

Parameters

|     |                     |                                                |
|-----|---------------------|------------------------------------------------|
| in  | <i>context</i>      | The reference to the overall context.          |
| in  | <i>input</i>        | The input image in <code>VX_DF_IMAGE_U8</code> |
| out | <i>distribution</i> | The output distribution.                       |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                   |
|-------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                           |
| *                       | An error occurred. See <code>vx_status_e</code> . |

## 3.26 Gaussian Image Pyramid

### 3.26.1 Detailed Description

Computes a Gaussian Image Pyramid from an input image.

This vision function creates the Gaussian image pyramid from the input image using the particular 5x5 Gaussian Kernel:

$$\mathbf{G} = \frac{1}{256} * \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

on each level of the pyramid then scales the image to the next level using [VX\\_INTERPOLATION\\_TYPE\\_NEAREST\\_NEIGHBOR](#). Level 0 shall always have the same resolution as the input image. For the Gaussian pyramid, level 0 shall be the same as the input image. The pyramids must be configured with one of the following level scaling:

- [VX\\_SCALE\\_PYRAMID\\_HALF](#)
- [VX\\_SCALE\\_PYRAMID\\_ORB](#)

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL](#) [vxGaussianPyramidNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_pyramid](#) gaussian)  
[Graph] Creates a node for a Gaussian Image Pyramid.
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxuGaussianPyramid](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_pyramid](#) gaussian)  
[Immediate] Computes a Gaussian pyramid from an input image.

### 3.26.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL](#) [vxGaussianPyramidNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_pyramid](#) gaussian )**

[Graph] Creates a node for a Gaussian Image Pyramid.

Parameters

|                     |                          |                                                                        |
|---------------------|--------------------------|------------------------------------------------------------------------|
| <a href="#">in</a>  | <a href="#">graph</a>    | The reference to the graph.                                            |
| <a href="#">in</a>  | <a href="#">input</a>    | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.              |
| <a href="#">out</a> | <a href="#">gaussian</a> | The Gaussian pyramid with <a href="#">VX_DF_IMAGE_U8</a> to construct. |

See also

[Object: Pyramid](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL](#) [vxuGaussianPyramid](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_pyramid](#) gaussian )**

[Immediate] Computes a Gaussian pyramid from an input image.

**Parameters**

|     |                 |                                                                        |
|-----|-----------------|------------------------------------------------------------------------|
| in  | <i>context</i>  | The reference to the overall context.                                  |
| in  | <i>input</i>    | The input image in <a href="#">VX_DF_IMAGE_U8</a>                      |
| out | <i>gaussian</i> | The Gaussian pyramid with <a href="#">VX_DF_IMAGE_U8</a> to construct. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.27 Integral Image

### 3.27.1 Detailed Description

Computes the integral image of the input.

Each output pixel is the sum of the corresponding input pixel and all other pixels above and to the left of it.

$$dst(x,y) = sum(x,y)$$

where, for  $x \geq 0$  and  $y \geq 0$

$$sum(x,y) = src(x,y) + sum(x-1,y) + sum(x,y-1) - sum(x-1,y-1)$$

otherwise,

$$sum(x,y) = 0$$

The overflow policy used is [VX\\_CONVERT\\_POLICY\\_WRAP](#).

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxIntegrallImageNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output)  
[Graph] Creates an Integral Image Node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuIntegrallImage](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output)  
[Immediate] Computes the integral image of the input.

### 3.27.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxIntegrallImageNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_image](#) output )**

[Graph] Creates an Integral Image Node.

Parameters

|     |               |                                                             |
|-----|---------------|-------------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                                 |
| in  | <i>input</i>  | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.   |
| out | <i>output</i> | The output image in <a href="#">VX_DF_IMAGE_U32</a> format. |

Returns

[vx\\_node](#).

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuIntegrallImage](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output )**

[Immediate] Computes the integral image of the input.

Parameters

|     |                |                                                             |
|-----|----------------|-------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                       |
| in  | <i>input</i>   | The input image in <a href="#">VX_DF_IMAGE_U8</a> format.   |
| out | <i>output</i>  | The output image in <a href="#">VX_DF_IMAGE_U32</a> format. |

Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                              |
| *                       | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.28 Magnitude

### 3.28.1 Detailed Description

Implements the Gradient Magnitude Computation Kernel.

This kernel takes two gradients in [VX\\_DF\\_IMAGE\\_S16](#) format and computes the [VX\\_DF\\_IMAGE\\_S16](#) normalized magnitude. Magnitude is computed as:

$$mag(x,y) = \sqrt{grad_x(x,y)^2 + grad_y(x,y)^2}$$

The conceptual definition describing the overflow is given as:

```
uint16 z = uint16(sqrt(double(uint32(int32(x) * int32(x)) + uint32(int32(y) * int32(y)))) + 0.5);
int16 mag = z > 32767 ? 32767 : z;
```

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL](#) [vxMagnitudeNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) mag)  
[Graph] Create a Magnitude node.
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxuMagnitude](#) ([vx\\_context](#) context, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) output)  
[Immediate] Invokes an immediate Magnitude.

### 3.28.2 Function Documentation

[vx\\_node](#) [VX\\_API\\_CALL](#) [vxMagnitudeNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) mag )

[Graph] Create a Magnitude node.

Parameters

|     |        |                                                                            |
|-----|--------|----------------------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                                |
| in  | grad_x | The input x image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| in  | grad_y | The input y image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| out | mag    | The magnitude image. This is in <a href="#">VX_DF_IMAGE_S16</a> format.    |

See also

[VX\\_KERNEL\\_MAGNITUDE](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

[vx\\_status](#) [VX\\_API\\_CALL](#) [vxuMagnitude](#) ( [vx\\_context](#) context, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) output )

[Immediate] Invokes an immediate Magnitude.

Parameters



|     |                |                                                                              |
|-----|----------------|------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                        |
| in  | <i>grad_x</i>  | The input x image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format.   |
| in  | <i>grad_y</i>  | The input y image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format.   |
| out | <i>output</i>  | The magnitude image. This will be in <a href="#">VX_DF_IMAGE_S16</a> format. |

#### Returns

A [vx\\_status\\_e](#) enumeration.

#### Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.29 Mean and Standard Deviation

### 3.29.1 Detailed Description

Computes the mean pixel value and the standard deviation of the pixels in the input image (which has a dimension width and height).

The mean value is computed as:

$$\mu = \frac{\left( \sum_{y=0}^h \sum_{x=0}^w src(x,y) \right)}{(width * height)}$$

The standard deviation is computed as:

$$\sigma = \sqrt{\frac{\left( \sum_{y=0}^h \sum_{x=0}^w (\mu - src(x,y))^2 \right)}{(width * height)}}$$

### Functions

- **vx\_node** VX\_API\_CALL **vxMeanStdDevNode** (**vx\_graph** graph, **vx\_image** input, **vx\_scalar** mean, **vx\_scalar** stddev)  
[Graph] Creates a mean value and standard deviation node.
- **vx\_status** VX\_API\_CALL **vxuMeanStdDev** (**vx\_context** context, **vx\_image** input, **vx\_float32** \*mean, **vx\_float32** \*stddev)  
[Immediate] Computes the mean value and standard deviation.

### 3.29.2 Function Documentation

**vx\_node** VX\_API\_CALL **vxMeanStdDevNode** ( **vx\_graph** graph, **vx\_image** input, **vx\_scalar** mean, **vx\_scalar** stddev )

[Graph] Creates a mean value and standard deviation node.

Parameters

|     |        |                                                                             |
|-----|--------|-----------------------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                                 |
| in  | input  | The input image. <a href="#">VX_DF_IMAGE_U8</a> is supported.               |
| out | mean   | The <a href="#">VX_TYPE_FLOAT32</a> average pixel value.                    |
| out | stddev | The <a href="#">VX_TYPE_FLOAT32</a> standard deviation of the pixel values. |

Returns

**vx\_node**.

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>vx_node</b> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** VX\_API\_CALL **vxuMeanStdDev** ( **vx\_context** context, **vx\_image** input, **vx\_float32** \* mean, **vx\_float32** \* stddev )

[Immediate] Computes the mean value and standard deviation.

Parameters

|    |         |                                                               |
|----|---------|---------------------------------------------------------------|
| in | context | The reference to the overall context.                         |
| in | input   | The input image. <a href="#">VX_DF_IMAGE_U8</a> is supported. |

|     |               |                                             |
|-----|---------------|---------------------------------------------|
| out | <i>mean</i>   | The average pixel value.                    |
| out | <i>stddev</i> | The standard deviation of the pixel values. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.30 Median Filter

### 3.30.1 Detailed Description

Computes a median pixel value over a window of the input image.

The median is the middle value over an odd-numbered, sorted range of values.

### Functions

- `vx_node` `VX_API_CALL vxMedian3x3Node` (`vx_graph` *graph*, `vx_image` *input*, `vx_image` *output*)  
[Graph] Creates a Median Image Node.
- `vx_status` `VX_API_CALL vxuMedian3x3` (`vx_context` *context*, `vx_image` *input*, `vx_image` *output*)  
[Immediate] Computes a median filter on the image by a 3x3 window.

### 3.30.2 Function Documentation

`vx_node` `VX_API_CALL vxMedian3x3Node` ( `vx_graph` *graph*, `vx_image` *input*, `vx_image` *output* )

[Graph] Creates a Median Image Node.

Parameters

|                  |                     |                                                         |
|------------------|---------------------|---------------------------------------------------------|
| <code>in</code>  | <code>graph</code>  | The reference to the graph.                             |
| <code>in</code>  | <code>input</code>  | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| <code>out</code> | <code>output</code> | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

`vx_status` `VX_API_CALL vxuMedian3x3` ( `vx_context` *context*, `vx_image` *input*, `vx_image` *output* )

[Immediate] Computes a median filter on the image by a 3x3 window.

Parameters

|                  |                      |                                                         |
|------------------|----------------------|---------------------------------------------------------|
| <code>in</code>  | <code>context</code> | The reference to the overall context.                   |
| <code>in</code>  | <code>input</code>   | The input image in <code>VX_DF_IMAGE_U8</code> format.  |
| <code>out</code> | <code>output</code>  | The output image in <code>VX_DF_IMAGE_U8</code> format. |

Returns

A `vx_status_e` enumeration.

Return values

|                         |                                                   |
|-------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code> | Success                                           |
| *                       | An error occurred. See <code>vx_status_e</code> . |

## 3.31 Min, Max Location

### 3.31.1 Detailed Description

Finds the minimum and maximum values in an image and a location for each.

If the input image has several minimums/maximums, the kernel returns all of them.

$$\begin{aligned} \minVal = & \min_{\substack{0 \leq x' \leq width \\ 0 \leq y' \leq height}} src(x', y') \end{aligned}$$

$$\begin{aligned} \maxVal = & \max_{\substack{0 \leq x' \leq width \\ 0 \leq y' \leq height}} src(x', y') \end{aligned}$$

### Functions

- **vx\_node** **VX\_API\_CALL vxMinMaxLocNode** (**vx\_graph** graph, **vx\_image** input, **vx\_scalar** minVal, **vx\_scalar** maxVal, **vx\_array** minLoc, **vx\_array** maxLoc, **vx\_scalar** minCount, **vx\_scalar** maxCount)

[Graph] Creates a min,max,loc node.

- **vx\_status** **VX\_API\_CALL vxuMinMaxLoc** (**vx\_context** context, **vx\_image** input, **vx\_scalar** minVal, **vx\_scalar** maxVal, **vx\_array** minLoc, **vx\_array** maxLoc, **vx\_scalar** minCount, **vx\_scalar** maxCount)

[Immediate] Computes the minimum and maximum values of the image.

### 3.31.2 Function Documentation

**vx\_node** **VX\_API\_CALL vxMinMaxLocNode** ( **vx\_graph** graph, **vx\_image** input, **vx\_scalar** minVal, **vx\_scalar** maxVal, **vx\_array** minLoc, **vx\_array** maxLoc, **vx\_scalar** minCount, **vx\_scalar** maxCount )

[Graph] Creates a min,max,loc node.

Parameters

|     |          |                                                                                                                                                                          |
|-----|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in  | graph    | The reference to create the graph.                                                                                                                                       |
| in  | input    | The input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.                                                                             |
| out | minVal   | The minimum value in the image, which corresponds to the type of the input.                                                                                              |
| out | maxVal   | The maximum value in the image, which corresponds to the type of the input.                                                                                              |
| out | minLoc   | The minimum <a href="#">VX_TYPE_COORDINATES2D</a> locations (optional). If the input image has several minimums, the kernel will return up to the capacity of the array. |
| out | maxLoc   | The maximum <a href="#">VX_TYPE_COORDINATES2D</a> locations (optional). If the input image has several maximums, the kernel will return up to the capacity of the array. |
| out | minCount | The total number of detected minimums in image (optional). Use a <a href="#">VX_TY↔PE_UINT32</a> scalar.                                                                 |
| out | maxCount | The total number of detected maximums in image (optional). Use a <a href="#">VX_TY↔PE_UINT32</a> scalar.                                                                 |

Returns

**vx\_node**.

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>vx_node</b> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL vxuMinMaxLoc** ( **vx\_context** context, **vx\_image** input, **vx\_scalar** minVal, **vx\_scalar** maxVal, **vx\_array** minLoc, **vx\_array** maxLoc, **vx\_scalar** minCount, **vx\_scalar** maxCount )

[Immediate] Computes the minimum and maximum values of the image.

**Parameters**

|     |                 |                                                                                                                 |
|-----|-----------------|-----------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i>  | The reference to the overall context.                                                                           |
| in  | <i>input</i>    | The input image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format.                    |
| out | <i>minVal</i>   | The minimum value in the image.                                                                                 |
| out | <i>maxVal</i>   | The maximum value in the image.                                                                                 |
| out | <i>minLoc</i>   | The minimum locations (optional). If the input image has several minimums, the kernel will return all of them). |
| out | <i>maxLoc</i>   | The maximum locations (optional). If the input image has several maximums, the kernel will return all of them). |
| out | <i>minCount</i> | The total number of detected minimums in image (optional).                                                      |
| out | <i>maxCount</i> | The total number of detected maximums in image (optional).                                                      |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.32 Optical Flow Pyramid (LK)

### 3.32.1 Detailed Description

Computes the optical flow using the Lucas-Kanade method between two pyramid images.

The function is an implementation of the algorithm described in [1]. The function inputs are two `vx_pyramid` objects, old and new, along with a `vx_array` of `vx_keypoint_t` structs to track from the old `vx_pyramid`. The function outputs a `vx_array` of `vx_keypoint_t` structs that were tracked from the old `vx_pyramid` to the new `vx_pyramid`. Each element in the `vx_array` of `vx_keypoint_t` structs in the new array may be valid or not. The implementation shall return the same number of `vx_keypoint_t` structs in the new `vx_array` that were in the older `vx_array`.

In more detail: The Lucas-Kanade method finds the affine motion vector  $V$  for each point in the old image tracking points array, using the following equation:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x^2 & \sum_i I_x * I_y \\ \sum_i I_x * I_y & \sum_i I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x * I_t \\ -\sum_i I_y * I_t \end{bmatrix}$$

Where  $I_x$  and  $I_y$  are obtained using the Scharr gradients on the input image:

$$G_x = \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

$I_t$  is obtained by a simple difference between the same pixel in both images.  $i$  is defined as the adjacent pixels to the point  $p(x,y)$  under consideration. With a given window size of  $M$ ,  $i$  is  $M^2$  points. The pixel  $p(x,y)$  is centered in the window. In practice, to get an accurate solution, it is necessary to iterate multiple times on this scheme (in a Newton-Raphson fashion) until:

- the residual of the affine motion vector is smaller than a threshold
- And/or maximum number of iteration achieved. Each iteration, the estimation of the previous iteration is used by changing  $I_t$  to be the difference between the old image and the pixel with the estimated coordinates in the new image. Each iteration the function checks if the pixel to track was lost. The criteria for lost tracking is that the matrix above is invertible. (The determinant of the matrix is less than a threshold :  $10^{-7}$  .) Or the minimum eigenvalue of the matrix is smaller then a threshold (  $10^{-4}$  ). Also lost tracking happens when the point tracked coordinate is outside the image coordinates. When `vx_true_e` is given as the input to `use_initial_estimates`, the algorithm starts by calculating  $I_t$  as the difference between the old image and the pixel with the initial estimated coordinates in the new image. The input `vx_array` of `vx_keypoint_t` structs with `tracking_status` set to zero (lost) are copied to the new `vx_array`.

Clients are responsible for editing the output `vx_array` of `vx_keypoint_t` structs array before applying it as the input `vx_array` of `vx_keypoint_t` structs for the next frame. For example, `vx_keypoint_t` structs with `tracking_status` set to zero may be removed by a client for efficiency.

This function changes just the  $x$ ,  $y$ , and `tracking_status` members of the `vx_keypoint_t` structure and behaves as if it copied the rest from the old tracking `vx_keypoint_t` to new image `vx_keypoint_t`.

## Functions

- `vx_node` VX\_API\_CALL `vxOpticalFlowPyrLKNode` (`vx_graph` graph, `vx_pyramid` old\_images, `vx_pyramid` new\_images, `vx_array` old\_points, `vx_array` new\_points\_estimates, `vx_array` new\_points, `vx_enum` termination, `vx_scalar` epsilon, `vx_scalar` num\_iterations, `vx_scalar` use\_initial\_estimate, `vx_size` window\_dimension)

[Graph] Creates a Lucas Kanade Tracking Node.

- `vx_status` VX\_API\_CALL `vxuOpticalFlowPyrLK` (`vx_context` context, `vx_pyramid` old\_images, `vx_pyramid` new\_images, `vx_array` old\_points, `vx_array` new\_points\_estimates, `vx_array` new\_points, `vx_enum` termination, `vx_scalar` epsilon, `vx_scalar` num\_iterations, `vx_scalar` use\_initial\_estimate, `vx_size` window\_dimension)

[Immediate] Computes an optical flow on two images.

### 3.32.2 Function Documentation

**`vx_node VX_API_CALL vxOpticalFlowPyrLKNode ( vx_graph graph, vx_pyramid old_images, vx_pyramid new_images, vx_array old_points, vx_array new_points_estimates, vx_array new_points, vx_enum termination, vx_scalar epsilon, vx_scalar num_iterations, vx_scalar use_initial_estimate, vx_size window_dimension )`**

[Graph] Creates a Lucas Kanade Tracking Node.



**Parameters**

|     |                                         |                                                                                                                                                                                                           |
|-----|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>graph</i>                            | The reference to the graph.                                                                                                                                                                               |
| in  | <i>old_images</i>                       | Input of first (old) image pyramid in <a href="#">VX_DF_IMAGE_U8</a> .                                                                                                                                    |
| in  | <i>new_images</i>                       | Input of destination (new) image pyramid <a href="#">VX_DF_IMAGE_U8</a> .                                                                                                                                 |
| in  | <i>old_points</i>                       | An array of key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> ; those key points are defined at the <i>old_images</i> high resolution pyramid.                                 |
| in  | <i>new_points</i> ↔<br><i>estimates</i> | An array of estimation on what is the output key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> ; those keypoints are defined at the <i>new_images</i> high resolution pyramid. |
| out | <i>new_points</i>                       | An output array of key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> ; those key points are defined at the <i>new_images</i> high resolution pyramid.                          |
| in  | <i>termination</i>                      | The termination can be <a href="#">VX_TERM_CRITERIA_ITERATIONS</a> or <a href="#">VX_TERM_CRITERIA_EPSILON</a> or <a href="#">VX_TERM_CRITERIA_BOTH</a> .                                                 |
| in  | <i>epsilon</i>                          | The <a href="#">vx_float32</a> error for terminating the algorithm.                                                                                                                                       |
| in  | <i>num_iterations</i>                   | The number of iterations. Use a <a href="#">VX_TYPE_UINT32</a> scalar.                                                                                                                                    |
| in  | <i>use_initial</i> ↔<br><i>estimate</i> | Use a <a href="#">VX_TYPE_BOOL</a> scalar.                                                                                                                                                                |
| in  | <i>window</i> ↔<br><i>dimension</i>     | The size of the window on which to perform the algorithm. See <a href="#">VX_CONTEXT_ATTRIBUTE_OPTICAL_FLOW_WINDOW_MAXIMUM_DIMENSION</a>                                                                  |

**Returns**

[vx\\_node](#).

**Return values**

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuOpticalFlowPyrLK ( vx\_context context, vx\_pyramid old\_images, vx\_pyramid new\_images, vx\_array old\_points, vx\_array new\_points\_estimates, vx\_array new\_points, vx\_enum termination, vx\_scalar epsilon, vx\_scalar num\_iterations, vx\_scalar use\_initial\_estimate, vx\_size window\_dimension )**

[Immediate] Computes an optical flow on two images.

**Parameters**

|     |                                         |                                                                                                                                                                                                        |
|-----|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i>                          | The reference to the overall context.                                                                                                                                                                  |
| in  | <i>old_images</i>                       | Input of first (old) image pyramid                                                                                                                                                                     |
| in  | <i>new_images</i>                       | Input of destination (new) image pyramid                                                                                                                                                               |
| in  | <i>old_points</i>                       | an array of key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> those key points are defined at the <i>old_images</i> high resolution pyramid                                 |
| in  | <i>new_points</i> ↔<br><i>estimates</i> | an array of estimation on what is the output key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> those keypoints are defined at the <i>new_images</i> high resolution pyramid |
| out | <i>new_points</i>                       | an output array of key points in a <a href="#">vx_array</a> of <a href="#">VX_TYPE_KEYPOINT</a> those key points are defined at the <i>new_images</i> high resolution pyramid                          |
| in  | <i>termination</i>                      | termination can be <a href="#">VX_TERM_CRITERIA_ITERATIONS</a> or <a href="#">VX_TERM_CRITERIA_EPSILON</a> or <a href="#">VX_TERM_CRITERIA_BOTH</a>                                                    |
| in  | <i>epsilon</i>                          | is the <a href="#">vx_float32</a> error for terminating the algorithm                                                                                                                                  |
| in  | <i>num_iterations</i>                   | is the number of iterations. Use a <a href="#">VX_TYPE_UINT32</a> scalar.                                                                                                                              |
| in  | <i>use_initial</i> ↔<br><i>estimate</i> | Can be set to either <a href="#">vx_false_e</a> or <a href="#">vx_true_e</a> .                                                                                                                         |

|    |                               |                                                                                                                                                |
|----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>window_↔<br/>dimension</i> | The size of the window on which to perform the algorithm. See <a href="#">VX_CONTE↔<br/>XT_ATTRIBUTE_OPTICAL_FLOW_WINDOW_MAXIMUM_DIMENSION</a> |
|----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.33 Phase

### 3.33.1 Detailed Description

Implements the Gradient Phase Computation Kernel.

This kernel takes two gradients in [VX\\_DF\\_IMAGE\\_S16](#) format and computes the angles for each pixel and stores this in a [VX\\_DF\\_IMAGE\\_U8](#) image.

$$\phi = \tan^{-1} \frac{\text{grad}_y(x,y)}{\text{grad}_x(x,y)}$$

Where  $\phi$  is then translated to  $0 \leq \phi < 2\pi$ . Each  $\phi$  value is then mapped to the range 0 to 255 inclusive.

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxPhaseNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) orientation)  
[Graph] Creates a Phase node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuPhase](#) ([vx\\_context](#) context, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) output)  
[Immediate] Invokes an immediate Phase.

### 3.33.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxPhaseNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) orientation )**

[Graph] Creates a Phase node.

Parameters

|     |             |                                                                            |
|-----|-------------|----------------------------------------------------------------------------|
| in  | graph       | The reference to the graph.                                                |
| in  | grad_x      | The input x image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| in  | grad_y      | The input y image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| out | orientation | The phase image. This is in <a href="#">VX_DF_IMAGE_U8</a> format.         |

See also

[VX\\_KERNEL\\_PHASE](#)

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuPhase](#) ( [vx\\_context](#) context, [vx\\_image](#) grad\_x, [vx\\_image](#) grad\_y, [vx\\_image](#) output )**

[Immediate] Invokes an immediate Phase.

Parameters

|    |         |                                       |
|----|---------|---------------------------------------|
| in | context | The reference to the overall context. |
|----|---------|---------------------------------------|

|     |               |                                                                            |
|-----|---------------|----------------------------------------------------------------------------|
| in  | <i>grad_x</i> | The input x image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| in  | <i>grad_y</i> | The input y image. This must be in <a href="#">VX_DF_IMAGE_S16</a> format. |
| out | <i>output</i> | The phase image. This will be in <a href="#">VX_DF_IMAGE_U8</a> format.    |

#### Returns

A [vx\\_status\\_e](#) enumeration.

#### Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.34 Pixel-wise Multiplication

### 3.34.1 Detailed Description

Performs element-wise multiplication between two images and a scalar value.

Pixel-wise multiplication is performed between the pixel values in two `VX_DF_IMAGE_U8` or `VX_DF_IMAGE_S16` images and a scalar floating-point number *scale*. The output image can be `VX_DF_IMAGE_U8` only if both source images are `VX_DF_IMAGE_U8` and the output image is explicitly set to `VX_DF_IMAGE_U8`. It is otherwise `VX_DF_IMAGE_S16`. If one of the input images is of type `VX_DF_IMAGE_S16`, all values are converted to `VX_DF_IMAGE_S16`.

The scale with a value of  $1/2^n$ , where  $n$  is an integer and  $0 \leq n \leq 15$ , and  $1/255$  (0x1.010102p-8 C99 float hex) must be supported. The support for other values of scale is not prohibited. Furthermore, for scale with a value of  $1/255$  the rounding policy of `VX_ROUND_POLICY_TO_NEAREST_EVEN` must be supported whereas for the scale with value of  $1/2^n$  the rounding policy of `VX_ROUND_POLICY_TO_ZERO` must be supported. The support of other rounding modes for any values of scale is not prohibited.

The rounding policy `VX_ROUND_POLICY_TO_ZERO` for this function is defined as:

$$reference(x,y,scale) = truncate(((int32_t)in1(x,y)) * ((int32_t)in2(x,y)) * (double)scale)$$

The rounding policy `VX_ROUND_POLICY_TO_NEAREST_EVEN` for this function is defined as:

$$reference(x,y,scale) = round_{to\_nearest\_even}(((int32_t)in1(x,y)) * ((int32_t)in2(x,y)) * (double)scale)$$

The overflow handling is controlled by an overflow-policy parameter. For each pixel value in the two input images:

$$out(x,y) = in_1(x,y)in_2(x,y)scale$$

### Functions

- `vx_node` `VX_API_CALL vxMultiplyNode` (`vx_graph` graph, `vx_image` in1, `vx_image` in2, `vx_scalar` scale, `vx_enum` overflow\_policy, `vx_enum` rounding\_policy, `vx_image` out)  
[Graph] Creates an pixelwise-multiplication node.
- `vx_status` `VX_API_CALL vxuMultiply` (`vx_context` context, `vx_image` in1, `vx_image` in2, `vx_float32` scale, `vx_enum` overflow\_policy, `vx_enum` rounding\_policy, `vx_image` out)  
[Immediate] Performs elementwise multiplications on pixel values in the input images and a scale.

### 3.34.2 Function Documentation

`vx_node` `VX_API_CALL vxMultiplyNode` ( `vx_graph` graph, `vx_image` in1, `vx_image` in2, `vx_scalar` scale, `vx_enum` overflow\_policy, `vx_enum` rounding\_policy, `vx_image` out )

[Graph] Creates an pixelwise-multiplication node.

Parameters

|     |                 |                                                                                                  |
|-----|-----------------|--------------------------------------------------------------------------------------------------|
| in  | graph           | The reference to the graph.                                                                      |
| in  | in1             | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> .                    |
| in  | in2             | An input image, <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> .                    |
| in  | scale           | A non-negative <code>VX_TYPE_FLOAT32</code> multiplied to each product before overflow handling. |
| in  | overflow_policy | A <code>VX_TYPE_ENUM</code> of the <code>vx_convert_policy_e</code> enumeration.                 |
| in  | rounding_policy | A <code>VX_TYPE_ENUM</code> of the <code>vx_round_policy_e</code> enumeration.                   |
| out | out             | The output image, a <code>VX_DF_IMAGE_U8</code> or <code>VX_DF_IMAGE_S16</code> image.           |

Returns

`vx_node`.

## Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL** **vxuMultiply** ( **vx\_context** *context*, **vx\_image** *in1*, **vx\_image** *in2*, **vx\_float32** *scale*, **vx\_enum** *overflow\_policy*, **vx\_enum** *rounding\_policy*, **vx\_image** *out* )

[Immediate] Performs elementwise multiplications on pixel values in the input images and a scale.

## Parameters

|     |                        |                                                                                               |
|-----|------------------------|-----------------------------------------------------------------------------------------------|
| in  | <i>context</i>         | The reference to the overall context.                                                         |
| in  | <i>in1</i>             | A <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> input image.              |
| in  | <i>in2</i>             | A <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> input image.              |
| in  | <i>scale</i>           | The scale value.                                                                              |
| in  | <i>overflow_policy</i> | A <a href="#">vx_convert_policy_e</a> enumeration.                                            |
| in  | <i>rounding_policy</i> | A <a href="#">vx_round_policy_e</a> enumeration.                                              |
| out | <i>out</i>             | The output image in <a href="#">VX_DF_IMAGE_U8</a> or <a href="#">VX_DF_IMAGE_S16</a> format. |

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <b>VX_SUCCESS</b> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.35 Remap

### 3.35.1 Detailed Description

Maps output pixels in an image from input pixels in an image.

Remap takes a remap table object [vx\\_remap](#) to map a set of output pixels back to source input pixels. A remap is typically defined as:

$$output(x,y) = input(mapx(x,y),mapy(x,y));$$

for every (x,y) in the destination image

However, the mapping functions are contained in the [vx\\_remap](#) object.

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxRemapNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_remap](#) table, [vx\\_enum](#) policy, [vx\\_image](#) output)  
[Graph] Creates a Remap Node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuRemap](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_remap](#) table, [vx\\_enum](#) policy, [vx\\_image](#) output)  
[Immediate] Remaps an output image from an input image.

### 3.35.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxRemapNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_remap](#) table, [vx\\_enum](#) policy, [vx\\_image](#) output )**

[Graph] Creates a Remap Node.

Parameters

|     |        |                                                                                                                                   |
|-----|--------|-----------------------------------------------------------------------------------------------------------------------------------|
| in  | graph  | The reference to the graph that will contain the node.                                                                            |
| in  | input  | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                   |
| in  | table  | The remap table object.                                                                                                           |
| in  | policy | An interpolation type from <a href="#">vx_interpolation_type_e</a> . <a href="#">VX_INTERPOLATION_TYPE_AREA</a> is not supported. |
| out | output | The output <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                  |

Note

Only [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#) value [VX\\_BORDER\\_MODE\\_UNDEFINED](#) or [VX\\_BORDER\\_MODE\\_CONSTANT](#) is supported.

Returns

A [vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuRemap](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_remap](#) table, [vx\\_enum](#) policy, [vx\\_image](#) output )**

[Immediate] Remaps an output image from an input image.

Parameters

|     |                |                                                                                                                                      |
|-----|----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                                                                                |
| in  | <i>input</i>   | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                      |
| in  | <i>table</i>   | The remap table object.                                                                                                              |
| in  | <i>policy</i>  | The interpolation policy from <a href="#">vx_interpolation_type_e</a> . <a href="#">VX_INTERPOLATION_TYPE_AREA</a> is not supported. |
| out | <i>output</i>  | The output <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                     |

#### Returns

A [vx\\_status\\_e](#) enumeration.



## 3.36 Scale Image

### 3.36.1 Detailed Description

Implements the Image Resizing Kernel.

Performs a Gaussian Blur on an image then half-scales it.

This kernel resizes an image from the source to the destination dimensions. The supported interpolation types are currently:

- `VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR`
- `VX_INTERPOLATION_TYPE_AREA`
- `VX_INTERPOLATION_TYPE_BILINEAR`

The sample positions used to determine output pixel values are generated by scaling the outside edges of the source image pixels to the outside edges of the destination image pixels. As described in the documentation for `vx_interpolation_type_e`, samples are taken at pixel centers. This means that, unless the scale is 1:1, the sample position for the top left destination pixel typically does not fall exactly on the top left source pixel but will be generated by interpolation.

That is, the sample positions corresponding in source and destination are defined by the following equations:

$$\begin{aligned}x_{input} &= \left( (x_{output} + 0.5) * \frac{width_{input}}{width_{output}} \right) - 0.5 \\y_{input} &= \left( (y_{output} + 0.5) * \frac{height_{input}}{height_{output}} \right) - 0.5 \\x_{output} &= \left( (x_{input} + 0.5) * \frac{width_{output}}{width_{input}} \right) - 0.5 \\y_{output} &= \left( (y_{input} + 0.5) * \frac{height_{output}}{height_{input}} \right) - 0.5\end{aligned}$$

- For `VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR`, the output value is that of the pixel whose centre is closest to the sample point.
- For `VX_INTERPOLATION_TYPE_BILINEAR`, the output value is formed by a weighted average of the nearest source pixels to the sample point. That is:

$$\begin{aligned}x_{lower} &= \lfloor x_{input} \rfloor \\y_{lower} &= \lfloor y_{input} \rfloor \\s &= x_{input} - x_{lower} \\t &= y_{input} - y_{lower} \\out\_put(x_{input}, y_{input}) &= (1-s)(1-t) * input(x_{lower}, y_{lower}) + s(1-t) * input(x_{lower} + 1, y_{lower}) \\&\quad + (1-s)t * input(x_{lower}, y_{lower} + 1) + s * t * input(x_{lower} + 1, y_{lower} + 1)\end{aligned}$$

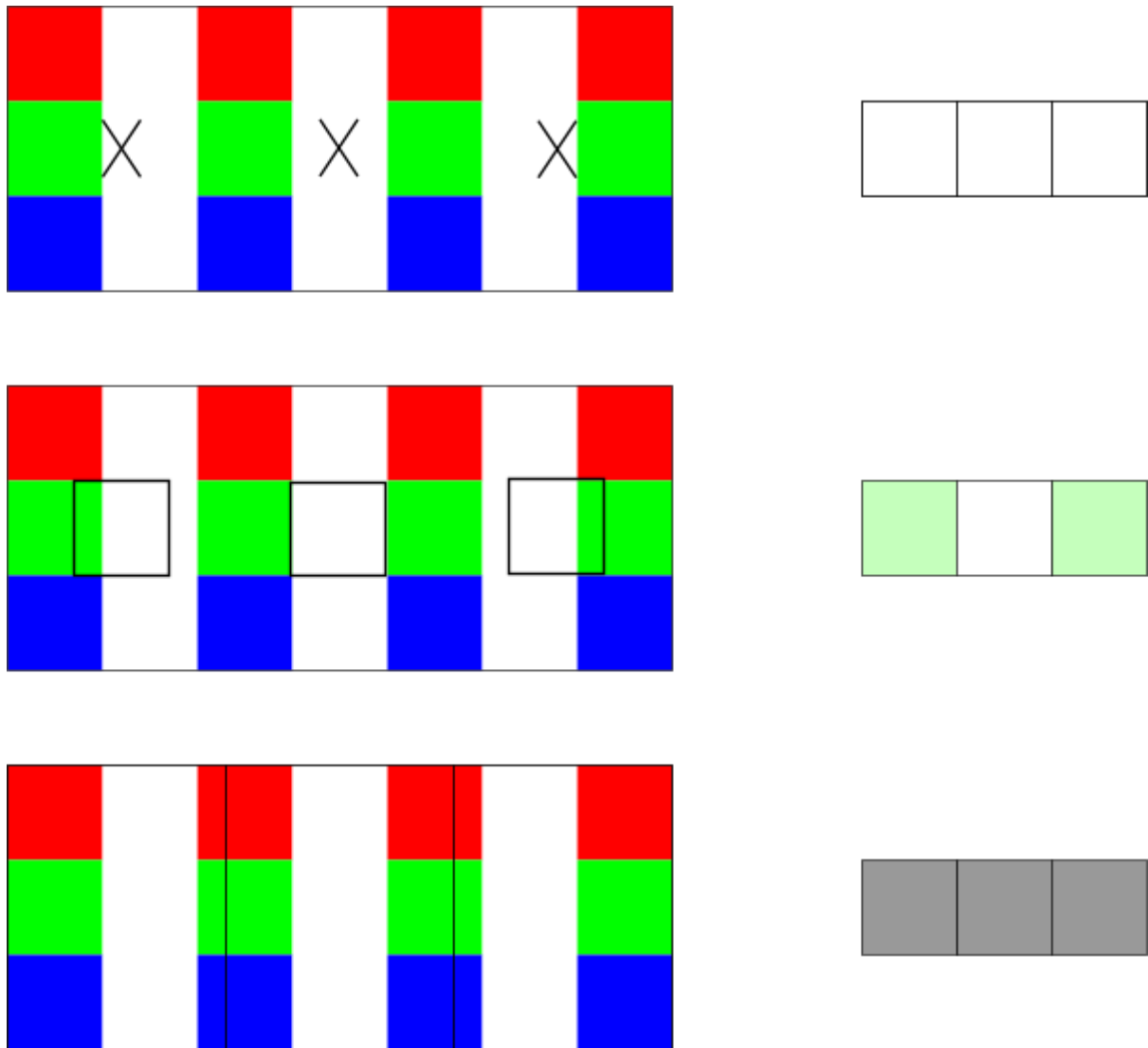
- For `VX_INTERPOLATION_TYPE_AREA`, the implementation is expected to generate each output pixel by sampling all the source pixels that are at least partly covered by the area bounded by:

$$\left( x_{output} * \frac{width_{input}}{width_{output}} \right) - 0.5, \left( y_{output} * \frac{height_{input}}{height_{output}} \right) - 0.5$$

and

$$\left( (x_{output} + 1) * \frac{width_{input}}{width_{output}} \right) - 0.5, \left( (y_{output} + 1) * \frac{height_{input}}{height_{output}} \right) - 0.5$$

The details of this sampling method are implementation-defined. The implementation should perform enough sampling to avoid aliasing, but there is no requirement that the sample areas for adjacent output pixels be disjoint, nor that the pixels be weighted evenly.



The above diagram shows three sampling methods used to shrink a 7x3 image to 3x1.

The topmost image pair shows nearest-neighbor sampling, with crosses on the left image marking the sample positions in the source that are used to generate the output image on the right. As the pixel centre closest to the sample position is white in all cases, the resulting 3x1 image is white.

The middle image pair shows bilinear sampling, with black squares on the left image showing the region in the source being sampled to generate each pixel on the destination image on the right. This sample area is always the size of an input pixel. The outer destination pixels partly sample from the outermost green pixels, so their resulting value is a weighted average of white and green.

The bottom image pair shows area sampling. The black rectangles in the source image on the left show the bounds of the projection of the destination pixels onto the source. The destination pixels on the right are formed by averaging at least those source pixels whose areas are wholly or partly contained within those rectangles. The manner of this averaging is implementation-defined; the example shown here weights the contribution of each source pixel by the amount of that pixel's area contained within the black rectangle.

## Functions

- `vx_node VX_API_CALL vxHalfScaleGaussianNode (vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size)`  
*[Graph] Performs a Gaussian Blur on an image then half-scales it.*
- `vx_node VX_API_CALL vxScaleImageNode (vx_graph graph, vx_image src, vx_image dst, vx_enum type)`  
*[Graph] Creates a Scale Image Node.*

- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxuHalfScaleGaussian](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_image](#) output, [vx\\_int32](#) kernel\_size)  
[Immediate] Performs a Gaussian Blur on an image then half-scales it.
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxuScaleImage](#) ([vx\\_context](#) context, [vx\\_image](#) src, [vx\\_image](#) dst, [vx\\_enum](#) type)  
[Immediate] Scales an input image to an output image.

### 3.36.2 Function Documentation

**`vx_node VX_API_CALL vxScaleImageNode ( vx_graph graph, vx_image src, vx_image dst, vx_enum type )`**

[Graph] Creates a Scale Image Node.

Parameters

|     |              |                                                                |
|-----|--------------|----------------------------------------------------------------|
| in  | <i>graph</i> | The reference to the graph.                                    |
| in  | <i>src</i>   | The source image of type <a href="#">VX_DF_IMAGE_U8</a> .      |
| out | <i>dst</i>   | The destination image of type <a href="#">VX_DF_IMAGE_U8</a> . |
| in  | <i>type</i>  | The interpolation type to use.                                 |

See also

[vx\\_interpolation\\_type\\_e](#).

Note

The destination image must have a defined size and format. Only [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#) value [VX\\_BORDER\\_MODE\\_UNDEFINED](#), [VX\\_BORDER\\_MODE\\_REPLICATE](#) or [VX\\_BORDER\\_MODE\\_CONSTANT](#) is supported.

Returns

[vx\\_node](#).

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**`vx_node VX_API_CALL vxHalfScaleGaussianNode ( vx_graph graph, vx_image input, vx_image output, vx_int32 kernel_size )`**

[Graph] Performs a Gaussian Blur on an image then half-scales it.

The output image size is determined by:

$$W_{output} = \frac{W_{input} + 1}{2}, H_{output} = \frac{H_{input} + 1}{2}$$

Parameters

|     |                    |                                                                      |
|-----|--------------------|----------------------------------------------------------------------|
| in  | <i>graph</i>       | The reference to the graph.                                          |
| in  | <i>input</i>       | The input <a href="#">VX_DF_IMAGE_U8</a> image.                      |
| out | <i>output</i>      | The output <a href="#">VX_DF_IMAGE_U8</a> image.                     |
| in  | <i>kernel_size</i> | The input size of the Gaussian filter. Supported values are 3 and 5. |

Returns

[vx\\_node](#).

## Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuScaleImage ( vx\_context context, vx\_image src, vx\_image dst, vx\_enum type )**

[Immediate] Scales an input image to an output image.

## Parameters

|     |                |                                                                |
|-----|----------------|----------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                          |
| in  | <i>src</i>     | The source image of type <a href="#">VX_DF_IMAGE_U8</a> .      |
| out | <i>dst</i>     | The destination image of type <a href="#">VX_DF_IMAGE_U8</a> . |
| in  | <i>type</i>    | The interpolation type.                                        |

## See also

[vx\\_interpolation\\_type\\_e](#).

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

**vx\_status VX\_API\_CALL vxuHalfScaleGaussian ( vx\_context context, vx\_image input, vx\_image output, vx\_int32 kernel\_size )**

[Immediate] Performs a Gaussian Blur on an image then half-scales it.

## Parameters

|     |                    |                                                                      |
|-----|--------------------|----------------------------------------------------------------------|
| in  | <i>context</i>     | The reference to the overall context.                                |
| in  | <i>input</i>       | The input <a href="#">VX_DF_IMAGE_U8</a> image.                      |
| out | <i>output</i>      | The output <a href="#">VX_DF_IMAGE_U8</a> image.                     |
| in  | <i>kernel_size</i> | The input size of the Gaussian filter. Supported values are 3 and 5. |

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.37 Sobel 3x3

### 3.37.1 Detailed Description

Implements the Sobel Image Filter Kernel.

This kernel produces two output planes (one can be omitted) in the x and y plane. The Sobel Operators  $G_x, G_y$  are defined as:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

### Functions

- **vx\_node** `VX_API_CALL vxSobel3x3Node` (**vx\_graph** graph, **vx\_image** input, **vx\_image** output\_x, **vx\_image** output\_y)  
[Graph] Creates a Sobel3x3 node.
- **vx\_status** `VX_API_CALL vxuSobel3x3` (**vx\_context** context, **vx\_image** input, **vx\_image** output\_x, **vx\_image** output\_y)  
[Immediate] Invokes an immediate Sobel 3x3.

### 3.37.2 Function Documentation

**vx\_node** `VX_API_CALL vxSobel3x3Node` ( **vx\_graph** graph, **vx\_image** input, **vx\_image** output\_x, **vx\_image** output\_y )

[Graph] Creates a Sobel3x3 node.

Parameters

|     |          |                                                                                     |
|-----|----------|-------------------------------------------------------------------------------------|
| in  | graph    | The reference to the graph.                                                         |
| in  | input    | The input image in <code>VX_DF_IMAGE_U8</code> format.                              |
| out | output_x | [optional] The output gradient in the x direction in <code>VX_DF_IMAGE_S16</code> . |
| out | output_y | [optional] The output gradient in the y direction in <code>VX_DF_IMAGE_S16</code> . |

See also

[VX\\_KERNEL\\_SOBEL\\_3x3](#)

Returns

**vx\_node**.

Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>vx_node</b> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** `VX_API_CALL vxuSobel3x3` ( **vx\_context** context, **vx\_image** input, **vx\_image** output\_x, **vx\_image** output\_y )

[Immediate] Invokes an immediate Sobel 3x3.

Parameters

|     |          |                                                                                     |
|-----|----------|-------------------------------------------------------------------------------------|
| in  | context  | The reference to the overall context.                                               |
| in  | input    | The input image in <code>VX_DF_IMAGE_U8</code> format.                              |
| out | output_x | [optional] The output gradient in the x direction in <code>VX_DF_IMAGE_S16</code> . |

|     |                 |                                                                                        |
|-----|-----------------|----------------------------------------------------------------------------------------|
| out | <i>output_y</i> | [optional] The output gradient in the y direction in <a href="#">VX_DF_IMAGE_S16</a> . |
|-----|-----------------|----------------------------------------------------------------------------------------|

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.38 TableLookup

### 3.38.1 Detailed Description

Implements the Table Lookup Image Kernel.

This kernel uses each pixel in an image to index into a LUT and put the indexed LUT value into the output image. The format supported is [VX\\_DF\\_IMAGE\\_U8](#).

### Functions

- [vx\\_node](#) [VX\\_API\\_CALL vxTableLookupNode](#) ([vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_lut](#) lut, [vx\\_image](#) output)  
[Graph] Creates a Table Lookup node.
- [vx\\_status](#) [VX\\_API\\_CALL vxuTableLookup](#) ([vx\\_context](#) context, [vx\\_image](#) input, [vx\\_lut](#) lut, [vx\\_image](#) output)  
[Immediate] Processes the image through the LUT.

### 3.38.2 Function Documentation

**[vx\\_node](#) [VX\\_API\\_CALL vxTableLookupNode](#) ( [vx\\_graph](#) graph, [vx\\_image](#) input, [vx\\_lut](#) lut, [vx\\_image](#) output )**

[Graph] Creates a Table Lookup node.

Parameters

|     |        |                                                           |
|-----|--------|-----------------------------------------------------------|
| in  | graph  | The reference to the graph.                               |
| in  | input  | The input image in <a href="#">VX_DF_IMAGE_U8</a> .       |
| in  | lut    | The LUT which is of type <a href="#">VX_TYPE_UINT8</a> .  |
| out | output | The output image of type <a href="#">VX_DF_IMAGE_U8</a> . |

Returns

[vx\\_node](#).

Return values

|                         |                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_node</a> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

**[vx\\_status](#) [VX\\_API\\_CALL vxuTableLookup](#) ( [vx\\_context](#) context, [vx\\_image](#) input, [vx\\_lut](#) lut, [vx\\_image](#) output )**

[Immediate] Processes the image through the LUT.

Parameters

|     |         |                                                         |
|-----|---------|---------------------------------------------------------|
| in  | context | The reference to the overall context.                   |
| in  | input   | The input image in <a href="#">VX_DF_IMAGE_U8</a>       |
| in  | lut     | The LUT which is of type <a href="#">VX_TYPE_UINT8</a>  |
| out | output  | The output image of type <a href="#">VX_DF_IMAGE_U8</a> |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.39 Thresholding

### 3.39.1 Detailed Description

Thresholds an input image and produces an output Boolean image.

In `VX_THRESHOLD_TYPE_BINARY`, the output is determined by:

$$dst(x,y) = \begin{cases} 255 & \text{if } src(x,y) > threshold \\ 0 & \text{otherwise} \end{cases}$$

In `VX_THRESHOLD_TYPE_RANGE`, the output is determined by:

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > upper \\ 0 & \text{if } src(x,y) < lower \\ 255 & \text{otherwise} \end{cases}$$

### Functions

- `vx_node` `VX_API_CALL vxThresholdNode` (`vx_graph` *graph*, `vx_image` *input*, `vx_threshold` *thresh*, `vx_image` *output*)  
[Graph] Creates a Threshold node.
- `vx_status` `VX_API_CALL vxuThreshold` (`vx_context` *context*, `vx_image` *input*, `vx_threshold` *thresh*, `vx_image` *output*)  
[Immediate] Threshold's an input image and produces a `VX_DF_IMAGE_U8` \* boolean image.

### 3.39.2 Function Documentation

**`vx_node` `VX_API_CALL vxThresholdNode` ( `vx_graph` *graph*, `vx_image` *input*, `vx_threshold` *thresh*, `vx_image` *output* )**

[Graph] Creates a Threshold node.

Parameters

|     |               |                                                                       |
|-----|---------------|-----------------------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                                           |
| in  | <i>input</i>  | The input image. <code>VX_DF_IMAGE_U8</code> is supported.            |
| in  | <i>thresh</i> | The thresholding object that defines the parameters of the operation. |
| out | <i>output</i> | The output Boolean image. Values are either 0 or 255.                 |

Returns

`vx_node`.

Return values

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <code>vxGetStatus</code> |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|

**`vx_status` `VX_API_CALL vxuThreshold` ( `vx_context` *context*, `vx_image` *input*, `vx_threshold` *thresh*, `vx_image` *output* )**

[Immediate] Threshold's an input image and produces a `VX_DF_IMAGE_U8` \* boolean image.

Parameters

|    |                |                                                            |
|----|----------------|------------------------------------------------------------|
| in | <i>context</i> | The reference to the overall context.                      |
| in | <i>input</i>   | The input image. <code>VX_DF_IMAGE_U8</code> is supported. |



|     |               |                                                                       |
|-----|---------------|-----------------------------------------------------------------------|
| in  | <i>thresh</i> | The thresholding object that defines the parameters of the operation. |
| out | <i>output</i> | The output Boolean image. Values are either 0 or 255.                 |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.40 Warp Affine

### 3.40.1 Detailed Description

Performs an affine transform on an image.

This kernel performs an affine transform with a 2x3 Matrix  $M$  with this method of pixel coordinate translation:

$$x0 = M_{1,1} * x + M_{1,2} * y + M_{1,3} \quad (3.28)$$

$$y0 = M_{2,1} * x + M_{2,2} * y + M_{2,3} \quad (3.29)$$

$$out\ put(x,y) = in\ put(x0,y0) \quad (3.30)$$

This translates into the C declaration:

```
// x0 = a x + b y + c;
// y0 = d x + e y + f;
vx_float32 mat[3][2] = {
 {a, d}, // 'x' coefficients
 {b, e}, // 'y' coefficients
 {c, f}, // 'offsets'
};
vx_matrix matrix = vxCreateMatrix(context,
 VX_TYPE_FLOAT32, 2, 3);
vxWriteMatrix(matrix, mat);
```

## Functions

- **vx\_status** **VX\_API\_CALL** **vxuWarpAffine** (**vx\_context** context, **vx\_image** input, **vx\_matrix** matrix, **vx\_enum** type, **vx\_image** output)  
[Immediate] Performs an Affine warp on an image.
- **vx\_node** **VX\_API\_CALL** **vxWarpAffineNode** (**vx\_graph** graph, **vx\_image** input, **vx\_matrix** matrix, **vx\_enum** type, **vx\_image** output)  
[Graph] Creates an Affine Warp Node.

### 3.40.2 Function Documentation

**vx\_node** **VX\_API\_CALL** **vxWarpAffineNode** ( **vx\_graph** graph, **vx\_image** input, **vx\_matrix** matrix, **vx\_enum** type, **vx\_image** output )

[Graph] Creates an Affine Warp Node.

Parameters

|     |        |                                                                                                                  |
|-----|--------|------------------------------------------------------------------------------------------------------------------|
| in  | graph  | The reference to the graph.                                                                                      |
| in  | input  | The input <b>VX_DF_IMAGE_U8</b> image.                                                                           |
| in  | matrix | The affine matrix. Must be 2x3 of type <b>VX_TYPE_FLOAT32</b> .                                                  |
| in  | type   | The interpolation type from <b>vx_interpolation_type_e</b> . <b>VX_INTERPOLATION_TYPE_AREA</b> is not supported. |
| out | output | The output <b>VX_DF_IMAGE_U8</b> image.                                                                          |

Note

Only **VX\_NODE\_ATTRIBUTE\_BORDER\_MODE** value **VX\_BORDER\_MODE\_UNDEFINED** or **VX\_BORDER\_MODE\_CONSTANT** is supported.

Returns

**vx\_node**.

Return values

|                      |                                                                                                                            |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>vx_node</code> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status VX\_API\_CALL vxuWarpAffine ( vx\_context *context*, vx\_image *input*, vx\_matrix *matrix*, vx\_enum *type*, vx\_image *output* )**

[Immediate] Performs an Affine warp on an image.

Parameters

|     |                |                                                                                                                                    |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                                                                              |
| in  | <i>input</i>   | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                    |
| in  | <i>matrix</i>  | The affine matrix. Must be 2x3 of type <a href="#">VX_TYPE_FLOAT32</a> .                                                           |
| in  | <i>type</i>    | The interpolation type from <a href="#">vx_interpolation_type_e</a> . <a href="#">VX_INTERPOLATION_TYPE_AREA</a> is not supported. |
| out | <i>output</i>  | The output <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                   |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <a href="#">VX_SUCCESS</a> | Success                                              |
| *                          | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.41 Warp Perspective

### 3.41.1 Detailed Description

Performs a perspective transform on an image.

This kernel performs an perspective transform with a 3x3 Matrix  $M$  with this method of pixel coordinate translation:

$$x0 = M_{1,1} * x + M_{1,2} * y + M_{1,3} \quad (3.31)$$

$$y0 = M_{2,1} * x + M_{2,2} * y + M_{2,3} \quad (3.32)$$

$$z0 = M_{3,1} * x + M_{3,2} * y + M_{3,3} \quad (3.33)$$

$$out\ put(x,y) = input\left(\frac{x0}{z0}, \frac{y0}{z0}\right) \quad (3.34)$$

This translates into the C declaration:

```
// x0 = a x + b y + c;
// y0 = d x + e y + f;
// z0 = g x + h y + i;
vx_float32 mat[3][3] = {
 {a, d, g}, // 'x' coefficients
 {b, e, h}, // 'y' coefficients
 {c, f, i}, // 'offsets'
};
vx_matrix matrix = vxCreateMatrix(context,
 VX_TYPE_FLOAT32, 3, 3);
vxWriteMatrix(matrix, mat);
```

## Functions

- **vx\_status** **VX\_API\_CALL** **vxuWarpPerspective** (**vx\_context** context, **vx\_image** input, **vx\_matrix** matrix, **vx\_enum** type, **vx\_image** output)  
[Immediate] Performs an Perspective warp on an image.
- **vx\_node** **VX\_API\_CALL** **vxWarpPerspectiveNode** (**vx\_graph** graph, **vx\_image** input, **vx\_matrix** matrix, **vx\_enum** type, **vx\_image** output)  
[Graph] Creates a Perspective Warp Node.

### 3.41.2 Function Documentation

**vx\_node** **VX\_API\_CALL** **vxWarpPerspectiveNode** ( **vx\_graph** *graph*, **vx\_image** *input*, **vx\_matrix** *matrix*, **vx\_enum** *type*, **vx\_image** *output* )

[Graph] Creates a Perspective Warp Node.

Parameters

|     |               |                                                                                                                                    |
|-----|---------------|------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>graph</i>  | The reference to the graph.                                                                                                        |
| in  | <i>input</i>  | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                    |
| in  | <i>matrix</i> | The perspective matrix. Must be 3x3 of type <a href="#">VX_TYPE_FLOAT32</a> .                                                      |
| in  | <i>type</i>   | The interpolation type from <a href="#">vx_interpolation_type_e</a> . <a href="#">VX_INTERPOLATION_TYPE_AREA</a> is not supported. |
| out | <i>output</i> | The output <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                   |

Note

Only [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#) value [VX\\_BORDER\\_MODE\\_UNDEFINED](#) or [VX\\_BORDER\\_MODE\\_CONSTANT](#) is supported.

Returns

[vx\\_node](#).

## Return values

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>vx_node</i> | A node reference. Any possible errors preventing a successful creation should be checked using <a href="#">vxGetStatus</a> |
|----------------|----------------------------------------------------------------------------------------------------------------------------|

**vx\_status** **VX\_API\_CALL** **vxuWarpPerspective** ( **vx\_context** *context*, **vx\_image** *input*, **vx\_matrix** *matrix*, **vx\_enum** *type*, **vx\_image** *output* )

[Immediate] Performs an Perspective warp on an image.

## Parameters

|     |                |                                                                                                                                     |
|-----|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| in  | <i>context</i> | The reference to the overall context.                                                                                               |
| in  | <i>input</i>   | The input <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                     |
| in  | <i>matrix</i>  | The perspective matrix. Must be 3x3 of type <a href="#">VX_TYPE_FLOAT32</a> .                                                       |
| in  | <i>type</i>    | The interpolation type from <a href="#">vx_interpolation_type_e</a> . <a href="#">VX_INTERPOLATION↵_TYPE_AREA</a> is not supported. |
| out | <i>output</i>  | The output <a href="#">VX_DF_IMAGE_U8</a> image.                                                                                    |

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>VX_SUCCESS</i> | Success                                              |
| *                 | An error occurred. See <a href="#">vx_status_e</a> . |

## 3.42 Basic Features

### 3.42.1 Detailed Description

The basic parts of OpenVX needed for computation.

Types in OpenVX intended to be derived from the C99 Section 7.18 standard definition of fixed width types.

#### Modules

- [Objects](#)

*Defines the basic objects within OpenVX.*

#### Data Structures

- struct [vx\\_coordinates2d\\_t](#)  
*The 2D Coordinates structure. [More...](#)*
- struct [vx\\_coordinates3d\\_t](#)  
*The 3D Coordinates structure. [More...](#)*
- struct [vx\\_delta\\_rectangle\\_t](#)  
*The changes in dimensions of the rectangle between input and output images in an output parameter validator. Used in conjunction with [VX\\_META\\_FORMAT\\_ATTRIBUTE\\_DELTA\\_RECTANGLE](#) and [vxSetMetaFormatAttribute](#). [More...](#)*
- struct [vx\\_keypoint\\_t](#)  
*The keypoint data structure. [More...](#)*
- struct [vx\\_rectangle\\_t](#)  
*The rectangle data structure that is shared with the users. [More...](#)*

#### Macros

- `#define VX\_ATTRIBUTE\_BASE(vendor, object) (((vendor) << 20) | (object << 8))`  
*Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.*
- `#define VX\_ATTRIBUTE\_ID\_MASK (0x000000FF)`  
*An object's attribute ID is within the range of  $[0, 2^8 - 1]$  (inclusive).*
- `#define VX\_DF\_IMAGE(a, b, c, d) ((a) | (b << 8) | (c << 16) | (d << 24))`  
*Converts a set of four chars into a `uint32_t` container of a `VX_DF_IMAGE` code.*
- `#define VX\_ENUM\_BASE(vendor, id) (((vendor) << 20) | (id << 12))`  
*Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.*
- `#define VX\_ENUM\_MASK (0x00000FFF)`  
*A generic enumeration list can have values between  $[0, 2^{12} - 1]$  (inclusive).*
- `#define VX\_ENUM\_TYPE(e) (((vx\_uint32)e & VX\_ENUM\_TYPE\_MASK) >> 12)`  
*A macro to extract the enum type from an enumerated value.*
- `#define VX\_ENUM\_TYPE\_MASK (0x000FF000)`  
*A type of enumeration. The valid range is between  $[0, 2^8 - 1]$  (inclusive).*
- `#define VX\_FMT\_REF "%p"`
- `#define VX\_FMT\_SIZE "%zu"`
- `#define VX\_KERNEL\_BASE(vendor, lib) (((vendor) << 20) | (lib << 12))`  
*Defines the manner in which to combine the Vendor and Library IDs to get the base value of the enumeration.*
- `#define VX\_KERNEL\_MASK (0x00000FFF)`  
*An individual kernel in a library has its own unique ID within  $[0, 2^{12} - 1]$  (inclusive).*
- `#define VX\_LIBRARY(e) (((vx\_uint32)e & VX\_LIBRARY\_MASK) >> 12)`  
*A macro to extract the kernel library enumeration from a enumerated kernel value.*
- `#define VX\_LIBRARY\_MASK (0x000FF000)`  
*A library is a set of vision kernels with its own ID supplied by a vendor. The vendor defines the library ID. The range is  $[0, 2^8 - 1]$  inclusive.*

- `#define VX_MAX_LOG_MESSAGE_LEN (1024)`  
*Defines the maximum length of a message buffer to copy from the log.*
- `#define VX_SCALE_UNITY (1024u)`
- `#define VX_TYPE(e) (((vx_uint32)e & VX_TYPE_MASK) >> 8)`  
*A macro to extract the type from an enumerated attribute value.*
- `#define VX_TYPE_MASK (0x000FFF00)`  
*A type mask removes the scalar/object type from the attribute. It is 3 nibbles in size and is contained between the third and second byte.*
- `#define VX_VENDOR(e) (((vx_uint32)e & VX_VENDOR_MASK) >> 20)`  
*A macro to extract the vendor ID from the enumerated value.*
- `#define VX_VENDOR_MASK (0xFFF00000)`  
*Vendor IDs are 2 nibbles in size and are located in the upper byte of the 4 bytes of an enumeration.*
- `#define VX_VERSION VX_VERSION_1_0`
- `#define VX_VERSION_1_0 (VX_VERSION_MAJOR(1) | VX_VERSION_MINOR(0))`  
*Defines the predefined version number for 1.0.*
- `#define VX_VERSION_MAJOR(x) ((x & 0xFF) << 8)`
- `#define VX_VERSION_MINOR(x) ((x & 0xFF) << 0)`

## Typedefs

- `typedef char vx_char`  
*An 8 bit ASCII character.*
- `typedef uint32_t vx_df_image`  
*Used to hold a VX\_DF\_IMAGE code to describe the pixel format and color space.*
- `typedef int32_t vx_enum`  
*Sets the standard enumeration type size to be a fixed quantity.*
- `typedef float vx_float32`  
*A 32-bit float value.*
- `typedef double vx_float64`  
*A 64-bit float value (aka double).*
- `typedef int16_t vx_int16`  
*A 16-bit signed value.*
- `typedef int32_t vx_int32`  
*A 32-bit signed value.*
- `typedef int64_t vx_int64`  
*A 64-bit signed value.*
- `typedef int8_t vx_int8`  
*An 8-bit signed value.*
- `typedef size_t vx_size`  
*A wrapper of `size_t` to keep the naming convention uniform.*
- `typedef vx_enum vx_status`  
*A formal status type with known fixed size.*
- `typedef uint16_t vx_uint16`  
*A 16-bit unsigned value.*
- `typedef uint32_t vx_uint32`  
*A 32-bit unsigned value.*
- `typedef uint64_t vx_uint64`  
*A 64-bit unsigned value.*
- `typedef uint8_t vx_uint8`  
*An 8-bit unsigned value.*

## Enumerations

- enum `vx_bool` {  
`vx_false_e` = 0,  
`vx_true_e` }

*A Boolean value. This allows 0 to be FALSE, as it is in C, and any non-zero to be TRUE.*

- enum `vx_channel_e` {  
`VX_CHANNEL_0` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x0,  
`VX_CHANNEL_1` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x1,  
`VX_CHANNEL_2` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x2,  
`VX_CHANNEL_3` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x3,  
`VX_CHANNEL_R` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x10,  
`VX_CHANNEL_G` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x11,  
`VX_CHANNEL_B` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x12,  
`VX_CHANNEL_A` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x13,  
`VX_CHANNEL_Y` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x14,  
`VX_CHANNEL_U` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x15,  
`VX_CHANNEL_V` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CHANNEL` << 12)) + 0x16 }

*The channel enumerations for channel extractions.*

- enum `vx_convert_policy_e` {  
`VX_CONVERT_POLICY_WRAP` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CONVERT_POLICY` << 12)) + 0x0,  
`VX_CONVERT_POLICY_SATURATE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_CONVERT_POLICY` << 12)) + 0x1 }

*The Conversion Policy Enumeration.*

- enum `vx_df_image_e` {  
`VX_DF_IMAGE_VIRT` = (( 'V' ) | ( 'I' << 8) | ( 'R' << 16) | ( 'T' << 24)),  
`VX_DF_IMAGE_RGB` = (( 'R' ) | ( 'G' << 8) | ( 'B' << 16) | ( '2' << 24)),  
`VX_DF_IMAGE_RGBX` = (( 'R' ) | ( 'G' << 8) | ( 'B' << 16) | ( 'A' << 24)),  
`VX_DF_IMAGE_NV12` = (( 'N' ) | ( 'V' << 8) | ( '1' << 16) | ( '2' << 24)),  
`VX_DF_IMAGE_NV21` = (( 'N' ) | ( 'V' << 8) | ( '2' << 16) | ( '1' << 24)),  
`VX_DF_IMAGE_UYVY` = (( 'U' ) | ( 'Y' << 8) | ( 'V' << 16) | ( 'Y' << 24)),  
`VX_DF_IMAGE_YUYV` = (( 'Y' ) | ( 'U' << 8) | ( 'Y' << 16) | ( 'V' << 24)),  
`VX_DF_IMAGE_IYUV` = (( 'I' ) | ( 'Y' << 8) | ( 'U' << 16) | ( 'V' << 24)),  
`VX_DF_IMAGE_YUV4` = (( 'Y' ) | ( 'U' << 8) | ( 'V' << 16) | ( '4' << 24)),  
`VX_DF_IMAGE_U8` = (( 'U' ) | ( '0' << 8) | ( '0' << 16) | ( '8' << 24)),  
`VX_DF_IMAGE_U16` = (( 'U' ) | ( '0' << 8) | ( '1' << 16) | ( '6' << 24)),  
`VX_DF_IMAGE_S16` = (( 'S' ) | ( '0' << 8) | ( '1' << 16) | ( '6' << 24)),  
`VX_DF_IMAGE_U32` = (( 'U' ) | ( '0' << 8) | ( '3' << 16) | ( '2' << 24)),  
`VX_DF_IMAGE_S32` = (( 'S' ) | ( '0' << 8) | ( '3' << 16) | ( '2' << 24)) }

*Based on the `VX_DF_IMAGE` definition.*

- enum `vx_enum_e` {



```

VX_ENUM_DIRECTION = 0x00,
VX_ENUM_ACTION = 0x01,
VX_ENUM_HINT = 0x02,
VX_ENUM_DIRECTIVE = 0x03,
VX_ENUM_INTERPOLATION = 0x04,
VX_ENUM_OVERFLOW = 0x05,
VX_ENUM_COLOR_SPACE = 0x06,
VX_ENUM_COLOR_RANGE = 0x07,
VX_ENUM_PARAMETER_STATE = 0x08,
VX_ENUM_CHANNEL = 0x09,
VX_ENUM_CONVERT_POLICY = 0x0A,
VX_ENUM_THRESHOLD_TYPE = 0x0B,
VX_ENUM_BORDER_MODE = 0x0C,
VX_ENUM_COMPARISON = 0x0D,
VX_ENUM_IMPORT_MEM = 0x0E,
VX_ENUM_TERM_CRITERIA = 0x0F,
VX_ENUM_NORM_TYPE = 0x10,
VX_ENUM_ACCESSOR = 0x11,
VX_ENUM_ROUND_POLICY = 0x12 }

```

*The set of supported enumerations in OpenVX.*

- enum `vx_interpolation_type_e` {  
`VX_INTERPOLATION_TYPE_NEAREST_NEIGHBOR` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_I`↵  
`INTERPOLATION` << 12)) + 0x0,  
`VX_INTERPOLATION_TYPE_BILINEAR` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_INTERPOLATI`↵  
`ON` << 12)) + 0x1,  
`VX_INTERPOLATION_TYPE_AREA` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_INTERPOLATION`  
<< 12)) + 0x2 }

*The image reconstruction filters supported by image resampling operations.*

- enum `vx_status_e` {  
`VX_STATUS_MIN` = -25,  
`VX_ERROR_REFERENCE_NONZERO` = -24,  
`VX_ERROR_MULTIPLE_WRITERS` = -23,  
`VX_ERROR_GRAPH_ABANDONED` = -22,  
`VX_ERROR_GRAPH_SCHEDULED` = -21,  
`VX_ERROR_INVALID_SCOPE` = -20,  
`VX_ERROR_INVALID_NODE` = -19,  
`VX_ERROR_INVALID_GRAPH` = -18,  
`VX_ERROR_INVALID_TYPE` = -17,  
`VX_ERROR_INVALID_VALUE` = -16,  
`VX_ERROR_INVALID_DIMENSION` = -15,  
`VX_ERROR_INVALID_FORMAT` = -14,  
`VX_ERROR_INVALID_LINK` = -13,  
`VX_ERROR_INVALID_REFERENCE` = -12,  
`VX_ERROR_INVALID_MODULE` = -11,  
`VX_ERROR_INVALID_PARAMETERS` = -10,  
`VX_ERROR_OPTIMIZED_AWAY` = -9,  
`VX_ERROR_NO_MEMORY` = -8,  
`VX_ERROR_NO_RESOURCES` = -7,  
`VX_ERROR_NOT_COMPATIBLE` = -6,  
`VX_ERROR_NOT_ALLOCATED` = -5,  
`VX_ERROR_NOT_SUFFICIENT` = -4,  
`VX_ERROR_NOT_SUPPORTED` = -3,  
`VX_ERROR_NOT_IMPLEMENTED` = -2,  
`VX_FAILURE` = -1,  
`VX_SUCCESS` = 0 }

*The enumeration of all status codes.*

- enum `vx_type_e` {

```

VX_TYPE_INVALID = 0x000,
VX_TYPE_CHAR = 0x001,
VX_TYPE_INT8 = 0x002,
VX_TYPE_UINT8 = 0x003,
VX_TYPE_INT16 = 0x004,
VX_TYPE_UINT16 = 0x005,
VX_TYPE_INT32 = 0x006,
VX_TYPE_UINT32 = 0x007,
VX_TYPE_INT64 = 0x008,
VX_TYPE_UINT64 = 0x009,
VX_TYPE_FLOAT32 = 0x00A,
VX_TYPE_FLOAT64 = 0x00B,
VX_TYPE_ENUM = 0x00C,
VX_TYPE_SIZE = 0x00D,
VX_TYPE_DF_IMAGE = 0x00E,
VX_TYPE_BOOL = 0x010,
VX_TYPE_SCALAR_MAX,
VX_TYPE_RECTANGLE = 0x020,
VX_TYPE_KEYPOINT = 0x021,
VX_TYPE_COORDINATES2D = 0x022,
VX_TYPE_COORDINATES3D = 0x023,
VX_TYPE_USER_STRUCT_START = 0x100,
VX_TYPE_STRUCT_MAX = VX_TYPE_USER_STRUCT_START - 1,
VX_TYPE_VENDOR_STRUCT_START = 0x400,
VX_TYPE_USER_STRUCT_END = VX_TYPE_VENDOR_STRUCT_START - 1,
VX_TYPE_VENDOR_STRUCT_END = 0x7FF,
VX_TYPE_REFERENCE = 0x800,
VX_TYPE_CONTEXT = 0x801,
VX_TYPE_GRAPH = 0x802,
VX_TYPE_NODE = 0x803,
VX_TYPE_KERNEL = 0x804,
VX_TYPE_PARAMETER = 0x805,
VX_TYPE_DELAY = 0x806,
VX_TYPE_LUT = 0x807,
VX_TYPE_DISTRIBUTION = 0x808,
VX_TYPE_PYRAMID = 0x809,
VX_TYPE_THRESHOLD = 0x80A,
VX_TYPE_MATRIX = 0x80B,
VX_TYPE_CONVOLUTION = 0x80C,
VX_TYPE_SCALAR = 0x80D,
VX_TYPE_ARRAY = 0x80E,
VX_TYPE_IMAGE = 0x80F,
VX_TYPE_REMAP = 0x810,
VX_TYPE_ERROR = 0x811,
VX_TYPE_META_FORMAT = 0x812,
VX_TYPE_VENDOR_OBJECT_START = 0xC00,
VX_TYPE_OBJECT_MAX = VX_TYPE_VENDOR_OBJECT_START - 1,
VX_TYPE_VENDOR_OBJECT_END = 0xFFF }

```

*The type enumeration lists all the known types in OpenVX.*

- enum `vx_vendor_id_e` {

```

VX_ID_KHRONOS = 0x000,
VX_ID_TI = 0x001,
VX_ID_QUALCOMM = 0x002,
VX_ID_NVIDIA = 0x003,
VX_ID_ARM = 0x004,
VX_ID_BDTI = 0x005,
VX_ID_RENESAS = 0x006,
VX_ID_VIVANTE = 0x007,
VX_ID_XILINX = 0x008,
VX_ID_AXIS = 0x009,
VX_ID_MOVIDIUS = 0x00A,
VX_ID_SAMSUNG = 0x00B,
VX_ID_FREESCALE = 0x00C,
VX_ID_AMD = 0x00D,
VX_ID_BROADCOM = 0x00E,
VX_ID_INTEL = 0x00F,
VX_ID_MARVELL = 0x010,
VX_ID_MEDIATEK = 0x011,
VX_ID_ST = 0x012,
VX_ID_CEVA = 0x013,
VX_ID_ITSEEZ = 0x014,
VX_ID_IMAGINATION = 0x015,
VX_ID_COGNIVUE = 0x016,
VX_ID_VIDEANTIS = 0x017,
VX_ID_SYNOPSYS = 0x018,
VX_ID_MAX = 0xFF,
VX_ID_DEFAULT = VX_ID_MAX }

```

The Vendor ID of the Implementation. As new vendors submit their implementations, this enumeration will grow.

## Functions

- `vx_status` `VX_API_CALL vxGetStatus(vx_reference reference)`

Provides a generic API to return status values from Object constructors if they fail.

### 3.42.2 Data Structure Documentation

#### **struct vx\_coordinates2d\_t**

The 2D Coordinates structure.

Definition at line 1427 of file `vx_types.h`.

Data Fields

|                        |   |                   |
|------------------------|---|-------------------|
| <code>vx_uint32</code> | x | The X coordinate. |
| <code>vx_uint32</code> | y | The Y coordinate. |

#### **struct vx\_coordinates3d\_t**

The 3D Coordinates structure.

Definition at line 1435 of file `vx_types.h`.

Data Fields

|                        |   |                   |
|------------------------|---|-------------------|
| <code>vx_uint32</code> | x | The X coordinate. |
| <code>vx_uint32</code> | y | The Y coordinate. |

|                           |   |                   |
|---------------------------|---|-------------------|
| <a href="#">vx_uint32</a> | z | The Z coordinate. |
|---------------------------|---|-------------------|

### struct vx\_delta\_rectangle\_t

The changes in dimensions of the rectangle between input and output images in an output parameter validator. Used in conjunction with [VX\\_META\\_FORMAT\\_ATTRIBUTE\\_DELTA\\_RECTANGLE](#) and [vxSetMetaFormatAttribute](#).

See also

[vx\\_kernel\\_output\\_validate\\_f](#)  
[vx\\_meta\\_format](#)

Definition at line 1417 of file [vx\\_types.h](#).

#### Data Fields

|                          |               |                            |
|--------------------------|---------------|----------------------------|
| <a href="#">vx_int32</a> | delta_start_x | The change in the start x. |
| <a href="#">vx_int32</a> | delta_start_y | The change in the start y. |
| <a href="#">vx_int32</a> | delta_end_x   | The change in the end x.   |
| <a href="#">vx_int32</a> | delta_end_y   | The change in the end y.   |

### struct vx\_keypoint\_t

The keypoint data structure.

Definition at line 1389 of file [vx\\_types.h](#).

#### Data Fields

|                            |                 |                                                                                  |
|----------------------------|-----------------|----------------------------------------------------------------------------------|
| <a href="#">vx_int32</a>   | x               | The x coordinate.                                                                |
| <a href="#">vx_int32</a>   | y               | The y coordinate.                                                                |
| <a href="#">vx_float32</a> | strength        | The strength of the keypoint. Its definition is specific to the corner detector. |
| <a href="#">vx_float32</a> | scale           | Initialized to 0 by corner detectors.                                            |
| <a href="#">vx_float32</a> | orientation     | Initialized to 0 by corner detectors.                                            |
| <a href="#">vx_int32</a>   | tracking_status | A zero indicates a lost point. Initialized to 1 by corner detectors.             |
| <a href="#">vx_float32</a> | error           | A tracking method specific error. Initialized to 0 by corner detectors.          |

### struct vx\_rectangle\_t

The rectangle data structure that is shared with the users.

Definition at line 1402 of file [vx\\_types.h](#).

#### Data Fields

|                           |         |                         |
|---------------------------|---------|-------------------------|
| <a href="#">vx_uint32</a> | start_x | The Start X coordinate. |
| <a href="#">vx_uint32</a> | start_y | The Start Y coordinate. |
| <a href="#">vx_uint32</a> | end_x   | The End X coordinate.   |
| <a href="#">vx_uint32</a> | end_y   | The End Y coordinate.   |

## 3.42.3 Macro Definition Documentation

**#define VX\_VERSION\_MAJOR( x ) ((x & 0xFF) << 8)**

Defines the major version number macro.

Definition at line 56 of file [vx.h](#).

**#define VX\_VERSION\_MINOR( x ) ((x & 0xFF) << 0)**

Defines the minor version number macro.

Definition at line 61 of file [vx.h](#).

**#define VX\_VERSION VX\_VERSION\_1\_0**

Defines the OpenVX Version Number.

Definition at line 71 of file [vx.h](#).

**#define VX\_TYPE\_MASK (0x000FFF00)**

A type mask removes the scalar/object type from the attribute. It is 3 nibbles in size and is contained between the third and second byte.

See also

[vx\\_type\\_e](#)

Definition at line 429 of file [vx\\_types.h](#).

**#define VX\_DF\_IMAGE( a, b, c, d ) ((a) | (b << 8) | (c << 16) | (d << 24))**

Converts a set of four chars into a `uint32_t` container of a `VX_DF_IMAGE` code.

Note

Use a [vx\\_df\\_image](#) variable to hold the value.

**#define VX\_ENUM\_BASE( vendor, id ) (((vendor) << 20) | (id << 12))**

Defines the manner in which to combine the Vendor and Object IDs to get the base value of the enumeration.

From any enumerated value (with exceptions), the vendor, and enumeration type should be extractable. Those types that are exceptions are [vx\\_vendor\\_id\\_e](#), [vx\\_type\\_e](#), [vx\\_enum\\_e](#), [vx\\_df\\_image\\_e](#), and [vx\\_bool](#).

**#define VX\_FMT\_REF "%p"**

Use to aid in debugging values in OpenVX.

Definition at line 1278 of file [vx\\_types.h](#).

**#define VX\_FMT\_SIZE "%zu"**

Use to aid in debugging values in OpenVX.

Definition at line 1282 of file [vx\\_types.h](#).

**#define VX\_SCALE\_UNITY (1024u)**

Use to indicate the 1:1 ratio in Q22.10 format.

Definition at line 1287 of file [vx\\_types.h](#).

**3.42.4 Typedef Documentation****typedef int32\_t vx\_enum**

Sets the standard enumeration type size to be a fixed quantity.

All enumerable fields must use this type as the container to enforce enumeration ranges and `sizeof()` operations.

Definition at line 141 of file [vx\\_types.h](#).

**typedef vx\_enum vx\_status**

A formal status type with known fixed size.

See also

[vx\\_status\\_e](#)

Definition at line 401 of file [vx\\_types.h](#).

### 3.42.5 Enumeration Type Documentation

#### enum vx\_bool

A Boolean value. This allows 0 to be FALSE, as it is in C, and any non-zero to be TRUE.

```
vx_bool ret = vx_true_e;
if (ret) printf("true!\n");
ret = vx_false_e;
if (!ret) printf("false!\n");
```

This would print both strings.

Enumerator

**vx\_false\_e** The "false" value.

**vx\_true\_e** The "true" value.

Definition at line 272 of file [vx\\_types.h](#).

#### enum vx\_type\_e

The type enumeration lists all the known types in OpenVX.

Enumerator

**VX\_TYPE\_INVALID** An invalid type value. When passed an error must be returned.

**VX\_TYPE\_CHAR** A [vx\\_char](#).

**VX\_TYPE\_INT8** A [vx\\_int8](#).

**VX\_TYPE\_UINT8** A [vx\\_uint8](#).

**VX\_TYPE\_INT16** A [vx\\_int16](#).

**VX\_TYPE\_UINT16** A [vx\\_uint16](#).

**VX\_TYPE\_INT32** A [vx\\_int32](#).

**VX\_TYPE\_UINT32** A [vx\\_uint32](#).

**VX\_TYPE\_INT64** A [vx\\_int64](#).

**VX\_TYPE\_UINT64** A [vx\\_uint64](#).

**VX\_TYPE\_FLOAT32** A [vx\\_float32](#).

**VX\_TYPE\_FLOAT64** A [vx\\_float64](#).

**VX\_TYPE\_ENUM** A [vx\\_enum](#). Equivalent in size to a [vx\\_int32](#).

**VX\_TYPE\_SIZE** A [vx\\_size](#).

**VX\_TYPE\_DF\_IMAGE** A [vx\\_df\\_image](#).

**VX\_TYPE\_BOOL** A [vx\\_bool](#).

**VX\_TYPE\_SCALAR\_MAX** A floating value for comparison between OpenVX scalars and OpenVX structs.

**VX\_TYPE\_RECTANGLE** A [vx\\_rectangle\\_t](#).

**VX\_TYPE\_KEYPOINT** A [vx\\_keypoint\\_t](#).

**VX\_TYPE\_COORDINATES2D** A [vx\\_coordinates2d\\_t](#).

**VX\_TYPE\_COORDINATES3D** A [vx\\_coordinates3d\\_t](#).

**VX\_TYPE\_USER\_STRUCT\_START** A floating value for user-defined struct base index.

**VX\_TYPE\_STRUCT\_MAX** A floating value for comparison between OpenVX structs and user structs.

**VX\_TYPE\_VENDOR\_STRUCT\_START** A floating value for vendor-defined struct base index.

**VX\_TYPE\_USER\_STRUCT\_END** A floating value for comparison between user structs and vendor structs.

**VX\_TYPE\_VENDOR\_STRUCT\_END** A floating value for comparison between vendor structs and OpenVX objects.

**VX\_TYPE\_REFERENCE** A [vx\\_reference](#).

**VX\_TYPE\_CONTEXT** A [vx\\_context](#).

**VX\_TYPE\_GRAPH** A [vx\\_graph](#).  
**VX\_TYPE\_NODE** A [vx\\_node](#).  
**VX\_TYPE\_KERNEL** A [vx\\_kernel](#).  
**VX\_TYPE\_PARAMETER** A [vx\\_parameter](#).  
**VX\_TYPE\_DELAY** A [vx\\_delay](#).  
**VX\_TYPE\_LUT** A [vx\\_lut](#).  
**VX\_TYPE\_DISTRIBUTION** A [vx\\_distribution](#).  
**VX\_TYPE\_PYRAMID** A [vx\\_pyramid](#).  
**VX\_TYPE\_THRESHOLD** A [vx\\_threshold](#).  
**VX\_TYPE\_MATRIX** A [vx\\_matrix](#).  
**VX\_TYPE\_CONVOLUTION** A [vx\\_convolution](#).  
**VX\_TYPE\_SCALAR** A [vx\\_scalar](#). when needed to be completely generic for kernel validation.  
**VX\_TYPE\_ARRAY** A [vx\\_array](#).  
**VX\_TYPE\_IMAGE** A [vx\\_image](#).  
**VX\_TYPE\_REMAP** A [vx\\_remap](#).  
**VX\_TYPE\_ERROR** An error object which has no type.  
**VX\_TYPE\_META\_FORMAT** A [vx\\_meta\\_format](#).  
**VX\_TYPE\_VENDOR\_OBJECT\_START** A floating value for vendor defined object base index.  
**VX\_TYPE\_OBJECT\_MAX** A value used for bound checking the OpenVX object types.  
**VX\_TYPE\_VENDOR\_OBJECT\_END** A value used for bound checking of vendor objects.

Definition at line 294 of file [vx\\_types.h](#).

#### enum vx\_status\_e

The enumeration of all status codes.

See also

[vx\\_status](#).

Enumerator

**VX\_STATUS\_MIN** Indicates the lower bound of status codes in VX. Used for bounds checks only.  
**VX\_ERROR\_REFERENCE\_NONZERO** Indicates that an operation did not complete due to a reference count being non-zero.  
**VX\_ERROR\_MULTIPLE\_WRITERS** Indicates that the graph has more than one node outputting to the same data object. This is an invalid graph structure.  
**VX\_ERROR\_GRAPH\_ABANDONED** Indicates that the graph is stopped due to an error or a callback that abandoned execution.  
**VX\_ERROR\_GRAPH\_SCHEDULED** Indicates that the supplied graph already has been scheduled and may be currently executing.  
**VX\_ERROR\_INVALID\_SCOPE** Indicates that the supplied parameter is from another scope and cannot be used in the current scope.  
**VX\_ERROR\_INVALID\_NODE** Indicates that the supplied node could not be created.  
**VX\_ERROR\_INVALID\_GRAPH** Indicates that the supplied graph has invalid connections (cycles).  
**VX\_ERROR\_INVALID\_TYPE** Indicates that the supplied type parameter is incorrect.  
**VX\_ERROR\_INVALID\_VALUE** Indicates that the supplied parameter has an incorrect value.  
**VX\_ERROR\_INVALID\_DIMENSION** Indicates that the supplied parameter is too big or too small in dimension.  
**VX\_ERROR\_INVALID\_FORMAT** Indicates that the supplied parameter is in an invalid format.

**VX\_ERROR\_INVALID\_LINK** Indicates that the link is not possible as specified. The parameters are incompatible.

**VX\_ERROR\_INVALID\_REFERENCE** Indicates that the reference provided is not valid.

**VX\_ERROR\_INVALID\_MODULE** This is returned from `vxLoadKernels` when the module does not contain the entry point.

**VX\_ERROR\_INVALID\_PARAMETERS** Indicates that the supplied parameter information does not match the kernel contract.

**VX\_ERROR\_OPTIMIZED\_AWAY** Indicates that the object referred to has been optimized out of existence.

**VX\_ERROR\_NO\_MEMORY** Indicates that an internal or implicit allocation failed. Typically catastrophic. After detection, deconstruct the context.

See also

[vxVerifyGraph](#).

**VX\_ERROR\_NO\_RESOURCES** Indicates that an internal or implicit resource can not be acquired (not memory). This is typically catastrophic. After detection, deconstruct the context.

See also

[vxVerifyGraph](#).

**VX\_ERROR\_NOT\_COMPATIBLE** Indicates that the attempt to link two parameters together failed due to type incompatibility.

**VX\_ERROR\_NOT\_ALLOCATED** Indicates to the system that the parameter must be allocated by the system.

**VX\_ERROR\_NOT\_SUFFICIENT** Indicates that the given graph has failed verification due to an insufficient number of required parameters, which cannot be automatically created. Typically this indicates required atomic parameters.

See also

[vxVerifyGraph](#).

**VX\_ERROR\_NOT\_SUPPORTED** Indicates that the requested set of parameters produce a configuration that cannot be supported. Refer to the supplied documentation on the configured kernels.

See also

[vx\\_kernel\\_e](#).

**VX\_ERROR\_NOT\_IMPLEMENTED** Indicates that the requested kernel is missing.

See also

[vx\\_kernel\\_e](#) [vxGetKernelByName](#).

**VX\_FAILURE** Indicates a generic error code, used when no other describes the error.

**VX\_SUCCESS** No error.

Definition at line 367 of file [vx\\_types.h](#).

#### enum vx\_enum\_e

The set of supported enumerations in OpenVX.

These can be extracted from enumerated values using [VX\\_ENUM\\_TYPE](#).

Enumerator

**VX\_ENUM\_DIRECTION** Parameter Direction.

**VX\_ENUM\_ACTION** Action Codes.

**VX\_ENUM\_HINT** Hint Values.

**VX\_ENUM\_DIRECTIVE** Directive Values.

**VX\_ENUM\_INTERPOLATION** Interpolation Types.

**VX\_ENUM\_OVERFLOW** Overflow Policies.



**VX\_ENUM\_COLOR\_SPACE** Color Space.  
**VX\_ENUM\_COLOR\_RANGE** Color Space Range.  
**VX\_ENUM\_PARAMETER\_STATE** Parameter State.  
**VX\_ENUM\_CHANNEL** Channel Name.  
**VX\_ENUM\_CONVERT\_POLICY** Convert Policy.  
**VX\_ENUM\_THRESHOLD\_TYPE** Threshold Type List.  
**VX\_ENUM\_BORDER\_MODE** Border Mode List.  
**VX\_ENUM\_COMPARISON** Comparison Values.  
**VX\_ENUM\_IMPORT\_MEM** The memory import enumeration.  
**VX\_ENUM\_TERM\_CRITERIA** A termination criteria.  
**VX\_ENUM\_NORM\_TYPE** A norm type.  
**VX\_ENUM\_ACCESSOR** An accessor flag type.  
**VX\_ENUM\_ROUND\_POLICY** Rounding Policy.

Definition at line 522 of file [vx\\_types.h](#).

#### **enum vx\_convert\_policy\_e**

The Conversion Policy Enumeration.

Enumerator

**VX\_CONVERT\_POLICY\_WRAP** Results are the least significant bits of the output operand, as if stored in two's complement binary format in the size of its bit-depth.  
**VX\_CONVERT\_POLICY\_SATURATE** Results are saturated to the bit depth of the output operand.

Definition at line 597 of file [vx\\_types.h](#).

#### **enum vx\_df\_image\_e**

Based on the VX\_DF\_IMAGE definition.

Note

Use [vx\\_df\\_image](#) to contain these values.

Enumerator

**VX\_DF\_IMAGE\_VIRT** A virtual image of no defined type.  
**VX\_DF\_IMAGE\_RGB** A single plane of 24-bit pixel as 3 interleaved 8-bit units of R then G then B data. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_RGBX** A single plane of 32-bit pixel as 4 interleaved 8-bit units of R then G then B data, then a *don't care* byte. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_NV12** A 2-plane YUV format of Luma (Y) and interleaved UV data at 4:2:0 sampling. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_NV21** A 2-lane YUV format of Luma (Y) and interleaved VU data at 4:2:0 sampling. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_UYVY** A single plane of 32-bit macro pixel of U0, Y0, V0, Y1 bytes. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_YUYV** A single plane of 32-bit macro pixel of Y0, U0, Y1, V0 bytes. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_IYUV** A 3 plane of 8-bit 4:2:0 sampled Y, U, V planes. This uses the BT709 full range by default.  
**VX\_DF\_IMAGE\_YUV4** A 3 plane of 8 bit 4:4:4 sampled Y, U, V planes. This uses the BT709 full range by default.

**VX\_DF\_IMAGE\_U8** A single plane of unsigned 8-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

**VX\_DF\_IMAGE\_U16** A single plane of unsigned 16-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

**VX\_DF\_IMAGE\_S16** A single plane of signed 16-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

**VX\_DF\_IMAGE\_U32** A single plane of unsigned 32-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

**VX\_DF\_IMAGE\_S32** A single plane of unsigned 32-bit data. The range of data is not specified, as it may be extracted from a YUV or generated.

Definition at line 610 of file [vx\\_types.h](#).

#### enum vx\_channel\_e

The channel enumerations for channel extractions.

See also

[vxChannelExtractNode](#)  
[vxuChannelExtract](#)  
[VX\\_KERNEL\\_CHANNEL\\_EXTRACT](#)

Enumerator

**VX\_CHANNEL\_0** Used by formats with unknown channel types.

**VX\_CHANNEL\_1** Used by formats with unknown channel types.

**VX\_CHANNEL\_2** Used by formats with unknown channel types.

**VX\_CHANNEL\_3** Used by formats with unknown channel types.

**VX\_CHANNEL\_R** Use to extract the RED channel, no matter the byte or packing order.

**VX\_CHANNEL\_G** Use to extract the GREEN channel, no matter the byte or packing order.

**VX\_CHANNEL\_B** Use to extract the BLUE channel, no matter the byte or packing order.

**VX\_CHANNEL\_A** Use to extract the ALPHA channel, no matter the byte or packing order.

**VX\_CHANNEL\_Y** Use to extract the LUMA channel, no matter the byte or packing order.

**VX\_CHANNEL\_U** Use to extract the Cb/U channel, no matter the byte or packing order.

**VX\_CHANNEL\_V** Use to extract the Cr/V/Value channel, no matter the byte or packing order.

Definition at line 989 of file [vx\\_types.h](#).

#### enum vx\_interpolation\_type\_e

The image reconstruction filters supported by image resampling operations.

The edge of a pixel is interpreted as being aligned to the edge of the image. The value for an output pixel is evaluated at the center of that pixel.

This means, for example, that an even enlargement of a factor of two in nearest-neighbor interpolation will replicate every source pixel into a 2x2 quad in the destination, and that an even shrink by a factor of two in bilinear interpolation will create each destination pixel by average a 2x2 quad of source pixels.

Samples that cross the boundary of the source image have values determined by the border mode - see [vx↵\\_border\\_mode\\_e](#) and [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#).

See also

[vxuScaleImage](#)  
[vxScaleImageNode](#)  
[VX\\_KERNEL\\_SCALE\\_IMAGE](#)  
[vxuWarpAffine](#)  
[vxWarpAffineNode](#)  
[VX\\_KERNEL\\_WARP\\_AFFINE](#)  
[vxuWarpPerspective](#)  
[vxWarpPerspectiveNode](#)  
[VX\\_KERNEL\\_WARP\\_PERSPECTIVE](#)

Enumerator

**VX\_INTERPOLATION\_TYPE\_NEAREST\_NEIGHBOR** Output values are defined to match the source pixel whose center is nearest to the sample position.

**VX\_INTERPOLATION\_TYPE\_BILINEAR** Output values are defined by bilinear interpolation between the pixels whose centers are closest to the sample position, weighted linearly by the distance of the sample from the pixel centers.

**VX\_INTERPOLATION\_TYPE\_AREA** Output values are determined by averaging the source pixels whose areas fall under the area of the destination pixel, projected onto the source image.

Definition at line 1049 of file [vx\\_types.h](#).

#### **enum vx\_vendor\_id\_e**

The Vendor ID of the Implementation. As new vendors submit their implementations, this enumeration will grow.

Enumerator

**VX\_ID\_KHRONOS** The Khronos Group.

**VX\_ID\_TI** Texas Instruments, Inc.

**VX\_ID\_QUALCOMM** Qualcomm, Inc.

**VX\_ID\_NVIDIA** NVIDIA Corporation.

**VX\_ID\_ARM** ARM Ltd.

**VX\_ID\_BDTI** Berkley Design Technology, Inc.

**VX\_ID\_RENESAS** Renesas Electronics.

**VX\_ID\_VIVANTE** Vivante Corporation.

**VX\_ID\_XILINX** Xilinx Inc.

**VX\_ID\_AXIS** Axis Communications.

**VX\_ID\_MOVIDIUS** Movidius Ltd.

**VX\_ID\_SAMSUNG** Samsung Electronics.

**VX\_ID\_FREESCALE** Freescale Semiconductor.

**VX\_ID\_AMD** Advanced Micro Devices.

**VX\_ID\_BROADCOM** Broadcom Corporation.

**VX\_ID\_INTEL** Intel Corporation.

**VX\_ID\_MARVELL** Marvell Technology Group Ltd.

**VX\_ID\_MEDIATEK** MediaTek, Inc.

**VX\_ID\_ST** STMicroelectronics.

**VX\_ID\_CEVA** CEVA DSP.

**VX\_ID\_ITSEEZ** Itseez, Inc.

**VX\_ID\_IMAGINATION** Imagination Technologies.

**VX\_ID\_COGNIVUE** CogniVue Corporation.

**VX\_ID\_VIDEANTIS** Videantis.

**VX\_ID\_SYNOPSYS** Synopsys.

**VX\_ID\_DEFAULT** For use by all Kernel authors until they can obtain an assigned ID.

Definition at line 36 of file [vx\\_vendors.h](#).

### 3.42.6 Function Documentation

#### **vx\_status VX\_API\_CALL vxGetStatus ( vx\_reference *reference* )**

Provides a generic API to return status values from Object constructors if they fail.

##### Note

Users do not need to strictly check every object creator as the errors should properly propagate and be detected during verification time or run-time.

```
vx_image img = vxCreateImage(context, 639, 480,
 VX_DF_IMAGE_UYVY);
vx_status status = vxGetStatus((vx_reference)img);
// status == VX_ERROR_INVALID_DIMENSIONS
vxReleaseImage(&img);
```

##### Precondition

Appropriate Object Creator function.

##### Postcondition

Appropriate Object Release function.

##### Parameters

|    |                  |                                                 |
|----|------------------|-------------------------------------------------|
| in | <i>reference</i> | The reference to check for construction errors. |
|----|------------------|-------------------------------------------------|

##### Returns

A `vx_status_e` enumeration.

##### Return values

|                         |                                                                     |
|-------------------------|---------------------------------------------------------------------|
| <code>VX_SUCCESS</code> | No error.                                                           |
| *                       | Some error occurred, please check enumeration list and constructor. |

## 3.43 Objects

### 3.43.1 Detailed Description

Defines the basic objects within OpenVX.

All objects in OpenVX derive from a [vx\\_reference](#) and contain a reference to the [vx\\_context](#) from which they were made, except the [vx\\_context](#) itself.

#### Modules

- [Object: Reference](#)  
*Defines the Reference Object interface.*
- [Object: Context](#)  
*Defines the Context Object Interface.*
- [Object: Graph](#)  
*Defines the Graph Object interface.*
- [Object: Node](#)  
*Defines the Node Object interface.*
- [Object: Array](#)  
*Defines the Array Object Interface.*
- [Object: Convolution](#)  
*Defines the Image Convolution Object interface.*
- [Object: Distribution](#)  
*Defines the Distribution Object Interface.*
- [Object: Image](#)  
*Defines the Image Object interface.*
- [Object: LUT](#)  
*Defines the Look-Up Table Interface.*
- [Object: Matrix](#)  
*Defines the Matrix Object Interface.*
- [Object: Pyramid](#)  
*Defines the Image Pyramid Object Interface.*
- [Object: Remap](#)  
*Defines the Remap Object Interface.*
- [Object: Scalar](#)  
*Defines the Scalar Object interface.*
- [Object: Threshold](#)  
*Defines the Threshold Object Interface.*

## 3.44 Object: Reference

### 3.44.1 Detailed Description

Defines the Reference Object interface.

All objects in OpenVX are derived (in the object-oriented sense) from [vx\\_reference](#). All objects shall be able to be cast back to this type safely.

#### Typedefs

- typedef struct \_vx\_reference \* [vx\\_reference](#)  
A generic opaque reference to any object within OpenVX.

#### Enumerations

- enum [vx\\_reference\\_attribute\\_e](#) {  
[VX\\_REF\\_ATTRIBUTE\\_COUNT](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REFERENCE](#) << 8)) + 0x0,  
[VX\\_REF\\_ATTRIBUTE\\_TYPE](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REFERENCE](#) << 8)) + 0x1 }  
The reference attributes list.

#### Functions

- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxQueryReference](#) ([vx\\_reference](#) ref, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
Queries any reference type for some basic information (count, type).

### 3.44.2 Typedef Documentation

**typedef struct \_vx\_reference\* vx\_reference**

A generic opaque reference to any object within OpenVX.

A user of OpenVX should not assume that this can be cast directly to anything; however, any object in OpenVX can be cast back to this for the purposes of querying attributes of the object or for passing the object as a parameter to functions that take a [vx\\_reference](#) type. If the API does not take that specific type but may take others, an error may be returned from the API.

Definition at line 134 of file [vx\\_types.h](#).

### 3.44.3 Enumeration Type Documentation

**enum vx\_reference\_attribute\_e**

The reference attributes list.

Enumerator

**VX\_REF\_ATTRIBUTE\_COUNT** Returns the reference count of the object. Use a [vx\\_uint32](#) parameter.

**VX\_REF\_ATTRIBUTE\_TYPE** Returns the [vx\\_type\\_e](#) of the reference. Use a [vx\\_enum](#) parameter.

Definition at line 676 of file [vx\\_types.h](#).

### 3.44.4 Function Documentation

**vx\_status** [VX\\_API\\_CALL](#) [vxQueryReference](#) ( [vx\\_reference](#) ref, [vx\\_enum](#) attribute, void \* ptr, [vx\\_size](#) size )

Queries any reference type for some basic information (count, type).

**Parameters**

|     |                  |                                                                              |
|-----|------------------|------------------------------------------------------------------------------|
| in  | <i>ref</i>       | The reference to query.                                                      |
| in  | <i>attribute</i> | The value for which to query. Use <a href="#">vx_reference_attribute_e</a> . |
| out | <i>ptr</i>       | The location at which to store the resulting value.                          |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.               |

**Returns**

A [vx\\_status\\_e](#) enumeration.

## 3.45 Object: Context

### 3.45.1 Detailed Description

Defines the Context Object Interface.

The OpenVX context is the object domain for all OpenVX objects. All data objects *live* in the context as well as all framework objects. The OpenVX context keeps reference counts on all objects and must do garbage collection during its deconstruction to free lost references. While multiple clients may connect to the OpenVX context, all data are private in that the references referring to data objects are given only to the creating party.

### Macros

- `#define VX_MAX_IMPLEMENTATION_NAME (64)`  
*Defines the maximum number of characters in a implementation string.*

### Typedefs

- `typedef struct _vx_context * vx_context`  
*An opaque reference to the implementation context.*

### Enumerations

- `enum vx_accessor_e {`  
`VX_READ_ONLY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x1,`  
`VX_WRITE_ONLY = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x2,`  
`VX_READ_AND_WRITE = ((( VX_ID_KHRONOS ) << 20) | ( VX_ENUM_ACCESSOR << 12)) + 0x3 }`  
*The memory accessor hint flags. These enumeration values are used to indicate desired system behavior, not the User intent. For example: these can be interpreted as hints to the system about cache operations or marshalling operations.*
- `enum vx_context_attribute_e {`  
`VX_CONTEXT_ATTRIBUTE_VENDOR_ID = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x0,`  
`VX_CONTEXT_ATTRIBUTE_VERSION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8))`  
`+ 0x1,`  
`VX_CONTEXT_ATTRIBUTE_UNIQUE_KERNELS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x2,`  
`VX_CONTEXT_ATTRIBUTE_MODULES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x3,`  
`VX_CONTEXT_ATTRIBUTE_REFERENCES = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x4,`  
`VX_CONTEXT_ATTRIBUTE_IMPLEMENTATION = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x5,`  
`VX_CONTEXT_ATTRIBUTE_EXTENSIONS_SIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x6,`  
`VX_CONTEXT_ATTRIBUTE_EXTENSIONS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0x7,`  
`VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION = ((( VX_ID_KHRONOS ) << 20)`  
`| ( VX_TYPE_CONTEXT << 8)) + 0x8,`  
`VX_CONTEXT_ATTRIBUTE_OPTICAL_FLOW_WINDOW_MAXIMUM_DIMENSION = ((( VX_ID_KHRONOS ) << 20)`  
`| ( VX_TYPE_CONTEXT << 8)) + 0x9,`  
`VX_CONTEXT_ATTRIBUTE_IMMEDIATE_BORDER_MODE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xA,`  
`VX_CONTEXT_ATTRIBUTE_UNIQUE_KERNEL_TABLE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_CONTEXT << 8)) + 0xB }`  
*A list of context attributes.*



- enum `vx_import_type_e` {  
`VX_IMPORT_TYPE_NONE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_IMPORT_MEM` << 12)) + 0x0,  
`VX_IMPORT_TYPE_HOST` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_IMPORT_MEM` << 12)) + 0x1  
}

*An enumeration of memory import types.*

- enum `vx_round_policy_e` {  
`VX_ROUND_POLICY_TO_ZERO` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_ROUND_POLICY` << 12)) + 0x1,  
`VX_ROUND_POLICY_TO_NEAREST_EVEN` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_ROUND_POLICY` << 12)) + 0x2 }

*The Round Policy Enumeration.*

- enum `vx_termination_criteria_e` {  
`VX_TERM_CRITERIA_ITERATIONS` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_TERM_CRITERIA` << 12)) + 0x0,  
`VX_TERM_CRITERIA_EPSILON` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_TERM_CRITERIA` << 12)) + 0x1,  
`VX_TERM_CRITERIA_BOTH` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_TERM_CRITERIA` << 12)) + 0x2 }

*The termination criteria list.*

## Functions

- `vx_context` `VX_API_CALL vxCreateContext` ()  
*Creates a `vx_context`.*
- `vx_context` `VX_API_CALL vxGetContext` (`vx_reference` reference)  
*Retrieves the context from any reference from within a context.*
- `vx_status` `VX_API_CALL vxQueryContext` (`vx_context` context, `vx_enum` attribute, void \*ptr, `vx_size` size)  
*Queries the context for some specific information.*
- `vx_status` `VX_API_CALL vxReleaseContext` (`vx_context` \*context)  
*Releases the OpenVX object context.*
- `vx_status` `VX_API_CALL vxSetContextAttribute` (`vx_context` context, `vx_enum` attribute, const void \*ptr, `vx_size` size)  
*Sets an attribute on the context.*

### 3.45.2 Typedef Documentation

**typedef struct \_vx\_context\* vx\_context**

An opaque reference to the implementation context.

See also

[vxCreateContext](#)

Definition at line 202 of file [vx\\_types.h](#).

### 3.45.3 Enumeration Type Documentation

**enum vx\_context\_attribute\_e**

A list of context attributes.

Enumerator

**`VX_CONTEXT_ATTRIBUTE_VENDOR_ID`** Queries the unique vendor ID. Use a `vx_uint16`.

**`VX_CONTEXT_ATTRIBUTE_VERSION`** Queries the OpenVX Version Number. Use a `vx_uint16`

**`VX_CONTEXT_ATTRIBUTE_UNIQUE_KERNELS`** Queries the context for the number of *unique* kernels. Use a `vx_uint32` parameter.

**VX\_CONTEXT\_ATTRIBUTE\_MODULES** Queries the context for the number of active modules. Use a [vx\\_uint32](#) parameter.

**VX\_CONTEXT\_ATTRIBUTE\_REFERENCES** Queries the context for the number of active references. Use a [vx\\_uint32](#) parameter.

**VX\_CONTEXT\_ATTRIBUTE\_IMPLEMENTATION** Queries the context for its implementation name. Use a [vx\\_char\[VX\\_MAX\\_IMPLEMENTATION\\_NAME\]](#) array.

**VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS\_SIZE** Queries the number of bytes in the extensions string. Use a [vx\\_size](#) parameter.

**VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS** Retrieves the extensions string. This is a space-separated string of extension names. Use a [vx\\_char](#) pointer allocated to the size returned from [VX\\_CONTEXT\\_ATTRIBUTE\\_EXTENSIONS\\_SIZE](#).

**VX\_CONTEXT\_ATTRIBUTE\_CONVOLUTION\_MAXIMUM\_DIMENSION** The maximum width or height of a convolution matrix. Use a [vx\\_size](#) parameter. Each vendor must support centered kernels of size  $w \times h$ , where both  $w$  and  $h$  are odd numbers,  $3 \leq w \leq n$  and  $3 \leq h \leq n$ , where  $n$  is the value of the [VX\\_CONTEXT\\_ATTRIBUTE\\_CONVOLUTION\\_MAXIMUM\\_DIMENSION](#) attribute.  $n$  is an odd number that should not be smaller than 9.  $w$  and  $h$  may or may not be equal to each other. All combinations of  $w$  and  $h$  meeting the conditions above must be supported. The behavior of [vxCreateConvolution](#) is undefined for values larger than the value returned by this attribute.

**VX\_CONTEXT\_ATTRIBUTE\_OPTICAL\_FLOW\_WINDOW\_MAXIMUM\_DIMENSION** The maximum window dimension of the OpticalFlowPyrLK kernel.

See also

[VX\\_KERNEL\\_OPTICAL\\_FLOW\\_PYR\\_LK](#). Use a [vx\\_size](#) parameter.

**VX\_CONTEXT\_ATTRIBUTE\_IMMEDIATE\_BORDER\_MODE** The border mode for immediate mode functions. Graph mode functions are unaffected by this attribute. Use a pointer to a [vx\\_border\\_mode\\_t](#) structure as parameter.

Note

The assumed default value for immediate mode functions is [VX\\_BORDER\\_MODE\\_UNDEFINED](#).

**VX\_CONTEXT\_ATTRIBUTE\_UNIQUE\_KERNEL\_TABLE** Returns the table of all unique the kernels that exist in the context. Use a [vx\\_kernel\\_info\\_t](#) array.

Precondition

You must call [vxQueryContext](#) with [VX\\_CONTEXT\\_ATTRIBUTE\\_UNIQUE\\_KERNELS](#) to compute the necessary size of the array.

Definition at line [686](#) of file [vx\\_types.h](#).

## enum vx\_import\_type\_e

An enumeration of memory import types.

Enumerator

**VX\_IMPORT\_TYPE\_NONE** For memory allocated through OpenVX, this is the import type.

**VX\_IMPORT\_TYPE\_HOST** The default memory type to import from the Host.

Definition at line [1018](#) of file [vx\\_types.h](#).

## enum vx\_termination\_criteria\_e

The termination criteria list.

See also

[Optical Flow Pyramid \(LK\)](#)

Enumerator

**VX\_TERM\_CRITERIA\_ITERATIONS** Indicates a termination after a set number of iterations.

**VX\_TERM\_CRITERIA\_EPSILON** Indicates a termination after matching against the value of epsilon provided to the function.

**VX\_TERM\_CRITERIA\_BOTH** Indicates that both an iterations and epsilon method are employed. Whichever one matches first causes the termination.

Definition at line 1121 of file [vx\\_types.h](#).

#### enum vx\_accessor\_e

The memory accessor hint flags. These enumeration values are used to indicate desired *system* behavior, not the **User** intent. For example: these can be interpreted as hints to the system about cache operations or marshalling operations.

Enumerator

**VX\_READ\_ONLY** The memory shall be treated by the system as if it were read-only. If the User writes to this memory, the results are implementation defined.

**VX\_WRITE\_ONLY** The memory shall be treated by the system as if it were write-only. If the User reads from this memory, the results are implementation defined.

**VX\_READ\_AND\_WRITE** The memory shall be treated by the system as if it were readable and writeable.

Definition at line 1159 of file [vx\\_types.h](#).

#### enum vx\_round\_policy\_e

The Round Policy Enumeration.

Enumerator

**VX\_ROUND\_POLICY\_TO\_ZERO** When scaling, this truncates the least significant values that are lost in operations.

**VX\_ROUND\_POLICY\_TO\_NEAREST\_EVEN** When scaling, this rounds to nearest even output value.

Definition at line 1176 of file [vx\\_types.h](#).

### 3.45.4 Function Documentation

#### vx\_context VX\_API\_CALL vxCreateContext ( )

Creates a [vx\\_context](#).

This creates a top-level object context for OpenVX.

Note

This is required to do anything else.

Returns

The reference to the implementation context [vx\\_context](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

Postcondition

[vxReleaseContext](#)

**vx\_status VX\_API\_CALL vxReleaseContext ( vx\_context \* *context* )**

Releases the OpenVX object context.

All reference counted objects are garbage-collected by the return of this call. No calls are possible using the parameter context after the context has been released until a new reference from [vxCreateContext](#) is returned. All outstanding references to OpenVX objects from this context are invalid after this call.

**Parameters**

|           |                |                                              |
|-----------|----------------|----------------------------------------------|
| <i>in</i> | <i>context</i> | The pointer to the reference to the context. |
|-----------|----------------|----------------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                         |                                               |
|-----------------------------------------|-----------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                    |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If context is not a <code>vx_context</code> . |

**Precondition**

`vxCreateContext`

**`vx_context VX_API_CALL vxGetContext ( vx_reference reference )`**

Retrieves the context from any reference from within a context.

**Parameters**

|           |                  |                                                  |
|-----------|------------------|--------------------------------------------------|
| <i>in</i> | <i>reference</i> | The reference from which to extract the context. |
|-----------|------------------|--------------------------------------------------|

**Returns**

The overall context that created the particular reference.

**`vx_status VX_API_CALL vxQueryContext ( vx_context context, vx_enum attribute, void * ptr, vx_size size )`**

Queries the context for some specific information.

**Parameters**

|            |                  |                                                                     |
|------------|------------------|---------------------------------------------------------------------|
| <i>in</i>  | <i>context</i>   | The reference to the context.                                       |
| <i>in</i>  | <i>attribute</i> | The attribute to query. Use a <code>vx_context_attribute_e</code> . |
| <i>out</i> | <i>ptr</i>       | The location at which to store the resulting value.                 |
| <i>in</i>  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.      |

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                          |                                                   |
|------------------------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code>                  | No errors.                                        |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the context is not a <code>vx_context</code> . |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If any of the other parameters are incorrect.     |

|                                     |                                                           |
|-------------------------------------|-----------------------------------------------------------|
| <code>VX_ERROR_NOT_SUPPORTED</code> | If the attribute is not supported on this implementation. |
|-------------------------------------|-----------------------------------------------------------|

**vx\_status VX\_API\_CALL vxSetContextAttribute ( vx\_context *context*, vx\_enum *attribute*, const void \* *ptr*, vx\_size *size* )**

Sets an attribute on the context.

Parameters

|    |                  |                                                                    |
|----|------------------|--------------------------------------------------------------------|
| in | <i>context</i>   | The handle to the overall context.                                 |
| in | <i>attribute</i> | The attribute to set from <a href="#">vx_context_attribute_e</a> . |
| in | <i>ptr</i>       | The pointer to the data to which to set the attribute.             |
| in | <i>size</i>      | The size in bytes of the data to which <i>ptr</i> points.          |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                                          |                                                      |
|------------------------------------------|------------------------------------------------------|
| <code>VX_SUCCESS</code>                  | No errors.                                           |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the context is not a <a href="#">vx_context</a> . |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If any of the other parameters are incorrect.        |
| <code>VX_ERROR_NOT_SUPPORTED</code>      | If the attribute is not settable.                    |

## 3.46 Object: Graph

### 3.46.1 Detailed Description

Defines the Graph Object interface.

A set of nodes connected in a directed (only goes one-way) acyclic (does not loop back) fashion. A Graph may have sets of Nodes that are unconnected to other sets of Nodes within the same Graph. See [Graph Formalisms](#).

#### Typedefs

- typedef struct \_vx\_graph \* vx\_graph  
*An opaque reference to a graph.*

#### Enumerations

- enum vx\_graph\_attribute\_e {  
 $VX\_GRAPH\_ATTRIBUTE\_NUMNODES = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_GRAPH << 8)) + 0x0,$   
 $VX\_GRAPH\_ATTRIBUTE\_STATUS = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_GRAPH << 8)) + 0x1,$   
 $VX\_GRAPH\_ATTRIBUTE\_PERFORMANCE = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_GRAPH << 8)) + 0x2,$   
 $VX\_GRAPH\_ATTRIBUTE\_NUMPARAMETERS = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_GRAPH << 8)) + 0x3 }$   
*The graph attributes list.*

#### Functions

- vx\_graph VX\_API\_CALL vxCreateGraph (vx\_context context)  
*Creates an empty graph.*
- vx\_bool VX\_API\_CALL vxIsGraphVerified (vx\_graph graph)  
*Returns a Boolean to indicate the state of graph verification.*
- vx\_status VX\_API\_CALL vxProcessGraph (vx\_graph graph)  
*This function causes the synchronous processing of a graph. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what vxVerifyGraph would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via vxVerifyGraph or vxProcessGraph then the graph is processed. This function blocks until the graph is completed.*
- vx\_status VX\_API\_CALL vxQueryGraph (vx\_graph graph, vx\_enum attribute, void \*ptr, vx\_size size)  
*Allows the user to query attributes of the Graph.*
- vx\_status VX\_API\_CALL vxReleaseGraph (vx\_graph \*graph)  
*Releases a reference to a graph. The object may not be garbage collected until its total reference count is zero. Once the reference count is zero, all node references in the graph are automatically released as well. Data referenced by those nodes may not be released as the user may have external references to the data.*
- vx\_status VX\_API\_CALL vxScheduleGraph (vx\_graph graph)  
*Schedules a graph for future execution.*
- vx\_status VX\_API\_CALL vxSetGraphAttribute (vx\_graph graph, vx\_enum attribute, const void \*ptr, vx\_size size)  
*Allows the set to attributes on the Graph.*
- vx\_status VX\_API\_CALL vxVerifyGraph (vx\_graph graph)  
*Verifies the state of the graph before it is executed. This is useful to catch programmer errors and contract errors. If not verified, the graph verifies before being processed.*
- vx\_status VX\_API\_CALL vxWaitGraph (vx\_graph graph)  
*Waits for a specific graph to complete. If the graph has been scheduled multiple times since the last call to vxWaitGraph, then vxWaitGraph returns only when the last scheduled execution completes.*

### 3.46.2 Typedef Documentation

**typedef struct \_vx\_graph\* vx\_graph**

An opaque reference to a graph.

See also

[vxCreateGraph](#)

Definition at line 195 of file [vx\\_types.h](#).

### 3.46.3 Enumeration Type Documentation

**enum vx\_graph\_attribute\_e**

The graph attributes list.

Enumerator

**VX\_GRAPH\_ATTRIBUTE\_NUMNODES** Returns the number of nodes in a graph. Use a [vx\\_uint32](#) parameter.

**VX\_GRAPH\_ATTRIBUTE\_STATUS** Returns the overall status of the graph. Use a [vx\\_status](#) parameter.

**VX\_GRAPH\_ATTRIBUTE\_PERFORMANCE** Returns the overall performance of the graph. Use a [vx\\_perf\\_t](#) parameter.

**VX\_GRAPH\_ATTRIBUTE\_NUMPARAMETERS** Returns the number of explicitly declared parameters on the graph. Use a [vx\\_uint32](#) parameter.

Definition at line 828 of file [vx\\_types.h](#).

### 3.46.4 Function Documentation

**vx\_graph VX\_API\_CALL vxCreateGraph ( vx\_context context )**

Creates an empty graph.

Parameters

|    |         |                                              |
|----|---------|----------------------------------------------|
| in | context | The reference to the implementation context. |
|----|---------|----------------------------------------------|

Returns

A graph reference [vx\\_graph](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_status VX\_API\_CALL vxReleaseGraph ( vx\_graph \* graph )**

Releases a reference to a graph. The object may not be garbage collected until its total reference count is zero. Once the reference count is zero, all node references in the graph are automatically released as well. Data referenced by those nodes may not be released as the user may have external references to the data.

Parameters

|    |       |                                      |
|----|-------|--------------------------------------|
| in | graph | The pointer to the graph to release. |
|----|-------|--------------------------------------|

Postcondition

After returning from this function the reference is zeroed.

Returns

A [vx\\_status\\_e](#) enumeration.



## Return values

|                                         |                                              |
|-----------------------------------------|----------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                   |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If graph is not a <a href="#">vx_graph</a> . |

**vx\_status VX\_API\_CALL vxVerifyGraph ( vx\_graph graph )**

Verifies the state of the graph before it is executed. This is useful to catch programmer errors and contract errors. If not verified, the graph verifies before being processed.

## Precondition

Memory for data objects is not guaranteed to exist before this call.

## Postcondition

After this call data objects exist unless the implementation optimized them out.

## Parameters

|    |              |                                       |
|----|--------------|---------------------------------------|
| in | <i>graph</i> | The reference to the graph to verify. |
|----|--------------|---------------------------------------|

## Returns

A status code for graphs with more than one error; it is undefined which error will be returned. Register a log callback using [vxRegisterLogCallback](#) to receive each specific error in the graph.

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                         |                                                                  |
|-----------------------------------------|------------------------------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                                       |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If graph is not a <a href="#">vx_graph</a> .                     |
| <code>VX_ERROR_MULTIPLE_WRITERS</code>  | If the graph contains more than one writer to any data object.   |
| <code>VX_ERROR_INVALID_NODE</code>      | If a node in the graph is invalid or failed to be created.       |
| <code>VX_ERROR_INVALID_GRAPH</code>     | If the graph contains cycles or some other invalid topology.     |
| <code>VX_ERROR_INVALID_TYPE</code>      | If any parameter on a node is given the wrong type.              |
| <code>VX_ERROR_INVALID_VALUE</code>     | If any value of any parameter is out of bounds of specification. |
| <code>VX_ERROR_INVALID_FORMAT</code>    | If the image format is not compatible.                           |

## See also

[vxProcessGraph](#)

**vx\_status VX\_API\_CALL vxProcessGraph ( vx\_graph graph )**

This function causes the synchronous processing of a graph. If the graph has not been verified, then the implementation verifies the graph immediately. If verification fails this function returns a status identical to what [vxVerifyGraph](#) would return. After the graph verifies successfully then processing occurs. If the graph was previously verified via [vxVerifyGraph](#) or [vxProcessGraph](#) then the graph is processed. This function blocks until the graph is completed.

**Parameters**

|           |              |                       |
|-----------|--------------|-----------------------|
| <i>in</i> | <i>graph</i> | The graph to execute. |
|-----------|--------------|-----------------------|

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                         |                                                  |
|-------------------------|--------------------------------------------------|
| <code>VX_SUCCESS</code> | Graph has been processed.                        |
| <code>VX_FAILURE</code> | A catastrophic error occurred during processing. |
| *                       | See <code>vxVerifyGraph</code> .                 |

**Precondition**

`vxVerifyGraph` must return `VX_SUCCESS` before this function will pass.

**See also**

`vxVerifyGraph`

**`vx_status VX_API_CALL vxScheduleGraph ( vx_graph graph )`**

Schedules a graph for future execution.

**Parameters**

|           |              |                        |
|-----------|--------------|------------------------|
| <i>in</i> | <i>graph</i> | The graph to schedule. |
|-----------|--------------|------------------------|

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                      |                                                               |
|--------------------------------------|---------------------------------------------------------------|
| <code>VX_ERROR_NO_RESOURCES</code>   | The graph cannot be scheduled now.                            |
| <code>VX_ERROR_NOT_SUFFICIENT</code> | The graph is not verified and has failed forced verification. |
| <code>VX_SUCCESS</code>              | The graph has been scheduled.                                 |

**Precondition**

`vxVerifyGraph` must return `VX_SUCCESS` before this function will pass.

**`vx_status VX_API_CALL vxWaitGraph ( vx_graph graph )`**

Waits for a specific graph to complete. If the graph has been scheduled multiple times since the last call to `vxWaitGraph`, then `vxWaitGraph` returns only when the last scheduled execution completes.

**Parameters**

|           |              |                       |
|-----------|--------------|-----------------------|
| <i>in</i> | <i>graph</i> | The graph to wait on. |
|-----------|--------------|-----------------------|

**Returns**

A `vx_status_e` enumeration.

## Return values

|                   |                                                                                                                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>VX_SUCCESS</i> | The graph has successfully completed execution and its outputs are the valid results of the most recent execution.                                                                                       |
| <i>VX_FAILURE</i> | An error occurred or the graph was never scheduled. Use <a href="#">vxQueryGraph</a> for the <i>VX_GRAPH_ATTRIBUTE_STATUS</i> attribute to determine the details. Output data of the graph is undefined. |

## Precondition

[vxScheduleGraph](#)

**vx\_status VX\_API\_CALL vxQueryGraph ( vx\_graph graph, vx\_enum attribute, void \* ptr, vx\_size size )**

Allows the user to query attributes of the Graph.

## Parameters

|     |                  |                                                                |
|-----|------------------|----------------------------------------------------------------|
| in  | <i>graph</i>     | The reference to the created graph.                            |
| in  | <i>attribute</i> | The <a href="#">vx_graph_attribute_e</a> type needed.          |
| out | <i>ptr</i>       | The location at which to store the resulting value.            |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points. |

## Returns

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxSetGraphAttribute ( vx\_graph graph, vx\_enum attribute, const void \* ptr, vx\_size size )**

Allows the set to attributes on the Graph.

## Parameters

|    |                  |                                                                |
|----|------------------|----------------------------------------------------------------|
| in | <i>graph</i>     | The reference to the graph.                                    |
| in | <i>attribute</i> | The <a href="#">vx_graph_attribute_e</a> type needed.          |
| in | <i>ptr</i>       | The location from which to read the value.                     |
| in | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points. |

## Returns

A [vx\\_status\\_e](#) enumeration.

**vx\_bool VX\_API\_CALL vxIsGraphVerified ( vx\_graph graph )**

Returns a Boolean to indicate the state of graph verification.

## Parameters

|    |              |                                      |
|----|--------------|--------------------------------------|
| in | <i>graph</i> | The reference to the graph to check. |
|----|--------------|--------------------------------------|

## Returns

A [vx\\_bool](#) value.

## Return values

|                  |                        |
|------------------|------------------------|
| <i>vx_true_e</i> | The graph is verified. |
|------------------|------------------------|

|                         |                                                                                                                                                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>vx_false_e</code> | The graph is not verified. It must be verified before execution either through <code>vxVerifyGraph</code> or automatically through <code>vxProcessGraph</code> or <code>vxScheduleGraph</code> . |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 3.47 Object: Node

### 3.47.1 Detailed Description

Defines the Node Object interface.

A node is an instance of a kernel that will be paired with a specific set of references (the parameters). Nodes are created from and associated with a single graph only. When a `vx_parameter` is extracted from a Node, an additional attribute can be accessed:

- *Reference* - The `vx_reference` assigned to this parameter index from the Node creation function (e.g., `vxSobel13x3Node`).

### Typedefs

- typedef struct \_vx\_node \* `vx_node`  
*An opaque reference to a kernel node.*

### Enumerations

- enum `vx_node_attribute_e` {  
`VX_NODE_ATTRIBUTE_STATUS` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_NODE` << 8)) + 0x0,  
`VX_NODE_ATTRIBUTE_PERFORMANCE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_NODE` << 8))  
+ 0x1,  
`VX_NODE_ATTRIBUTE_BORDER_MODE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_NODE` << 8))  
+ 0x2,  
`VX_NODE_ATTRIBUTE_LOCAL_DATA_SIZE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_NODE` << 8)) + 0x3,  
`VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_NODE` << 8)) + 0x4 }

*The node attributes list.*

### Functions

- `vx_status` `VX_API_CALL vxQueryNode` (`vx_node` node, `vx_enum` attribute, void \*ptr, `vx_size` size)  
*Allows a user to query information out of a node.*
- `vx_status` `VX_API_CALL vxReleaseNode` (`vx_node` \*node)  
*Releases a reference to a Node object. The object may not be garbage collected until its total reference count is zero.*
- `vx_status` `VX_API_CALL vxRemoveNode` (`vx_node` \*node)  
*Removes a Node from its parent Graph and releases it.*
- `vx_status` `VX_API_CALL vxSetNodeAttribute` (`vx_node` node, `vx_enum` attribute, const void \*ptr, `vx_size` size)  
*Allows a user to set attribute of a node before Graph Validation.*

### 3.47.2 Typedef Documentation

**typedef struct \_vx\_node\* vx\_node**

An opaque reference to a kernel node.

See also

`vxCreateGenericNode`

Definition at line 188 of file `vx_types.h`.

### 3.47.3 Enumeration Type Documentation

#### enum vx\_node\_attribute\_e

The node attributes list.

Enumerator

**VX\_NODE\_ATTRIBUTE\_STATUS** Queries the status of node execution. Use a [vx\\_status](#) parameter.

**VX\_NODE\_ATTRIBUTE\_PERFORMANCE** Queries the performance of the node execution. Use a [vx\\_perf\\_t](#) parameter.

**VX\_NODE\_ATTRIBUTE\_BORDER\_MODE** Gets or sets the border mode of the node. Use a [vx\\_border\\_mode\\_t](#) structure.

**VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_SIZE** Indicates the size of the kernel local memory area. Use a [vx\\_size](#) parameter.

**VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_PTR** Indicates the pointer kernel local memory area. Use a void \* parameter.

Definition at line 762 of file [vx\\_types.h](#).

### 3.47.4 Function Documentation

**vx\_status VX\_API\_CALL vxQueryNode ( vx\_node node, vx\_enum attribute, void \* ptr, vx\_size size )**

Allows a user to query information out of a node.

Parameters

|     |                  |                                                                         |
|-----|------------------|-------------------------------------------------------------------------|
| in  | <i>node</i>      | The reference to the node to query.                                     |
| in  | <i>attribute</i> | Use <a href="#">vx_node_attribute_e</a> value to query for information. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                     |
| in  | <i>size</i>      | The size in bytes in bytes of the container to which <i>ptr</i> points. |

Returns

A [vx\\_status\\_e](#) enumeration.

Return values

|                                    |                                |
|------------------------------------|--------------------------------|
| <b>VX_SUCCESS</b>                  | Successful                     |
| <b>VX_ERROR_INVALID_PARAMETERS</b> | The type or size is incorrect. |

**vx\_status VX\_API\_CALL vxSetNodeAttribute ( vx\_node node, vx\_enum attribute, const void \* ptr, vx\_size size )**

Allows a user to set attribute of a node before Graph Validation.

Parameters

|     |                  |                                                                         |
|-----|------------------|-------------------------------------------------------------------------|
| in  | <i>node</i>      | The reference to the node to set.                                       |
| in  | <i>attribute</i> | Use <a href="#">vx_node_attribute_e</a> value to query for information. |
| out | <i>ptr</i>       | The output pointer to where to send the value.                          |
| in  | <i>size</i>      | The size in bytes of the objects to which <i>ptr</i> points.            |

Note

Some attributes are inherited from the [vx\\_kernel](#), which was used to create the node. Some of these can be overridden using this API, notably [VX\\_NODE\\_ATTRIBUTE\\_LOCAL\\_DATA\\_SIZE](#) and [VX\\_NODE\\_ATTRIBUTE\\_LOCAL\\_DATA\\_PTR](#).

Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                   |                                          |
|-----------------------------------|------------------------------------------|
| <i>VX_SUCCESS</i>                 | The attribute was set.                   |
| <i>VX_ERROR_INVALID_REFERENCE</i> | node is not a vx_node.                   |
| <i>VX_ERROR_INVALID_PARAMETER</i> | size is not correct for the type needed. |

**vx\_status VX\_API\_CALL vxReleaseNode ( vx\_node \* node )**

Releases a reference to a Node object. The object may not be garbage collected until its total reference count is zero.

## Parameters

|    |             |                                                      |
|----|-------------|------------------------------------------------------|
| in | <i>node</i> | The pointer to the reference of the node to release. |
|----|-------------|------------------------------------------------------|

## Postcondition

After returning from this function the reference is zeroed.

## Returns

A *vx\_status\_e* enumeration.

## Return values

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                        |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If node is not a <i>vx_node</i> . |

**vx\_status VX\_API\_CALL vxRemoveNode ( vx\_node \* node )**

Removes a Node from its parent Graph and releases it.

## Parameters

|    |             |                                                |
|----|-------------|------------------------------------------------|
| in | <i>node</i> | The pointer to the node to remove and release. |
|----|-------------|------------------------------------------------|

## Postcondition

After returning from this function the reference is zeroed.

## Returns

A *vx\_status\_e* enumeration.

## Return values

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                        |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If node is not a <i>vx_node</i> . |

## 3.48 Object: Array

### 3.48.1 Detailed Description

Defines the Array Object Interface.

Array is a strongly-typed container, which provides random access by index to its elements in constant time. It uses value semantics for its own elements and holds copies of data. This is an example `for` loop over an Array:

```
vx_size i, stride = sizeof(vx_size);
void *base = NULL;
/* access entire array at once */
vxAccessArrayRange(array, 0, num_items, &stride, &base,
VX_READ_AND_WRITE);
for (i = 0; i < num_items; i++)
{
 vxArrayItem(mystruct, base, i, stride).some_uint += i;
 vxArrayItem(mystruct, base, i, stride).some_double = 3.14f;
}
vxCommitArrayRange(array, 0, num_items, base);
```

### Macros

- `#define vxArrayItem(type, ptr, index, stride) (*(type *) (vxFormatArrayPointer((ptr), (index), (stride))))`  
*Allows access to an array item as a typecast pointer deference.*
- `#define vxFormatArrayPointer(ptr, index, stride) (&(((vx_uint8*)(ptr))[(index) * (stride)]))`  
*Accesses a specific indexed element in an array.*

### Typedefs

- `typedef struct _vx_array * vx_array`  
*The Array Object. Array is a strongly-typed container for other data structures.*

### Enumerations

- `enum vx_array_attribute_e {`  
`VX_ARRAY_ATTRIBUTE_ITEMTYPE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x0,`  
`VX_ARRAY_ATTRIBUTE_NUMITEMS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x1,`  
`VX_ARRAY_ATTRIBUTE_CAPACITY = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x2,`  
`VX_ARRAY_ATTRIBUTE_ITEMSIZE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_ARRAY << 8)) + 0x3`  
`}`  
*The array object attributes.*

### Functions

- `vx_status VX_API_CALL vxAccessArrayRange (vx_array arr, vx_size start, vx_size end, vx_size *stride, void **ptr, vx_enum usage)`  
*Grants access to a sub-range of an Array. The number of elements in the sub-range is given by (end - start).*
- `vx_status VX_API_CALL vxAddArrayItems (vx_array arr, vx_size count, const void *ptr, vx_size stride)`  
*Adds items to the Array.*
- `vx_status VX_API_CALL vxCommitArrayRange (vx_array arr, vx_size start, vx_size end, const void *ptr)`  
*Commits data back to the Array object.*
- `vx_array VX_API_CALL vxCreateArray (vx_context context, vx_enum item_type, vx_size capacity)`  
*Creates a reference to an Array object.*
- `vx_array VX_API_CALL vxCreateVirtualArray (vx_graph graph, vx_enum item_type, vx_size capacity)`  
*Creates an opaque reference to a virtual Array with no direct user access.*
- `vx_status VX_API_CALL vxQueryArray (vx_array arr, vx_enum attribute, void *ptr, vx_size size)`  
*Queries the Array for some specific information.*



- **vx\_status** VX\_API\_CALL **vxReleaseArray** (**vx\_array** \*arr)

*Releases a reference of an Array object. The object may not be garbage collected until its total reference count is zero. After returning from this function the reference is zeroed.*

- **vx\_status** VX\_API\_CALL **vxTruncateArray** (**vx\_array** arr, **vx\_size** new\_num\_items)

*Truncates an Array (remove items from the end).*

### 3.48.2 Macro Definition Documentation

```
#define vxFormatArrayPointer(ptr, index, stride) (&(((vx_uint8*)(ptr))[(index) * (stride)]))
```

Accesses a specific indexed element in an array.

Parameters

|    |               |                                                                          |
|----|---------------|--------------------------------------------------------------------------|
| in | <i>ptr</i>    | The base pointer for the array range.                                    |
| in | <i>index</i>  | The index of the element, not byte, to access.                           |
| in | <i>stride</i> | The 'number of bytes' between the beginning of two consecutive elements. |

Definition at line 1846 of file [vx\\_api.h](#).

```
#define vxArrayItem(type, ptr, index, stride) (*(type*)(vxFormatArrayPointer((ptr), (index), (stride))))
```

Allows access to an array item as a typecast pointer dereference.

Parameters

|    |               |                                                                          |
|----|---------------|--------------------------------------------------------------------------|
| in | <i>type</i>   | The type of the item to access.                                          |
| in | <i>ptr</i>    | The base pointer for the array range.                                    |
| in | <i>index</i>  | The index of the element, not byte, to access.                           |
| in | <i>stride</i> | The 'number of bytes' between the beginning of two consecutive elements. |

Definition at line 1857 of file [vx\\_api.h](#).

### 3.48.3 Enumeration Type Documentation

```
enum vx_array_attribute_e
```

The array object attributes.

Enumerator

**VX\_ARRAY\_ATTRIBUTE\_ITEMTYPE** The type of the Array items. Use a [vx\\_enum](#) parameter.

**VX\_ARRAY\_ATTRIBUTE\_NUMITEMS** The number of items in the Array. Use a [vx\\_size](#) parameter.

**VX\_ARRAY\_ATTRIBUTE\_CAPACITY** The maximal number of items that the Array can hold. Use a [vx\\_size](#) parameter.

**VX\_ARRAY\_ATTRIBUTE\_ITEMSIZE** Queries an array item size. Use a [vx\\_size](#) parameter.

Definition at line 964 of file [vx\\_types.h](#).

### 3.48.4 Function Documentation

```
vx_array VX_API_CALL vxCreateArray (vx_context context, vx_enum item_type, vx_size capacity)
```

Creates a reference to an Array object.

User must specify the Array capacity (i.e., the maximal number of items that the array can hold).

**Parameters**

|    |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i>   | The reference to the overall Context.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| in | <i>item_type</i> | The type of objects to hold. Use: <ul style="list-style-type: none"> <li>• <code>VX_TYPE_RECTANGLE</code> for <code>vx_rectangle_t</code>.</li> <li>• <code>VX_TYPE_KEYPOINT</code> for <code>vx_keypoint_t</code>.</li> <li>• <code>VX_TYPE_COORDINATES2D</code> for <code>vx_coordinates2d_t</code>.</li> <li>• <code>VX_TYPE_COORDINATES3D</code> for <code>vx_coordinates3d_t</code>.</li> <li>• <code>vx_enum</code> Returned from <code>vxRegisterUserStruct</code>.</li> </ul> |
| in | <i>capacity</i>  | The maximal number of items that the array can hold.                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**Returns**

An array reference `vx_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**`vx_array VX_API_CALL vxCreateVirtualArray ( vx_graph graph, vx_enum item_type, vx_size capacity )`**

Creates an opaque reference to a virtual Array with no direct user access.

Virtual Arrays are useful when item type or capacity are unknown ahead of time and the Array is used as internal graph edge. Virtual arrays are scoped within the parent graph only.

All of the following constructions are allowed.

```
vx_context context = vxCreateContext();
vx_graph graph = vxCreateGraph(context);
vx_array virt[] = {
 vxCreateVirtualArray(graph, 0, 0), // totally unspecified
 vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 0), // unspecified
 capacity
 vxCreateVirtualArray(graph, VX_TYPE_KEYPOINT, 1000), // no access
};
```

**Parameters**

|    |                  |                                                                                                                   |
|----|------------------|-------------------------------------------------------------------------------------------------------------------|
| in | <i>graph</i>     | The reference to the parent graph.                                                                                |
| in | <i>item_type</i> | The type of objects to hold. This may be set to zero to indicate an unspecified item type.                        |
| in | <i>capacity</i>  | The maximal number of items that the array can hold. This may be set to zero to indicate an unspecified capacity. |

**See also**

`vxCreateArray` for a type list.

**Returns**

A array reference `vx_array`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**`vx_status VX_API_CALL vxReleaseArray ( vx_array * arr )`**

Releases a reference of an Array object. The object may not be garbage collected until its total reference count is zero. After returning from this function the reference is zeroed.

**Parameters**

|    |            |                                      |
|----|------------|--------------------------------------|
| in | <i>arr</i> | The pointer to the Array to release. |
|----|------------|--------------------------------------|

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                   |                                                   |
|-----------------------------------|---------------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                        |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If <i>arr</i> is not a <a href="#">vx_array</a> . |

**vx\_status VX\_API\_CALL vxQueryArray ( vx\_array *arr*, vx\_enum *attribute*, void \* *ptr*, vx\_size *size* )**

Queries the Array for some specific information.

**Parameters**

|     |                  |                                                                      |
|-----|------------------|----------------------------------------------------------------------|
| in  | <i>arr</i>       | The reference to the Array.                                          |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_array_attribute_e</a> . |
| out | <i>ptr</i>       | The location at which to store the resulting value.                  |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.       |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                    |                                                                          |
|------------------------------------|--------------------------------------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                                               |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the <i>arr</i> is not a <a href="#">vx_array</a> .                    |
| <i>VX_ERROR_NOT_SUPPORTED</i>      | If the <i>attribute</i> is not a value supported on this implementation. |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | If any of the other parameters are incorrect.                            |

**vx\_status VX\_API\_CALL vxAddArrayItems ( vx\_array *arr*, vx\_size *count*, const void \* *ptr*, vx\_size *stride* )**

Adds items to the Array.

This function increases the container size.

By default, the function does not reallocate memory, so if the container is already full (number of elements is equal to capacity) or it doesn't have enough space, the function returns [VX\\_FAILURE](#) error code.

**Parameters**

|    |               |                                                                        |
|----|---------------|------------------------------------------------------------------------|
| in | <i>arr</i>    | The reference to the Array.                                            |
| in | <i>count</i>  | The total number of elements to insert.                                |
| in | <i>ptr</i>    | The location at which to store the input values.                       |
| in | <i>stride</i> | The number of bytes between the beginning of two consecutive elements. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                    |                                               |
|------------------------------------|-----------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                    |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the <i>arr</i> is not a <i>vx_array</i> .  |
| <i>VX_FAILURE</i>                  | If the Array is full.                         |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | If any of the other parameters are incorrect. |

**vx\_status VX\_API\_CALL vxTruncateArray ( vx\_array *arr*, vx\_size *new\_num\_items* )**

Truncates an Array (remove items from the end).

Parameters

|         |                      |                                        |
|---------|----------------------|----------------------------------------|
| in, out | <i>arr</i>           | The reference to the Array.            |
| in      | <i>new_num_items</i> | The new number of items for the Array. |

Returns

A *vx\_status\_e* enumeration.

Return values

|                                    |                                                       |
|------------------------------------|-------------------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                            |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the <i>arr</i> is not a <i>vx_array</i> .          |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | The <i>new_size</i> is greater than the current size. |

**vx\_status VX\_API\_CALL vxAccessArrayRange ( vx\_array *arr*, vx\_size *start*, vx\_size *end*, vx\_size \* *stride*, void \*\* *ptr*, vx\_enum *usage* )**

Grants access to a sub-range of an Array. The number of elements in the sub-range is given by (end - start).

Parameters

|         |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in      | <i>arr</i>    | The reference to the Array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| in      | <i>start</i>  | The start index.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| in      | <i>end</i>    | The end index. (end - start) elements are accessed from start.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| in, out | <i>stride</i> | <p>A pointer to 'number of bytes' between the beginning of two consequent elements.</p> <ul style="list-style-type: none"> <li>Input case: <i>ptr</i> is a pointer to a non-NULL pointer. The stride parameter must be the address of a <i>vx_size</i> scalar that describes how the user will access the requested array data at address (*<i>ptr</i>).</li> <li>Output Case: <i>ptr</i> is a pointer to a NULL pointer. The function fills the <i>vx_size</i> scalar pointed by <i>stride</i> with the element stride information that the user must consult to access the array elements at address (*<i>ptr</i>).</li> </ul> |

|     |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| out | ptr   | <p>A pointer to a pointer to a location to store the requested data.</p> <ul style="list-style-type: none"> <li>Input Case: ptr is a pointer to a non-NULL pointer to a valid buffer. This buffer will be used in one of two ways, depending on the value of the usage parameter. If usage is <code>VX_WRITE_ONLY</code>, then the buffer must contain element data that the user wants to replace the array's element data with. Otherwise (i.e., usage is not <code>VX_WRITE_ONLY</code>), the array's current element data will be written to the memory starting at address (*ptr) as storage memory for the access request. The caller must ensure enough memory has been allocated for the requested array range with the requested stride.</li> <li>Output Case: ptr is a pointer to a NULL pointer. This NULL pointer will be overwritten with a pointer to the address where the requested data can be accessed. (*ptr) must eventually be provided as the ptr parameter of a call to <code>vxCommitArrayRange</code>.</li> </ul> |
| in  | usage | This declares the intended usage of the pointer using the <code>vx_accessor_e</code> enumeration.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Note**

The stride and ptr parameters must both be input, or both be output, otherwise the behavior is undefined.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                          |                                                                          |
|------------------------------------------|--------------------------------------------------------------------------|
| <code>VX_SUCCESS</code>                  | No errors.                                                               |
| <code>VX_ERROR_OPTIMIZED_AWAY</code>     | If the reference is a virtual array and cannot be accessed or committed. |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the <i>arr</i> is not a <code>vx_array</code> .                       |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If any of the other parameters are incorrect.                            |

**Postcondition**

`vxCommitArrayRange`

**vx\_status VX\_API\_CALL vxCommitArrayRange ( vx\_array arr, vx\_size start, vx\_size end, const void \* ptr )**

Commits data back to the Array object.

This allows a user to commit data to a sub-range of an Array. The number of elements in the sub-range is given by (end - start).

**Parameters**

|    |       |                                                                |
|----|-------|----------------------------------------------------------------|
| in | arr   | The reference to the Array.                                    |
| in | start | The start index.                                               |
| in | end   | The end index. (end - start) elements are accessed from start. |
| in | ptr   | The user supplied pointer.                                     |

**Returns**

A `vx_status_e` enumeration.

## Return values

|                                                 |                                                                          |
|-------------------------------------------------|--------------------------------------------------------------------------|
| <code>VX_SUCCESS</code>                         | No errors.                                                               |
| <code>VX_ERROR_OPTIMIZED_↵<br/>_AWAY</code>     | If the reference is a virtual array and cannot be accessed or committed. |
| <code>VX_ERROR_INVALID_R_↵<br/>EFERENCE</code>  | If the <i>arr</i> is not a <code>vx_array</code> .                       |
| <code>VX_ERROR_INVALID_P_↵<br/>ARAMETERS</code> | If any of the other parameters are incorrect.                            |

## 3.49 Object: Convolution

### 3.49.1 Detailed Description

Defines the Image Convolution Object interface.

#### Typedefs

- typedef struct \_vx\_convolution \* [vx\\_convolution](#)  
*The Convolution Object. A user-defined convolution kernel of MxM elements.*

#### Enumerations

- enum [vx\\_convolution\\_attribute\\_e](#) {  
[VX\\_CONVOLUTION\\_ATTRIBUTE\\_ROWS](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_CONVOLUTION << 8)) + 0x0,  
[VX\\_CONVOLUTION\\_ATTRIBUTE\\_COLUMNS](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_CONVOLUTION << 8)) + 0x1,  
[VX\\_CONVOLUTION\\_ATTRIBUTE\\_SCALE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_CONVOLUTION << 8)) + 0x2,  
[VX\\_CONVOLUTION\\_ATTRIBUTE\\_SIZE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_CONVOLUTION << 8)) + 0x3 }  
*The convolution attributes.*

#### Functions

- [vx\\_convolution](#) VX\_API\_CALL [vxCreateConvolution](#) ([vx\\_context](#) context, [vx\\_size](#) columns, [vx\\_size](#) rows)  
*Creates a reference to a convolution matrix object.*
- [vx\\_status](#) VX\_API\_CALL [vxQueryConvolution](#) ([vx\\_convolution](#) conv, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries an attribute on the convolution matrix object.*
- [vx\\_status](#) VX\_API\_CALL [vxReadConvolutionCoefficients](#) ([vx\\_convolution](#) conv, [vx\\_int16](#) \*array)  
*Gets the convolution data (copy).*
- [vx\\_status](#) VX\_API\_CALL [vxReleaseConvolution](#) ([vx\\_convolution](#) \*conv)  
*Releases the reference to a convolution matrix. The object may not be garbage collected until its total reference count is zero.*
- [vx\\_status](#) VX\_API\_CALL [vxSetConvolutionAttribute](#) ([vx\\_convolution](#) conv, [vx\\_enum](#) attribute, const void \*ptr, [vx\\_size](#) size)  
*Sets attributes on the convolution object.*
- [vx\\_status](#) VX\_API\_CALL [vxWriteConvolutionCoefficients](#) ([vx\\_convolution](#) conv, const [vx\\_int16](#) \*array)  
*Sets the convolution data (copy)*

### 3.49.2 Enumeration Type Documentation

#### enum [vx\\_convolution\\_attribute\\_e](#)

The convolution attributes.

Enumerator

- [VX\\_CONVOLUTION\\_ATTRIBUTE\\_ROWS](#)** The number of rows of the convolution matrix. Use a [vx\\_size](#) parameter.
- [VX\\_CONVOLUTION\\_ATTRIBUTE\\_COLUMNS](#)** The number of columns of the convolution matrix. Use a [vx\\_size](#) parameter.
- [VX\\_CONVOLUTION\\_ATTRIBUTE\\_SCALE](#)** The scale of the convolution matrix. Use a [vx\\_uint32](#) parameter.

## Note

For 1.0, only powers of 2 are supported up to  $2^{31}$ .

**VX\_CONVOLUTION\_ATTRIBUTE\_SIZE** The total size of the convolution matrix in bytes. Use a `vx_size` parameter.

Definition at line 916 of file `vx_types.h`.

### 3.49.3 Function Documentation

**vx\_convolution VX\_API\_CALL vxCreateConvolution ( vx\_context context, vx\_size columns, vx\_size rows )**

Creates a reference to a convolution matrix object.

## Parameters

|    |                |                                                                                                                                                                                              |
|----|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the overall context.                                                                                                                                                        |
| in | <i>columns</i> | The columns dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from <code>VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION</code> . |
| in | <i>rows</i>    | The rows dimension of the convolution. Must be odd and greater than or equal to 3 and less than the value returned from <code>VX_CONTEXT_ATTRIBUTE_CONVOLUTION_MAXIMUM_DIMENSION</code> .    |

## Returns

A convolution reference `vx_convolution`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx\_status VX\_API\_CALL vxReleaseConvolution ( vx\_convolution \* conv )**

Releases the reference to a convolution matrix. The object may not be garbage collected until its total reference count is zero.

## Parameters

|    |             |                                                   |
|----|-------------|---------------------------------------------------|
| in | <i>conv</i> | The pointer to the convolution matrix to release. |
|----|-------------|---------------------------------------------------|

## Postcondition

After returning from this function the reference is zeroed.

## Returns

A `vx_status_e` enumeration.

## Return values

|                                         |                                                             |
|-----------------------------------------|-------------------------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                                  |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If <code>conv</code> is not a <code>vx_convolution</code> . |

**vx\_status VX\_API\_CALL vxQueryConvolution ( vx\_convolution conv, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries an attribute on the convolution matrix object.



**Parameters**

|     |                  |                                                                                       |
|-----|------------------|---------------------------------------------------------------------------------------|
| in  | <i>conv</i>      | The convolution matrix object to set.                                                 |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_convolution_attribute_e</a> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                                   |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.                        |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxSetConvolutionAttribute ( vx\_convolution *conv*, vx\_enum *attribute*, const void \* *ptr*, vx\_size *size* )**

Sets attributes on the convolution object.

**Parameters**

|    |                  |                                                                                        |
|----|------------------|----------------------------------------------------------------------------------------|
| in | <i>conv</i>      | The coordinates object to set.                                                         |
| in | <i>attribute</i> | The attribute to modify. Use a <a href="#">vx_convolution_attribute_e</a> enumeration. |
| in | <i>ptr</i>       | The pointer to the value to which to set the attribute.                                |
| in | <i>size</i>      | The size in bytes of the data pointed to by <i>ptr</i> .                               |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxReadConvolutionCoefficients ( vx\_convolution *conv*, vx\_int16 \* *array* )**

Gets the convolution data (copy).

**Parameters**

|     |              |                                     |
|-----|--------------|-------------------------------------|
| in  | <i>conv</i>  | The reference to the convolution.   |
| out | <i>array</i> | The array to place the convolution. |

**See also**

[vxQueryConvolution](#) and [VX\\_CONVOLUTION\\_ATTRIBUTE\\_SIZE](#) to get the needed number of bytes of the array.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxWriteConvolutionCoefficients ( vx\_convolution *conv*, const vx\_int16 \* *array* )**

Sets the convolution data (copy)

**Parameters**

|    |              |                                                     |
|----|--------------|-----------------------------------------------------|
| in | <i>conv</i>  | The reference to the convolution.                   |
| in | <i>array</i> | The array containing the convolution to be written. |

**See also**

[vxQueryConvolution](#) and [VX\\_CONVOLUTION\\_ATTRIBUTE\\_SIZE](#) to get the needed number of bytes of the array.

**Returns**

A [vx\\_status\\_e](#) enumeration.

## 3.50 Object: Distribution

### 3.50.1 Detailed Description

Defines the Distribution Object Interface.

#### Typedefs

- typedef struct \_vx\_distribution \* [vx\\_distribution](#)

*The Distribution object. This has a user-defined number of bins over a user-defined range (within a uint32\_t range).*

#### Enumerations

- enum [vx\\_distribution\\_attribute\\_e](#) {  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_DIMENSIONS](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRI←  
BUTION << 8)) + 0x0,  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_OFFSET](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRIBUTI←  
ON << 8)) + 0x1,  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_RANGE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRIBUTION  
<< 8)) + 0x2,  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_BINS](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRIBUTION  
<< 8)) + 0x3,  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_WINDOW](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRIBUT←  
ION << 8)) + 0x4,  
[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_SIZE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_DISTRIBUTION  
<< 8)) + 0x5 }

*The distribution attribute list.*

#### Functions

- [vx\\_status](#) VX\_API\_CALL [vxAccessDistribution](#) ([vx\\_distribution](#) distribution, void \*\*ptr, [vx\\_enum](#) usage)  
*Grants access to a distribution object and increments the object reference count in case of success.*
- [vx\\_status](#) VX\_API\_CALL [vxCommitDistribution](#) ([vx\\_distribution](#) distribution, const void \*ptr)  
*Commits the distribution objec> and decrements the object reference count in case of success. The memory must be a vx\_uint32 array of a value at least as big as the value returned via [VX\\_DISTRIBUTION\\_ATTRIBUTE\\_BINS](#).*
- [vx\\_distribution](#) VX\_API\_CALL [vxCreateDistribution](#) ([vx\\_context](#) context, [vx\\_size](#) numBins, [vx\\_int32](#) offset, [vx\\_uint32](#) range)  
*Creates a reference to a 1D Distribution of a consecutive interval [offset, offset + range - 1] defined by a start offset and valid range, divided equally into numBins parts.*
- [vx\\_status](#) VX\_API\_CALL [vxQueryDistribution](#) ([vx\\_distribution](#) distribution, [vx\\_enum](#) attribute, void \*ptr, [vx\\_←  
size](#) size)  
*Queries a Distribution object.*
- [vx\\_status](#) VX\_API\_CALL [vxReleaseDistribution](#) ([vx\\_distribution](#) \*distribution)  
*Releases a reference to a distribution object. The object may not be garbage collected until its total reference count is zero.*

### 3.50.2 Enumeration Type Documentation

enum [vx\\_distribution\\_attribute\\_e](#)

The distribution attribute list.

Enumerator

**[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_DIMENSIONS](#)** Indicates the number of dimensions in the distribution. Use a [vx\\_size](#) parameter.

**[VX\\_DISTRIBUTION\\_ATTRIBUTE\\_OFFSET](#)** Indicates the start of the values to use (inclusive). Use a [vx\\_←  
int32](#) parameter.

**VX\_DISTRIBUTION\_ATTRIBUTE\_RANGE** Indicates end value to use as the range. Use a `vx_uint32` parameter.

**VX\_DISTRIBUTION\_ATTRIBUTE\_BINS** Indicates the number of bins. Use a `vx_size` parameter.

**VX\_DISTRIBUTION\_ATTRIBUTE\_WINDOW** Indicates the range of a bin. Use a `vx_uint32` parameter.

**VX\_DISTRIBUTION\_ATTRIBUTE\_SIZE** Indicates the total size of the distribution in bytes. Use a `vx_size` parameter.

Definition at line 854 of file `vx_types.h`.

### 3.50.3 Function Documentation

**vx\_distribution VX\_API\_CALL vxCreateDistribution ( vx\_context context, vx\_size numBins, vx\_int32 offset, vx\_uint32 range )**

Creates a reference to a 1D Distribution of a consecutive interval [offset, offset + range - 1] defined by a start offset and valid range, divided equally into numBins parts.

Parameters

|    |                |                                                                                        |
|----|----------------|----------------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the overall context.                                                  |
| in | <i>numBins</i> | The number of bins in the distribution.                                                |
| in | <i>offset</i>  | The start offset into the range value that marks the beginning of the 1D Distribution. |
| in | <i>range</i>   | The total number of the values.                                                        |

Returns

A distribution reference `vx_distribution`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx\_status VX\_API\_CALL vxReleaseDistribution ( vx\_distribution \* distribution )**

Releases a reference to a distribution object. The object may not be garbage collected until its total reference count is zero.

Parameters

|    |                     |                                               |
|----|---------------------|-----------------------------------------------|
| in | <i>distribution</i> | The reference to the distribution to release. |
|----|---------------------|-----------------------------------------------|

Postcondition

After returning from this function the reference is zeroed.

Returns

A `vx_status_e` enumeration.

Return values

|                                   |                                                         |
|-----------------------------------|---------------------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                              |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If distribution is not a <code>vx_distribution</code> . |

**vx\_status VX\_API\_CALL vxQueryDistribution ( vx\_distribution distribution, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries a Distribution object.

**Parameters**

|     |                     |                                                                                        |
|-----|---------------------|----------------------------------------------------------------------------------------|
| in  | <i>distribution</i> | The reference to the distribution to query.                                            |
| in  | <i>attribute</i>    | The attribute to query. Use a <a href="#">vx_distribution_attribute_e</a> enumeration. |
| out | <i>ptr</i>          | The location at which to store the resulting value.                                    |
| in  | <i>size</i>         | The size in bytes of the container to which <i>ptr</i> points.                         |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxAccessDistribution ( vx\_distribution *distribution*, void \*\* *ptr*, vx\_enum *usage* )**

Grants access to a distribution object and increments the object reference count in case of success.

**Parameters**

|         |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in      | <i>distribution</i> | The reference to the distribution to access.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| in, out | <i>ptr</i>          | The user-supplied address to a pointer, via which the requested contents are returned. <ul style="list-style-type: none"> <li>• If <i>ptr</i> is NULL, an error occurs.</li> <li>• If <i>ptr</i> is not NULL and (*<i>ptr</i>) is NULL, (*<i>ptr</i>) will be set to the address of a memory area managed by the OpenVX framework containing the requested data.</li> <li>• If both <i>ptr</i> and (*<i>ptr</i>) are not NULL, requested data will be copied to (*<i>ptr</i>) (optionally in case of write-only access).</li> </ul> |
| in      | <i>usage</i>        | The <a href="#">vx_accessor_e</a> value to describe the access of the object.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Postcondition**

[vxCommitDistribution](#)

**vx\_status VX\_API\_CALL vxCommitDistribution ( vx\_distribution *distribution*, const void \* *ptr* )**

Commits the distribution object and decrements the object reference count in case of success. The memory must be a vx\_uint32 array of a value at least as big as the value returned via [VX\\_DISTRIBUTION\\_ATTRIBUTE\\_BYTES](#).

**Parameters**

|    |                     |                                                                                                           |
|----|---------------------|-----------------------------------------------------------------------------------------------------------|
| in | <i>distribution</i> | The Distribution to modify.                                                                               |
| in | <i>ptr</i>          | The pointer provided or returned by <a href="#">vxAccessDistribution</a> . The <i>ptr</i> cannot be NULL. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Precondition**

[vxAccessDistribution](#).

## 3.51 Object: Image

### 3.51.1 Detailed Description

Defines the Image Object interface.

#### Data Structures

- struct `vx_imagepatch_addressing_t`

*The addressing image patch structure is used by the Host only to address pixels in an image patch. The fields of the structure are defined as: [More...](#)*

#### Macros

- #define `VX_IMAGEPATCH_ADDR_INIT` {0u, 0u, 0, 0, 0u, 0u, 0u, 0u}

*Use to initialize a `vx_imagepatch_addressing_t` structure on the stack.*

#### Typedefs

- typedef struct `_vx_image` \* `vx_image`

*An opaque reference to an image.*

#### Enumerations

- enum `vx_channel_range_e` {  
`VX_CHANNEL_RANGE_FULL` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_RANGE` << 12)) + 0x0,  
`VX_CHANNEL_RANGE_RESTRICTED` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_RANGE` << 12)) + 0x1 }

*The image channel range list used by the `VX_IMAGE_ATTRIBUTE_RANGE` attribute of a `vx_image`.*

- enum `vx_color_space_e` {  
`VX_COLOR_SPACE_NONE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_SPACE` << 12)) + 0x0,  
`VX_COLOR_SPACE_BT601_525` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_SPACE` << 12)) + 0x1,  
`VX_COLOR_SPACE_BT601_625` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_SPACE` << 12)) + 0x2,  
`VX_COLOR_SPACE_BT709` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_COLOR_SPACE` << 12)) + 0x3,  
`VX_COLOR_SPACE_DEFAULT` = `VX_COLOR_SPACE_BT709` }

*The image color space list used by the `VX_IMAGE_ATTRIBUTE_SPACE` attribute of a `vx_image`.*

- enum `vx_image_attribute_e` {  
`VX_IMAGE_ATTRIBUTE_WIDTH` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x0,  
`VX_IMAGE_ATTRIBUTE_HEIGHT` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x1,  
`VX_IMAGE_ATTRIBUTE_FORMAT` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x2,  
`VX_IMAGE_ATTRIBUTE_PLANES` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x3,  
`VX_IMAGE_ATTRIBUTE_SPACE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x4,  
`VX_IMAGE_ATTRIBUTE_RANGE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x5,  
`VX_IMAGE_ATTRIBUTE_SIZE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_IMAGE` << 8)) + 0x6 }

*The image attributes list.*

#### Functions

- `vx_status` `VX_API_CALL vxAccessImagePatch` (`vx_image` image, const `vx_rectangle_t` \*rect, `vx_uint32` plane\_index, `vx_imagepatch_addressing_t` \*addr, void \*\*ptr, `vx_enum` usage)

*Allows the User to extract a rectangular patch (subset) of an image from a single plane.*

- **vx\_status** VX\_API\_CALL **vxCommitImagePatch** (**vx\_image** image, **vx\_rectangle\_t** \*rect, **vx\_uint32** plane\_index, **vx\_imagepatch\_addressing\_t** \*addr, const void \*ptr)  
*This allows the User to commit a rectangular patch (subset) of an image from a single plane.*
- **vx\_size** VX\_API\_CALL **vxComputeImagePatchSize** (**vx\_image** image, const **vx\_rectangle\_t** \*rect, **vx\_uint32** plane\_index)  
*This computes the size needed to retrieve an image patch from an image.*
- **vx\_image** VX\_API\_CALL **vxCreateImage** (**vx\_context** context, **vx\_uint32** width, **vx\_uint32** height, **vx\_df\_image** color)  
*Creates an opaque reference to an image buffer.*
- **vx\_image** VX\_API\_CALL **vxCreateImageFromHandle** (**vx\_context** context, **vx\_df\_image** color, **vx\_imagepatch\_addressing\_t** addr[], void \*ptrs[], **vx\_enum** import\_type)  
*Creates a reference to an image object that was externally allocated.*
- **vx\_image** VX\_API\_CALL **vxCreateImageFromROI** (**vx\_image** img, const **vx\_rectangle\_t** \*rect)  
*Creates an image from another image given a rectangle. This second reference refers to the data in the original image. Updates to this image updates the parent image. The rectangle must be defined within the pixel space of the parent image.*
- **vx\_image** VX\_API\_CALL **vxCreateUniformImage** (**vx\_context** context, **vx\_uint32** width, **vx\_uint32** height, **vx\_df\_image** color, const void \*value)  
*Creates a reference to an image object that has a singular, uniform value in all pixels.*
- **vx\_image** VX\_API\_CALL **vxCreateVirtualImage** (**vx\_graph** graph, **vx\_uint32** width, **vx\_uint32** height, **vx\_df\_image** color)  
*Creates an opaque reference to an image buffer with no direct user access. This function allows setting the image width, height, or format.*
- void \*VX\_API\_CALL **vxFormatImagePatchAddress1d** (void \*ptr, **vx\_uint32** index, const **vx\_imagepatch\_addressing\_t** \*addr)  
*Accesses a specific indexed pixel in an image patch.*
- void \*VX\_API\_CALL **vxFormatImagePatchAddress2d** (void \*ptr, **vx\_uint32** x, **vx\_uint32** y, const **vx\_imagepatch\_addressing\_t** \*addr)  
*Accesses a specific pixel at a 2d coordinate in an image patch.*
- **vx\_status** VX\_API\_CALL **vxGetValidRegionImage** (**vx\_image** image, **vx\_rectangle\_t** \*rect)  
*Retrieves the valid region of the image as a rectangle.*
- **vx\_status** VX\_API\_CALL **vxQueryImage** (**vx\_image** image, **vx\_enum** attribute, void \*ptr, **vx\_size** size)  
*Retrieves various attributes of an image.*
- **vx\_status** VX\_API\_CALL **vxReleaseImage** (**vx\_image** \*image)  
*Releases a reference to an image object. The object may not be garbage collected until its total reference count is zero.*
- **vx\_status** VX\_API\_CALL **vxSetImageAttribute** (**vx\_image** image, **vx\_enum** attribute, const void \*ptr, **vx\_size** size)  
*Allows setting attributes on the image.*

### 3.51.2 Data Structure Documentation

#### struct vx\_imagepatch\_addressing\_t

The addressing image patch structure is used by the Host only to address pixels in an image patch. The fields of the structure are defined as:

- dim - The dimensions of the image in logical pixel units in the x & y direction.
- stride - The physical byte distance from a logical pixel to the next logically adjacent pixel in the positive x or y direction.
- scale - The relationship of scaling from the primary plane (typically the zero indexed plane) to this plane. An integer down-scaling factor of  $f$  shall be set to a value equal to  $scale = \frac{unity}{f}$  and an integer up-scaling factor of  $f$  shall be set to a value of  $scale = unity * f$ .  $unity$  is defined as **VX\_SCALE\_UNITY**.

- step - The step is the number of logical pixel units to skip to arrive at the next physically unique pixel. For example, on a plane that is half-scaled in a dimension, the step in that dimension is 2 to indicate that every other pixel in that dimension is an alias. This is useful in situations where iteration over unique pixels is required, such as in serializing or de-serializing the image patch information.

See also

### `vxAccessImagePatch`

```

/*
 * Copyright (c) 2013-2014 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 *
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
 vx_status status = VX_SUCCESS;
 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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x*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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y+addr.scale_y) /
 VX_SCALE_UNITY) +
 ((addr.stride_x*x+addr.scale_x) /
 VX_SCALE_UNITY);
 tmp[i] = pixel;
 }
 }
 }
}

```

```

 }
}

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

/* this commits the data back to the image. If rect were 0 or empty, it
 * would just decrement the reference (used when reading an image only)
 */
status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
}
vxReleaseImage(&image);
return status;
}

```

Definition at line 1309 of file [vx\\_types.h](#).

#### Data Fields

|                           |                       |                                                                                                                                                                                           |
|---------------------------|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_uint32</a> | <code>dim_x</code>    | Width of patch in X dimension in pixels.                                                                                                                                                  |
| <a href="#">vx_uint32</a> | <code>dim_y</code>    | Height of patch in Y dimension in pixels.                                                                                                                                                 |
| <a href="#">vx_int32</a>  | <code>stride_x</code> | Stride in X dimension in bytes.                                                                                                                                                           |
| <a href="#">vx_int32</a>  | <code>stride_y</code> | Stride in Y dimension in bytes.                                                                                                                                                           |
| <a href="#">vx_uint32</a> | <code>scale_x</code>  | Scale of X dimension. For sub-sampled planes this is the scaling factor of the dimension of the plane in relation to the zero plane. Use <a href="#">VX_SCALE_UNITY</a> in the numerator. |
| <a href="#">vx_uint32</a> | <code>scale_y</code>  | Scale of Y dimension. For sub-sampled planes this is the scaling factor of the dimension of the plane in relation to the zero plane. Use <a href="#">VX_SCALE_UNITY</a> in the numerator. |
| <a href="#">vx_uint32</a> | <code>step_x</code>   | Step of X dimension in pixels.                                                                                                                                                            |
| <a href="#">vx_uint32</a> | <code>step_y</code>   | Step of Y dimension in pixels.                                                                                                                                                            |

### 3.51.3 Typedef Documentation

**typedef struct \_vx\_image\* vx\_image**

An opaque reference to an image.

See also

[vxCreateImage](#)

Definition at line 166 of file [vx\\_types.h](#).

### 3.51.4 Enumeration Type Documentation

**enum vx\_image\_attribute\_e**

The image attributes list.

Enumerator

**VX\_IMAGE\_ATTRIBUTE\_WIDTH** Queries an image for its height. Use a [vx\\_uint32](#) parameter.

**VX\_IMAGE\_ATTRIBUTE\_HEIGHT** Queries an image for its width. Use a [vx\\_uint32](#) parameter.

**VX\_IMAGE\_ATTRIBUTE\_FORMAT** Queries an image for its format. Use a [vx\\_df\\_image](#) parameter.

**VX\_IMAGE\_ATTRIBUTE\_PLANES** Queries an image for its number of planes. Use a [vx\\_size](#) parameter.



**VX\_IMAGE\_ATTRIBUTE\_SPACE** Queries an image for its color space (see [vx\\_color\\_space\\_e](#)). Use a [vx\\_enum](#) parameter.

**VX\_IMAGE\_ATTRIBUTE\_RANGE** Queries an image for its channel range (see [vx\\_channel\\_range\\_e](#)). Use a [vx\\_enum](#) parameter.

**VX\_IMAGE\_ATTRIBUTE\_SIZE** Queries an image for its total number of bytes. Use a [vx\\_size](#) parameter.

Definition at line 800 of file [vx\\_types.h](#).

#### enum vx\_color\_space\_e

The image color space list used by the [VX\\_IMAGE\\_ATTRIBUTE\\_SPACE](#) attribute of a [vx\\_image](#).

Enumerator

**VX\_COLOR\_SPACE\_NONE** Use to indicate that no color space is used.

**VX\_COLOR\_SPACE\_BT601\_525** Use to indicate that the BT.601 coefficients and SMPTE C primaries are used for conversions.

**VX\_COLOR\_SPACE\_BT601\_625** Use to indicate that the BT.601 coefficients and BTU primaries are used for conversions.

**VX\_COLOR\_SPACE\_BT709** Use to indicate that the BT.709 coefficients are used for conversions.

**VX\_COLOR\_SPACE\_DEFAULT** All images in VX are by default BT.709.

Definition at line 1063 of file [vx\\_types.h](#).

#### enum vx\_channel\_range\_e

The image channel range list used by the [VX\\_IMAGE\\_ATTRIBUTE\\_RANGE](#) attribute of a [vx\\_image](#).

Enumerator

**VX\_CHANNEL\_RANGE\_FULL** Full range of the unit of the channel.

**VX\_CHANNEL\_RANGE\_RESTRICTED** Restricted range of the unit of the channel based on the space given.

Definition at line 1080 of file [vx\\_types.h](#).

### 3.51.5 Function Documentation

**vx\_image VX\_API\_CALL vxCreateImage ( vx\_context context, vx\_uint32 width, vx\_uint32 height, vx\_df\_image color )**

Creates an opaque reference to an image buffer.

Not guaranteed to exist until the [vx\\_graph](#) containing it has been verified.

Parameters

|    |                |                                                                                                                     |
|----|----------------|---------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the implementation context.                                                                        |
| in | <i>width</i>   | The image width in pixels.                                                                                          |
| in | <i>height</i>  | The image height in pixels.                                                                                         |
| in | <i>color</i>   | The VX_DF_IMAGE ( <a href="#">vx_df_image_e</a> ) code that represents the format of the image and the color space. |

Returns

An image reference [vx\\_image](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

See also

[vxAccessImagePatch](#) to obtain direct memory access to the image data.

**vx\_image VX\_API\_CALL vxCreateImageFromROI ( vx\_image img, const vx\_rectangle\_t \* rect )**

Creates an image from another image given a rectangle. This second reference refers to the data in the original image. Updates to this image updates the parent image. The rectangle must be defined within the pixel space of the parent image.

**Parameters**

|    |             |                                                                                            |
|----|-------------|--------------------------------------------------------------------------------------------|
| in | <i>img</i>  | The reference to the parent image.                                                         |
| in | <i>rect</i> | The region of interest rectangle. Must contain points within the parent image pixel space. |

**Returns**

An image reference [vx\\_image](#) to the sub-image. Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_image VX\_API\_CALL vxCreateUniformImage ( vx\_context context, vx\_uint32 width, vx\_uint32 height, vx\_df\_image color, const void \* value )**

Creates a reference to an image object that has a singular, uniform value in all pixels.

The value pointer must reflect the specific format of the desired image. For example:

| Color                            | Value Ptr                          |
|----------------------------------|------------------------------------|
| <a href="#">VX_DF_IMAGE_U8</a>   | vx_uint8 *                         |
| <a href="#">VX_DF_IMAGE_S16</a>  | vx_int16 *                         |
| <a href="#">VX_DF_IMAGE_U16</a>  | vx_uint16 *                        |
| <a href="#">VX_DF_IMAGE_S32</a>  | vx_int32 *                         |
| <a href="#">VX_DF_IMAGE_U32</a>  | vx_uint32 *                        |
| <a href="#">VX_DF_IMAGE_RGB</a>  | vx_uint8 pixel[3] in R, G, B order |
| <a href="#">VX_DF_IMAGE_RGBX</a> | vx_uint8 pixels[4]                 |
| Any YUV                          | vx_uint8 pixel[3] in Y, U, V order |

**Parameters**

|    |                |                                                                                                                     |
|----|----------------|---------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the implementation context.                                                                        |
| in | <i>width</i>   | The image width in pixels.                                                                                          |
| in | <i>height</i>  | The image height in pixels.                                                                                         |
| in | <i>color</i>   | The VX_DF_IMAGE ( <a href="#">vx_df_image_e</a> ) code that represents the format of the image and the color space. |
| in | <i>value</i>   | The pointer to the pixel value to which to set all pixels.                                                          |

**Returns**

An image reference [vx\\_image](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**See also**

[vxAccessImagePatch](#) to obtain direct memory access to the image data.

**Note**

[vxAccessImagePatch](#) and [vxCommitImagePatch](#) may be called with a uniform image reference.

**vx\_image VX\_API\_CALL vxCreateVirtualImage ( vx\_graph graph, vx\_uint32 width, vx\_uint32 height, vx\_df\_image color )**

Creates an opaque reference to an image buffer with no direct user access. This function allows setting the image width, height, or format.

Virtual data objects allow users to connect various nodes within a graph via data references without access to that data, but they also permit the implementation to take maximum advantage of possible optimizations. Use this API to create a data reference to link two or more nodes together when the intermediate data are not required to be accessed by outside entities. This API in particular allows the user to define the image format of the data without requiring the exact dimensions. Virtual objects are scoped within the graph they are declared a part of, and can't be shared outside of this scope. All of the following constructions of virtual images are valid.

```

vx_context context = vxCreateContext();
vx_graph graph = vxCreateGraph(context);
vx_image virt[] = {
 vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_U8), // no specified
 dimension
 vxCreateVirtualImage(graph, 320, 240, VX_DF_IMAGE_VIRT), // no
 specified format
 vxCreateVirtualImage(graph, 640, 480, VX_DF_IMAGE_U8), // no user
 access
};

```

**Parameters**

|    |               |                                                                                                                                                                                                                       |
|----|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>graph</i>  | The reference to the parent graph.                                                                                                                                                                                    |
| in | <i>width</i>  | The width of the image in pixels. A value of zero informs the interface that the value is unspecified.                                                                                                                |
| in | <i>height</i> | The height of the image in pixels. A value of zero informs the interface that the value is unspecified.                                                                                                               |
| in | <i>color</i>  | The VX_DF_IMAGE ( <a href="#">vx_df_image_e</a> ) code that represents the format of the image and the color space. A value of <a href="#">VX_DF_IMAGE_VIRT</a> informs the interface that the format is unspecified. |

**Returns**

An image reference [vx\\_image](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**Note**

Passing this reference to [vxAccessImagePatch](#) will return an error.

**vx\_image VX\_API\_CALL vxCreateImageFromHandle ( vx\_context context, vx\_df\_image color, vx\_imagepatch\_addressing\_t addrs[], void \* ptrs[], vx\_enum import\_type )**

Creates a reference to an image object that was externally allocated.

**Parameters**

|    |                    |                                                                                                                                                                          |
|----|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i>     | The reference to the implementation context.                                                                                                                             |
| in | <i>color</i>       | See the <a href="#">vx_df_image_e</a> codes. This mandates the number of planes needed to be valid in the <i>addrs</i> and <i>ptrs</i> arrays based on the format given. |
| in | <i>addrs[]</i>     | The array of image patch addressing structures that define the dimension and stride of the array of pointers.                                                            |
| in | <i>ptrs[]</i>      | The array of platform-defined references to each plane.                                                                                                                  |
| in | <i>import_type</i> | <a href="#">vx_import_type_e</a> . When giving <a href="#">VX_IMPORT_TYPE_HOST</a> the <i>ptrs</i> array is assumed to be HOST accessible pointers to memory.            |

**Returns**

An image reference [vx\\_image](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**Note**

The user must call [vxAccessImagePatch](#) prior to accessing the pixels of an image, even if the image was created via [vxCreateImageFromHandle](#). Reads or writes to memory referenced by *ptrs[]* after calling [vxCreateImageFromHandle](#) without first calling [vxAccessImagePatch](#) will result in undefined behavior.

**vx\_status VX\_API\_CALL vxQueryImage ( vx\_image image, vx\_enum attribute, void \* ptr, vx\_size size )**

Retrieves various attributes of an image.

**Parameters**

|     |                  |                                                                      |
|-----|------------------|----------------------------------------------------------------------|
| in  | <i>image</i>     | The reference to the image to query.                                 |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_image_attribute_e</a> . |
| out | <i>ptr</i>       | The location at which to store the resulting value.                  |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.       |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                    |                                                           |
|------------------------------------|-----------------------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                                |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the image is not a <a href="#">vx_image</a> .          |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | If any of the other parameters are incorrect.             |
| <i>VX_ERROR_NOT_SUPPORTED</i>      | If the attribute is not supported on this implementation. |

**vx\_status VX\_API\_CALL vxSetImageAttribute ( vx\_image *image*, vx\_enum *attribute*, const void \* *ptr*, vx\_size *size* )**

Allows setting attributes on the image.

**Parameters**

|    |                  |                                                                               |
|----|------------------|-------------------------------------------------------------------------------|
| in | <i>image</i>     | The reference to the image on which to set the attribute.                     |
| in | <i>attribute</i> | The attribute to set. Use a <a href="#">vx_image_attribute_e</a> enumeration. |
| in | <i>ptr</i>       | The pointer to the location from which to read the value.                     |
| in | <i>size</i>      | The size in bytes of the object pointed to by <i>ptr</i> .                    |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                    |                                                  |
|------------------------------------|--------------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                       |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the image is not a <a href="#">vx_image</a> . |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | If any of the other parameters are incorrect.    |

**vx\_status VX\_API\_CALL vxReleaseImage ( vx\_image \* *image* )**

Releases a reference to an image object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |              |                                      |
|----|--------------|--------------------------------------|
| in | <i>image</i> | The pointer to the image to release. |
|----|--------------|--------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                   |                                              |
|-----------------------------------|----------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                   |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If image is not a <a href="#">vx_image</a> . |

**vx\_size VX\_API\_CALL vxComputeImagePatchSize ( vx\_image *image*, const vx\_rectangle\_t \* *rect*, vx\_uint32 *plane\_index* )**

This computes the size needed to retrieve an image patch from an image.

## Parameters

|    |                    |                                                                                                         |
|----|--------------------|---------------------------------------------------------------------------------------------------------|
| in | <i>image</i>       | The reference to the image from which to extract the patch.                                             |
| in | <i>rect</i>        | The coordinates. Must be 0 <= start < end <= dimension where dimension is width for x and height for y. |
| in | <i>plane_index</i> | The plane index from which to get the data.                                                             |

## Returns

vx\_size

**vx\_status VX\_API\_CALL vxAccessImagePatch ( vx\_image *image*, const vx\_rectangle\_t \* *rect*, vx\_uint32 *plane\_index*, vx\_imagepatch\_addressing\_t \* *addr*, void \*\* *ptr*, vx\_enum *usage* )**

Allows the User to extract a rectangular patch (subset) of an image from a single plane.

## Parameters

|         |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in      | <i>image</i>       | The reference to the image from which to extract the patch.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| in      | <i>rect</i>        | The coordinates from which to get the patch. Must be 0 <= start < end.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| in      | <i>plane_index</i> | The plane index from which to get the data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| in, out | <i>addr</i>        | <p>A pointer to a structure describing the addressing information of the image patch to accessed.</p> <ul style="list-style-type: none"> <li>• Input case: ptr is a pointer to a non-NULL pointer. The addr parameter must be the address of an addressing structure that describes how the user will access the requested image data at address (*ptr).</li> <li>• Output case: ptr is a pointer to a NULL pointer. The function fills the structure pointed by addr with the addressing information that the user must consult to access the pixel data at address (*ptr). If the image being accessed was created via <a href="#">vxCreateImageFromHandle</a>, then the returned addressing information will be the identical to that of the addressing structure provided when <a href="#">vxCreateImageFromHandle</a> was called.</li> </ul> |

|         |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in, out | ptr   | <p>A pointer to a pointer to a location to store the requested data.</p> <ul style="list-style-type: none"> <li>Input case: ptr is a pointer to a non-NULL pointer to a valid pixel buffer. This buffer will be used in one of two ways, depending on the value of the usage parameter. If usage is <code>VX_WRITE_ONLY</code>, then the buffer must contain pixel data that the user wants to replace the image's pixel data with. Otherwise (i.e., usage is not <code>VX_WRITE_ONLY</code>), the image's current pixel data will be written to the memory starting at address (*ptr) as storage memory for the access request. The caller must ensure enough memory has been allocated for the requested patch with the requested addressing. If image was created via <code>vxCreateImageFromHandle</code>, and the pixel buffer pointed to by (*ptr) overlaps the original pixel buffer provided when image was created, then the results of such a call to <code>vxAccessImagePatch</code> are undefined.</li> <li>Output case: ptr is a pointer to a NULL pointer. This NULL pointer will be overwritten with a pointer to the address where the requested data can be accessed. If image was created via <code>vxCreateImageFromHandle</code> then the overwriting pointer must be within the original pixel buffer provided when image was created.</li> <li>(*ptr) must eventually be provided as the ptr parameter of a call to <code>vxCommitImagePatch</code>.</li> </ul> |
| in      | usage | <p>This declares the intended usage of the pointer using the <code>vx_accessor_e</code> enumeration. For uniform images Only <code>VX_READ_ONLY</code> is supported.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

**Note**

The addr and ptr parameters must both be input, or both be output, otherwise the behavior is undefined.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                          |                                                                                                                 |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>VX_ERROR_OPTIMIZED_AWAY</code>     | The reference is a virtual image and cannot be accessed or committed.                                           |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | The <i>start</i> , <i>end</i> , <i>plane_index</i> , <i>stride_x</i> , or <i>stride_y</i> pointer is incorrect. |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | The image reference is not actually an image reference.                                                         |

**Note**

The user may ask for data outside the bounds of the valid region, but such data has an undefined value. Users must be cautious to prevent passing in *uninitialized* pointers or addresses of uninitialized pointers to this function.

**Precondition**

`vxComputeImagePatchSize` if users wish to allocate their own memory.

**Postcondition**

`vxCommitImagePatch` with same (\*ptr) value.

```

/*
 * Copyright (c) 2013-2014 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including

```

```

* without limitation the rights to use, copy, modify, merge, publish,
* distribute, sublicense, and/or sell copies of the Materials, and to
* permit persons to whom the Materials are furnished to do so, subject to
* the following conditions:
*
* The above copyright notice and this permission notice shall be included
* in all copies or substantial portions of the Materials.
*
* THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
* IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
* CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
* TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
*/
#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
 vx_status status = VX_SUCCESS;
 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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x*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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y+addr.scale_y) /
 VX_SCALE_UNIT) +
 ((addr.stride_x*x+addr.scale_x) /
 VX_SCALE_UNIT);
 tmp[i] = pixel;
 }
 }

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

 /* this commits the data back to the image. If rect were 0 or empty, it

```

```

 * would just decrement the reference (used when reading an image only)
 */
 status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
 }
 vxReleaseImage(&image);
 return status;
}

```

**vx\_status VX\_API\_CALL vxCommitImagePatch ( vx\_image image, vx\_rectangle\_t \* rect, vx\_uint32 plane\_index, vx\_imagepatch\_addressing\_t \* addr, const void \* ptr )**

This allows the User to commit a rectangular patch (subset) of an image from a single plane.

#### Parameters

|    |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>image</i>       | The reference to the image from which to extract the patch.                                                                                                                                                                                                                                                                                                                                                                              |
| in | <i>rect</i>        | The coordinates to which to set the patch. Must be $0 \leq \text{start} \leq \text{end}$ . This may be 0 or a rectangle of zero area in order to indicate that the commit must only decrement the reference count.                                                                                                                                                                                                                       |
| in | <i>plane_index</i> | The plane index to which to set the data.                                                                                                                                                                                                                                                                                                                                                                                                |
| in | <i>addr</i>        | The addressing information for the image patch.                                                                                                                                                                                                                                                                                                                                                                                          |
| in | <i>ptr</i>         | A pointer to a pixel buffer to be committed. If the user previously provided a pointer to this buffer to <code>vxAccessImagePatch</code> , the buffer can be freed or re-used after <code>vxCommitImagePatch</code> completes. If the pointer was returned by <code>vxAccessImagePatch</code> , reads or writes to the location pointed by <i>ptr</i> after <code>vxCommitImagePatch</code> completes will result in undefined behavior. |

#### Returns

A `vx_status_e` enumeration.

#### Return values

|                                          |                                                                                                                 |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>VX_ERROR_OPTIMIZED_AWAY</code>     | The reference is a virtual image and cannot be accessed or committed.                                           |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | The <i>start</i> , <i>end</i> , <i>plane_index</i> , <i>stride_x</i> , or <i>stride_y</i> pointer is incorrect. |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | The image reference is not actually an image reference.                                                         |

```

/*
 * Copyright (c) 2013-2014 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 *
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
 vx_status status = VX_SUCCESS;
 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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x*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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y*addr.scale_y) /
 VX_SCALE_UNITY) +
 ((addr.stride_x*x*addr.scale_x) /
 VX_SCALE_UNITY);
 tmp[i] = pixel;
 }
 }

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

 /* this commits the data back to the image. If rect were 0 or empty, it
 * would just decrement the reference (used when reading an image only)
 */
 status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
}
vxReleaseImage(&image);
return status;
}

```

**Note**

If the implementation gives the client a pointer from `vxAccessImagePatch` then implementation-specific behavior may occur. If not, then a copy occurs from the users pointer to the internal data of the object.

If the rectangle intersects bounds of the current valid region, the valid region grows to the union of the two rectangles as long as they occur within the bounds of the original image dimensions.

```
void* VX_API_CALL vxFormatImagePatchAddress1d (void * ptr, vx_uint32 index, const
vx_imagepatch_addressing_t * addr)
```

Accesses a specific indexed pixel in an image patch.

**Parameters**

|    |              |                                                                                                                         |
|----|--------------|-------------------------------------------------------------------------------------------------------------------------|
| in | <i>ptr</i>   | The base pointer of the patch as returned from <a href="#">vxAccessImagePatch</a> .                                     |
| in | <i>index</i> | The 0 based index of the pixel count in the patch. Indexes increase horizontally by 1 then wrap around to the next row. |
| in | <i>addr</i>  | The pointer to the addressing mode information returned from <a href="#">vxAccessImagePatch</a> .                       |

**Returns**

**void \*** Returns the pointer to the specified pixel.

**Precondition**[vxAccessImagePatch](#)

```

/*
 * Copyright (c) 2013-2014 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 *
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.
 */
#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
 vx_status status = VX_SUCCESS;
 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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x*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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y*addr.scale_y) /
 VX_SCALE_UNITY) +
 ((addr.stride_x*x*addr.scale_x) /
 VX_SCALE_UNITY);
 tmp[i] = pixel;
 }
 }

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

 /* this commits the data back to the image. If rect were 0 or empty, it
 * would just decrement the reference (used when reading an image only)
 */
 status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
}
vxReleaseImage(&image);
return status;
}

```

**void\* VX\_API\_CALL vxFormatImagePatchAddress2d ( void \* ptr, vx\_uint32 x, vx\_uint32 y, const vx\_imagepatch\_addressing\_t \* addr )**

Accesses a specific pixel at a 2d coordinate in an image patch.

Parameters

|    |      |                                                                                                   |
|----|------|---------------------------------------------------------------------------------------------------|
| in | ptr  | The base pointer of the patch as returned from <a href="#">vxAccessImagePatch</a> .               |
| in | x    | The x dimension within the patch.                                                                 |
| in | y    | The y dimension within the patch.                                                                 |
| in | addr | The pointer to the addressing mode information returned from <a href="#">vxAccessImagePatch</a> . |

Returns

void \* Returns the pointer to the specified pixel.

Precondition

[vxAccessImagePatch](#)

```

/*
 * Copyright (c) 2013-2014 The Khronos Group Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and/or associated documentation files (the
 * "Materials"), to deal in the Materials without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Materials, and to
 * permit persons to whom the Materials are furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Materials.
 *
 * THE MATERIALS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.

```

```

*/
#include <VX/vx.h>

#define PATCH_DIM 16

vx_status example_imagepatch(vx_context context)
{
 vx_status status = VX_SUCCESS;
 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;

 /* a couple addressing options */

 /* use linear addressing function/macro */
 for (i = 0; i < addr.dim_x+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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_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+=addr.step_y) {
 for (x = 0; x < addr.dim_x; x+=addr.step_x) {
 vx_uint8 *tmp = (vx_uint8 *)base_ptr;
 i = ((addr.stride_y*y+addr.scale_y) /
 VX_SCALE_UNITY) +
 ((addr.stride_x*x+addr.scale_x) /
 VX_SCALE_UNITY);
 tmp[i] = pixel;
 }
 }

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

 /* this commits the data back to the image. If rect were 0 or empty, it
 * would just decrement the reference (used when reading an image only)
 */
 status = vxCommitImagePatch(image, &rect, plane, &addr, base_ptr);
 }
 vxReleaseImage(&image);
 return status;
}

```

**vx\_status VX\_API\_CALL vxGetValidRegionImage ( vx\_image *image*, vx\_rectangle\_t \* *rect* )**

Retrieves the valid region of the image as a rectangle.

After the image is allocated but has not been written to this returns the full rectangle of the image so that functions do not have to manage a case for uninitialized data. The image still retains an uninitialized value, but once the image is written to via any means such as [vxCommitImagePatch](#), the valid region is altered to contain the maximum bounds of the written area.

**Parameters**

|                  |              |                                                    |
|------------------|--------------|----------------------------------------------------|
| <code>in</code>  | <i>image</i> | The image from which to retrieve the valid region. |
| <code>out</code> | <i>rect</i>  | The destination rectangle.                         |

**Returns**

`vx_status`

**Return values**

|                                          |                |
|------------------------------------------|----------------|
| <code>VX_ERROR_INVALID_REFERENCE</code>  | Invalid image. |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | Invalid rect.  |
| <code>VX_SUCCESS</code>                  | Valid image.   |

**Note**

This rectangle can be passed directly to [vxAccessImagePatch](#) to get the full valid region of the image. Modifications from [vxCommitImagePatch](#) grows the valid region.

## 3.52 Object: LUT

### 3.52.1 Detailed Description

Defines the Look-Up Table Interface.

A lookup table is an array that simplifies run-time computation by replacing computation with a simpler array indexing operation.

#### Typedefs

- typedef struct \_vx\_lut \* [vx\\_lut](#)  
*The Look-Up Table (LUT) Object.*

#### Enumerations

- enum [vx\\_lut\\_attribute\\_e](#) {  
[VX\\_LUT\\_ATTRIBUTE\\_TYPE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_LUT << 8)) + 0x0,  
[VX\\_LUT\\_ATTRIBUTE\\_COUNT](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_LUT << 8)) + 0x1,  
[VX\\_LUT\\_ATTRIBUTE\\_SIZE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_LUT << 8)) + 0x2 }  
*The Look-Up Table (LUT) attribute list.*

#### Functions

- [vx\\_status](#) VX\_API\_CALL [vxAccessLUT](#) ([vx\\_lut](#) lut, void \*\*ptr, [vx\\_enum](#) usage)  
*Grants access to a LUT table and increments the object reference count in case of success.*
- [vx\\_status](#) VX\_API\_CALL [vxCommitLUT](#) ([vx\\_lut](#) lut, const void \*ptr)  
*Commits the Lookup Table and decrements the object reference count in case of success.*
- [vx\\_lut](#) VX\_API\_CALL [vxCreateLUT](#) ([vx\\_context](#) context, [vx\\_enum](#) data\_type, [vx\\_size](#) count)  
*Creates LUT object of a given type.*
- [vx\\_status](#) VX\_API\_CALL [vxQueryLUT](#) ([vx\\_lut](#) lut, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries attributes from a LUT.*
- [vx\\_status](#) VX\_API\_CALL [vxReleaseLUT](#) ([vx\\_lut](#) \*lut)  
*Releases a reference to a LUT object. The object may not be garbage collected until its total reference count is zero.*

### 3.52.2 Enumeration Type Documentation

**enum [vx\\_lut\\_attribute\\_e](#)**

The Look-Up Table (LUT) attribute list.

Enumerator

**[VX\\_LUT\\_ATTRIBUTE\\_TYPE](#)** Indicates the value type of the LUT. Use a [vx\\_enum](#).

**[VX\\_LUT\\_ATTRIBUTE\\_COUNT](#)** Indicates the number of elements in the LUT. Use a [vx\\_size](#).

**[VX\\_LUT\\_ATTRIBUTE\\_SIZE](#)** Indicates the total size of the LUT in bytes. Uses a [vx\\_size](#).

Definition at line 842 of file [vx\\_types.h](#).

### 3.52.3 Function Documentation

**[vx\\_lut](#) VX\_API\_CALL [vxCreateLUT](#) ( [vx\\_context](#) context, [vx\\_enum](#) data\_type, [vx\\_size](#) count )**

Creates LUT object of a given type.

**Parameters**

|    |                  |                                     |
|----|------------------|-------------------------------------|
| in | <i>context</i>   | The reference to the context.       |
| in | <i>data_type</i> | The type of data stored in the LUT. |
| in | <i>count</i>     | The number of entries desired.      |

**Note**

For OpenVX 1.0, count must be equal to 256 and data\_type can only be [VX\\_TYPE\\_UINT8](#).

**Returns**

An LUT reference [vx\\_lut](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_status VX\_API\_CALL vxReleaseLUT ( vx\_lut \* lut )**

Releases a reference to a LUT object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |            |                                    |
|----|------------|------------------------------------|
| in | <i>lut</i> | The pointer to the LUT to release. |
|----|------------|------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                   |                                          |
|-----------------------------------|------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                               |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If lut is not a <a href="#">vx_lut</a> . |

**vx\_status VX\_API\_CALL vxQueryLUT ( vx\_lut lut, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries attributes from a LUT.

**Parameters**

|     |                  |                                                                               |
|-----|------------------|-------------------------------------------------------------------------------|
| in  | <i>lut</i>       | The LUT to query.                                                             |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_lut_attribute_e</a> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                           |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.                |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxAccessLUT ( vx\_lut lut, void \*\* ptr, vx\_enum usage )**

Grants access to a LUT table and increments the object reference count in case of success.

There are several variations of call methodology:

- If *ptr* is NULL (which means the current data of the LUT is not desired), the LUT reference count is incremented.
- If *ptr* is not NULL but (\*ptr) is NULL, (\*ptr) will contain the address of the LUT data when the function returns and the reference count will be incremented. Whether the (\*ptr) address is mapped or allocated is undefined. (\*ptr) must be returned to [vxCommitLUT](#).



- If *ptr* is not NULL and (\*ptr) is not NULL, the user is signalling the implementation to copy the LUT data into the location specified by (\*ptr). Users must use `vxQueryLUT` with `VX_LUT_ATTRIBUTE_SIZE` to determine how much memory to allocate for the LUT data.

In any case, `vxCommitLUT` must be called after LUT access is complete.

**Parameters**

|                      |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>in</code>      | <code>lut</code>   | The LUT from which to get the data.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <code>in, out</code> | <code>ptr</code>   | <p>ptr The user-supplied address to a pointer, via which the requested contents are returned.</p> <ul style="list-style-type: none"> <li>• If ptr is NULL, an error occurs.</li> <li>• If ptr is not NULL and (*ptr) is NULL, (*ptr) will be set to the address of a memory area managed by the OpenVX framework containing the requested data.</li> <li>• If both ptr and (*ptr) are not NULL, requested data will be copied to (*ptr) (optionally in case of write-only access).</li> </ul> |
| <code>in</code>      | <code>usage</code> | This declares the intended usage of the pointer using the * <code>vx_accessor_e</code> enumeration.                                                                                                                                                                                                                                                                                                                                                                                           |

**Returns**

A `vx_status_e` enumeration.

**Postcondition**

`vxCommitLUT`

**`vx_status VX_API_CALL vxCommitLUT ( vx_lut lut, const void * ptr )`**

Commits the Lookup Table and decrements the object reference count in case of success.

Commits the data back to the LUT object and decrements the reference count. There are several variations of call methodology:

- If a user should allocated their own memory for the LUT data copy, the user is obligated to free this memory.
- If `ptr` is not NULL and the (\*ptr) for `vxAccessLUT` was NULL, it is undefined whether the implementation will unmap or copy and free the memory.

**Parameters**

|                 |                  |                                                                                     |
|-----------------|------------------|-------------------------------------------------------------------------------------|
| <code>in</code> | <code>lut</code> | The LUT to modify.                                                                  |
| <code>in</code> | <code>ptr</code> | The pointer provided or returned by <code>vxAccessLUT</code> . This cannot be NULL. |

**Returns**

A `vx_status_e` enumeration.

**Precondition**

`vxAccessLUT`.

## 3.53 Object: Matrix

### 3.53.1 Detailed Description

Defines the Matrix Object Interface.

#### Typedefs

- typedef struct \_vx\_matrix \* [vx\\_matrix](#)  
*The Matrix Object. An MxN matrix of some unit type.*

#### Enumerations

- enum [vx\\_matrix\\_attribute\\_e](#) {  
[VX\\_MATRIX\\_ATTRIBUTE\\_TYPE](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_MATRIX](#) << 8)) + 0x0,  
[VX\\_MATRIX\\_ATTRIBUTE\\_ROWS](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_MATRIX](#) << 8)) + 0x1,  
[VX\\_MATRIX\\_ATTRIBUTE\\_COLUMNS](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_MATRIX](#) << 8)) + 0x2,  
[VX\\_MATRIX\\_ATTRIBUTE\\_SIZE](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_MATRIX](#) << 8)) + 0x3 }  
*The matrix attributes.*

#### Functions

- [vx\\_matrix](#) [VX\\_API\\_CALL](#) [vxCreateMatrix](#) ([vx\\_context](#) c, [vx\\_enum](#) data\_type, [vx\\_size](#) columns, [vx\\_size](#) rows)  
*Creates a reference to a matrix object.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxQueryMatrix](#) ([vx\\_matrix](#) mat, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries an attribute on the matrix object.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxReadMatrix](#) ([vx\\_matrix](#) mat, void \*array)  
*Gets the matrix data (copy).*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxReleaseMatrix](#) ([vx\\_matrix](#) \*mat)  
*Releases a reference to a matrix object. The object may not be garbage collected until its total reference count is zero.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxWriteMatrix](#) ([vx\\_matrix](#) mat, const void \*array)  
*Sets the matrix data (copy)*

### 3.53.2 Enumeration Type Documentation

#### enum [vx\\_matrix\\_attribute\\_e](#)

The matrix attributes.

Enumerator

- [VX\\_MATRIX\\_ATTRIBUTE\\_TYPE](#)** The value type of the matrix. Use a [vx\\_enum](#) parameter.
- [VX\\_MATRIX\\_ATTRIBUTE\\_ROWS](#)** The M dimension of the matrix. Use a [vx\\_size](#) parameter.
- [VX\\_MATRIX\\_ATTRIBUTE\\_COLUMNS](#)** The N dimension of the matrix. Use a [vx\\_size](#) parameter.
- [VX\\_MATRIX\\_ATTRIBUTE\\_SIZE](#)** The total size of the matrix in bytes. Use a [vx\\_size](#) parameter.

Definition at line 902 of file [vx\\_types.h](#).

### 3.53.3 Function Documentation

[vx\\_matrix](#) [VX\\_API\\_CALL](#) [vxCreateMatrix](#) ( [vx\\_context](#) c, [vx\\_enum](#) data\_type, [vx\\_size](#) columns, [vx\\_size](#) rows )

Creates a reference to a matrix object.

**Parameters**

|    |                  |                                                                                                   |
|----|------------------|---------------------------------------------------------------------------------------------------|
| in | <i>c</i>         | The reference to the overall context.                                                             |
| in | <i>data_type</i> | The unit format of the matrix. <a href="#">VX_TYPE_INT32</a> or <a href="#">VX_TYPE_FLOAT32</a> . |
| in | <i>columns</i>   | The first dimensionality.                                                                         |
| in | <i>rows</i>      | The second dimensionality.                                                                        |

**Returns**

An matrix reference [vx\\_matrix](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_status VX\_API\_CALL vxReleaseMatrix ( vx\_matrix \* mat )**

Releases a reference to a matrix object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |            |                                  |
|----|------------|----------------------------------|
| in | <i>mat</i> | The matrix reference to release. |
|----|------------|----------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                   |                                                    |
|-----------------------------------|----------------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                         |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If <i>mat</i> is not a <a href="#">vx_matrix</a> . |

**vx\_status VX\_API\_CALL vxQueryMatrix ( vx\_matrix mat, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries an attribute on the matrix object.

**Parameters**

|     |                  |                                                                                  |
|-----|------------------|----------------------------------------------------------------------------------|
| in  | <i>mat</i>       | The matrix object to set.                                                        |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_matrix_attribute_e</a> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                              |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.                   |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxReadMatrix ( vx\_matrix mat, void \* array )**

Gets the matrix data (copy).

**Parameters**

|     |              |                                         |
|-----|--------------|-----------------------------------------|
| in  | <i>mat</i>   | The reference to the matrix.            |
| out | <i>array</i> | The array in which to place the matrix. |

**See also**

[vxQueryMatrix](#) and [VX\\_MATRIX\\_ATTRIBUTE\\_COLUMNS](#) and [VX\\_MATRIX\\_ATTRIBUTE\\_ROWS](#) to get the needed number of elements of the array.

## Returns

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxWriteMatrix ( vx\_matrix *mat*, const void \* *array* )**

Sets the matrix data (copy)

## Parameters

|    |              |                                                |
|----|--------------|------------------------------------------------|
| in | <i>mat</i>   | The reference to the matrix.                   |
| in | <i>array</i> | The array containing the matrix to be written. |

## See also

[vxQueryMatrix](#) and [VX\\_MATRIX\\_ATTRIBUTE\\_COLUMNS](#) and [VX\\_MATRIX\\_ATTRIBUTE\\_ROWS](#) to get the needed number of elements of the array.'

## Returns

A [vx\\_status\\_e](#) enumeration.

## 3.54 Object: Pyramid

### 3.54.1 Detailed Description

Defines the Image Pyramid Object Interface.

A Pyramid object in OpenVX represents a collection of related images. Typically, these images are created by either downscaling or upscaling a *base image*, contained in level zero of the pyramid. Successive levels of the pyramid increase or decrease in size by a factor given by the `VX_PYRAMID_ATTRIBUTE_SCALE` attribute. For instance, in a pyramid with 3 levels and `VX_SCALE_PYRAMID_HALF`, the level one image is one-half the width and one-half the height of the level zero image, and the level two image is one-quarter the width and one quarter the height of the level zero image. When downscaling or upscaling results in a non-integral number of pixels at any level, fractional pixels always get rounded up to the nearest integer. (E.g., a 3-level image pyramid beginning with level zero having a width of 9 and a scaling of `VX_SCALE_PYRAMID_HALF` results in the level one image with a width of  $5 = \text{ceil}(9 * 0.5)$  and a level two image with a width of  $3 = \text{ceil}(5 * 0.5)$ ). Position  $(r_N, c_N)$  at level  $N$  corresponds to position  $(r_{N-1}/\text{scale}, c_{N-1}/\text{scale})$  at level  $N - 1$ .

### Macros

- `#define VX_SCALE_PYRAMID_HALF (0.5f)`  
*Use to indicate a half-scale pyramid.*
- `#define VX_SCALE_PYRAMID_ORB ((vx_float32)0.8408964f)`  
*Use to indicate a ORB scaled pyramid whose scaling factor is  $\frac{1}{\sqrt{2}}$ .*

### Typedefs

- `typedef struct _vx_pyramid * vx_pyramid`  
*The Image Pyramid object. A set of scaled images.*

### Enumerations

- `enum vx_pyramid_attribute_e {`  
`VX_PYRAMID_ATTRIBUTE_LEVELS = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x0,`  
`VX_PYRAMID_ATTRIBUTE_SCALE = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x1,`  
`VX_PYRAMID_ATTRIBUTE_WIDTH = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x2,`  
`VX_PYRAMID_ATTRIBUTE_HEIGHT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x3,`  
`VX_PYRAMID_ATTRIBUTE_FORMAT = ((( VX_ID_KHRONOS ) << 20) | ( VX_TYPE_PYRAMID << 8)) + 0x4 }`  
*The pyramid object attributes.*

### Functions

- `vx_pyramid VX_API_CALL vxCreatePyramid (vx_context context, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format)`  
*Creates a reference to a pyramid object of the supplied number of levels.*
- `vx_pyramid VX_API_CALL vxCreateVirtualPyramid (vx_graph graph, vx_size levels, vx_float32 scale, vx_uint32 width, vx_uint32 height, vx_df_image format)`  
*Creates a reference to a virtual pyramid object of the supplied number of levels.*
- `vx_image VX_API_CALL vxGetPyramidLevel (vx_pyramid pyr, vx_uint32 index)`  
*Retrieves a level of the pyramid as a `vx_image`, which can be used elsewhere in OpenVX. A call to `vxReleaseImage` is necessary to release an image for each call of `vxGetPyramidLevel`.*
- `vx_status VX_API_CALL vxQueryPyramid (vx_pyramid pyr, vx_enum attribute, void *ptr, vx_size size)`  
*Queries an attribute from an image pyramid.*

- **vx\_status** **VX\_API\_CALL** **vxReleasePyramid** (**vx\_pyramid** \*pyr)

Releases a reference to a pyramid object. The object may not be garbage collected until its total reference count is zero.

### 3.54.2 Enumeration Type Documentation

#### enum vx\_pyramid\_attribute\_e

The pyramid object attributes.

Enumerator

**VX\_PYRAMID\_ATTRIBUTE\_LEVELS** The number of levels of the pyramid. Use a **vx\_size** parameter.

**VX\_PYRAMID\_ATTRIBUTE\_SCALE** The scale factor between each level of the pyramid. Use a **vx\_float32** parameter.

**VX\_PYRAMID\_ATTRIBUTE\_WIDTH** The width of the 0th image in pixels. Use a **vx\_uint32** parameter.

**VX\_PYRAMID\_ATTRIBUTE\_HEIGHT** The height of the 0th image in pixels. Use a **vx\_uint32** parameter.

**VX\_PYRAMID\_ATTRIBUTE\_FORMAT** The **vx\_df\_image\_e** format of the image. Use a **vx\_df\_image** parameter.

Definition at line 934 of file **vx\_types.h**.

### 3.54.3 Function Documentation

**vx\_pyramid** **VX\_API\_CALL** **vxCreatePyramid** ( **vx\_context** context, **vx\_size** levels, **vx\_float32** scale, **vx\_uint32** width, **vx\_uint32** height, **vx\_df\_image** format )

Creates a reference to a pyramid object of the supplied number of levels.

Parameters

|    |         |                                                                                                                                                                                                                   |
|----|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | context | The reference to the overall context.                                                                                                                                                                             |
| in | levels  | The number of levels desired. This is required to be a non-zero value.                                                                                                                                            |
| in | scale   | Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. In OpenVX 1.0, the only permissible values are <b>VX_SCALE_PYRAMID_HALF</b> or <b>VX_SCALE_PYRAMID_ORB</b> . |
| in | width   | The width of the 0th level image in pixels.                                                                                                                                                                       |
| in | height  | The height of the 0th level image in pixels.                                                                                                                                                                      |
| in | format  | The format of all images in the pyramid. NV12, NV21, IYUV, UYVY and YUYV formats are not supported.                                                                                                               |

Returns

A pyramid reference **vx\_pyramid** to the sub-image. Any possible errors preventing a successful creation should be checked using **vxGetStatus**.

**vx\_pyramid** **VX\_API\_CALL** **vxCreateVirtualPyramid** ( **vx\_graph** graph, **vx\_size** levels, **vx\_float32** scale, **vx\_uint32** width, **vx\_uint32** height, **vx\_df\_image** format )

Creates a reference to a virtual pyramid object of the supplied number of levels.

Virtual Pyramids can be used to connect Nodes together when the contents of the pyramids will not be accessed by the user of the API. All of the following constructions are valid:

```
vx_context context = vxCreateContext();
vx_graph graph = vxCreateGraph(context);
vx_pyramid virt[] = {
 vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 0, 0
 , VX_DF_IMAGE_VIRT), // no dimension and format specified for level 0
 vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640,
 480, VX_DF_IMAGE_VIRT), // no format specified.
 vxCreateVirtualPyramid(graph, 4, VX_SCALE_PYRAMID_HALF, 640,
 480, VX_DF_IMAGE_U8), // no access
};
```

**Parameters**

|    |               |                                                                                                                                                                                                                                     |
|----|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>graph</i>  | The reference to the parent graph.                                                                                                                                                                                                  |
| in | <i>levels</i> | The number of levels desired. This is required to be a non-zero value.                                                                                                                                                              |
| in | <i>scale</i>  | Used to indicate the scale between pyramid levels. This is required to be a non-zero positive value. In OpenVX 1.0, the only permissible values are <a href="#">VX_SCALE_PYRAMID_HALF</a> or <a href="#">VX_SCALE_PYRAMID_ORB</a> . |
| in | <i>width</i>  | The width of the 0th level image in pixels. This may be set to zero to indicate to the interface that the value is unspecified.                                                                                                     |
| in | <i>height</i> | The height of the 0th level image in pixels. This may be set to zero to indicate to the interface that the value is unspecified.                                                                                                    |
| in | <i>format</i> | The format of all images in the pyramid. This may be set to <a href="#">VX_DF_IMAGE_VIRT</a> to indicate that the format is unspecified.                                                                                            |

**Returns**

A pyramid reference [vx\\_pyramid](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**Note**

Images extracted with [vxGetPyramidLevel](#) behave as Virtual Images and cause [vxAccessImagePatch](#) to return errors.

**vx\_status VX\_API\_CALL vxReleasePyramid ( vx\_pyramid \* pyr )**

Releases a reference to a pyramid object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |            |                                        |
|----|------------|----------------------------------------|
| in | <i>pyr</i> | The pointer to the pyramid to release. |
|----|------------|----------------------------------------|

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                            |                                              |
|--------------------------------------------|----------------------------------------------|
| <a href="#">VX_SUCCESS</a>                 | No errors.                                   |
| <a href="#">VX_ERROR_INVALID_REFERENCE</a> | If pyr is not a <a href="#">vx_pyramid</a> . |

**Postcondition**

After returning from this function the reference is zeroed.

**vx\_status VX\_API\_CALL vxQueryPyramid ( vx\_pyramid pyr, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries an attribute from an image pyramid.

**Parameters**

|    |                  |                                                                                             |
|----|------------------|---------------------------------------------------------------------------------------------|
| in | <i>pyr</i>       | The pyramid to query.                                                                       |
| in | <i>attribute</i> | The attribute for which to query. Use a <a href="#">vx_pyramid_attribute_e</a> enumeration. |



|     |             |                                                                |
|-----|-------------|----------------------------------------------------------------|
| out | <i>ptr</i>  | The location at which to store the resulting value.            |
| in  | <i>size</i> | The size in bytes of the container to which <i>ptr</i> points. |

**Returns**

A `vx_status_e` enumeration.

**vx\_image VX\_API\_CALL vxGetPyramidLevel ( vx\_pyramid *pyr*, vx\_uint32 *index* )**

Retrieves a level of the pyramid as a `vx_image`, which can be used elsewhere in OpenVX. A call to `vxReleaseImage` is necessary to release an image for each call of `vxGetPyramidLevel`.

**Parameters**

|    |              |                                                              |
|----|--------------|--------------------------------------------------------------|
| in | <i>pyr</i>   | The pyramid object.                                          |
| in | <i>index</i> | The index of the level, such that index is less than levels. |

**Returns**

A `vx_image` reference.

**Return values**

|   |                                                    |
|---|----------------------------------------------------|
| 0 | Indicates that the index or the object is invalid. |
|---|----------------------------------------------------|

## 3.55 Object: Remap

### 3.55.1 Detailed Description

Defines the Remap Object Interface.

#### Typedefs

- typedef struct \_vx\_remap \* [vx\\_remap](#)

*The remap table Object. A remap table contains per-pixel mapping of output pixels to input pixels.*

#### Enumerations

- enum [vx\\_remap\\_attribute\\_e](#) {  
[VX\\_REMAP\\_ATTRIBUTE\\_SOURCE\\_WIDTH](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REMAP](#) << 8)) + 0x0,  
[VX\\_REMAP\\_ATTRIBUTE\\_SOURCE\\_HEIGHT](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REMAP](#) << 8)) + 0x1,  
[VX\\_REMAP\\_ATTRIBUTE\\_DESTINATION\\_WIDTH](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REMAP](#) << 8)) + 0x2,  
[VX\\_REMAP\\_ATTRIBUTE\\_DESTINATION\\_HEIGHT](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_REMAP](#) << 8)) + 0x3 }

*The remap object attributes.*

#### Functions

- [vx\\_remap](#) [VX\\_API\\_CALL](#) [vxCreateRemap](#) ([vx\\_context](#) context, [vx\\_uint32](#) src\_width, [vx\\_uint32](#) src\_height, [vx\\_uint32](#) dst\_width, [vx\\_uint32](#) dst\_height)  
*Creates a remap table object.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxGetRemapPoint](#) ([vx\\_remap](#) table, [vx\\_uint32](#) dst\_x, [vx\\_uint32](#) dst\_y, [vx\\_float32](#) \*src\_x, [vx\\_float32](#) \*src\_y)  
*Retrieves the source pixel point from a destination pixel.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxQueryRemap](#) ([vx\\_remap](#) r, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries attributes from a Remap table.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxReleaseRemap](#) ([vx\\_remap](#) \*table)  
*Releases a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxSetRemapPoint](#) ([vx\\_remap](#) table, [vx\\_uint32](#) dst\_x, [vx\\_uint32](#) dst\_y, [vx\\_float32](#) src\_x, [vx\\_float32](#) src\_y)  
*Assigns a destination pixel mapping to the source pixel.*

### 3.55.2 Enumeration Type Documentation

**enum [vx\\_remap\\_attribute\\_e](#)**

The remap object attributes.

Enumerator

**[VX\\_REMAP\\_ATTRIBUTE\\_SOURCE\\_WIDTH](#)** The source width. Use a [vx\\_uint32](#) parameter.

**[VX\\_REMAP\\_ATTRIBUTE\\_SOURCE\\_HEIGHT](#)** The source height. Use a [vx\\_uint32](#) parameter.

**[VX\\_REMAP\\_ATTRIBUTE\\_DESTINATION\\_WIDTH](#)** The destination width. Use a [vx\\_uint32](#) parameter.

**[VX\\_REMAP\\_ATTRIBUTE\\_DESTINATION\\_HEIGHT](#)** The destination height. Use a [vx\\_uint32](#) parameter.

Definition at line 950 of file [vx\\_types.h](#).

### 3.55.3 Function Documentation

**vx\_remap VX\_API\_CALL vxCreateRemap ( vx\_context *context*, vx\_uint32 *src\_width*, vx\_uint32 *src\_height*, vx\_uint32 *dst\_width*, vx\_uint32 *dst\_height* )**

Creates a remap table object.

**Parameters**

|    |                   |                                            |
|----|-------------------|--------------------------------------------|
| in | <i>context</i>    | The reference to the overall context.      |
| in | <i>src_width</i>  | Width of the source image in pixel.        |
| in | <i>src_height</i> | Height of the source image in pixels.      |
| in | <i>dst_width</i>  | Width of the destination image in pixels.  |
| in | <i>dst_height</i> | Height of the destination image in pixels. |

**Returns**

A remap reference `vx_remap`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx\_status VX\_API\_CALL vxReleaseRemap ( vx\_remap \* table )**

Releases a reference to a remap table object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |              |                                            |
|----|--------------|--------------------------------------------|
| in | <i>table</i> | The pointer to the remap table to release. |
|----|--------------|--------------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                   |                                           |
|-----------------------------------|-------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If table is not a <code>vx_remap</code> . |

**vx\_status VX\_API\_CALL vxSetRemapPoint ( vx\_remap table, vx\_uint32 dst\_x, vx\_uint32 dst\_y, vx\_float32 src\_x, vx\_float32 src\_y )**

Assigns a destination pixel mapping to the source pixel.

**Parameters**

|    |              |                                                                         |
|----|--------------|-------------------------------------------------------------------------|
| in | <i>table</i> | The remap table reference.                                              |
| in | <i>dst_x</i> | The destination x coordinate.                                           |
| in | <i>dst_y</i> | The destination y coordinate.                                           |
| in | <i>src_x</i> | The source x coordinate in float representation to allow interpolation. |
| in | <i>src_y</i> | The source y coordinate in float representation to allow interpolation. |

**Returns**

A `vx_status_e` enumeration.

**vx\_status VX\_API\_CALL vxGetRemapPoint ( vx\_remap table, vx\_uint32 dst\_x, vx\_uint32 dst\_y, vx\_float32 \* src\_x, vx\_float32 \* src\_y )**

Retrieves the source pixel point from a destination pixel.

**Parameters**

|     |              |                                                                                                              |
|-----|--------------|--------------------------------------------------------------------------------------------------------------|
| in  | <i>table</i> | The remap table reference.                                                                                   |
| in  | <i>dst_x</i> | The destination x coordinate.                                                                                |
| in  | <i>dst_y</i> | The destination y coordinate.                                                                                |
| out | <i>src_x</i> | The pointer to the location to store the source x coordinate in float representation to allow interpolation. |
| out | <i>src_y</i> | The pointer to the location to store the source y coordinate in float representation to allow interpolation. |

**Returns**

A `vx_status_e` enumeration.

**vx\_status VX\_API\_CALL vxQueryRemap ( vx\_remap *r*, vx\_enum *attribute*, void \* *ptr*, vx\_size *size* )**

Queries attributes from a Remap table.

**Parameters**

|     |                  |                                                                              |
|-----|------------------|------------------------------------------------------------------------------|
| in  | <i>r</i>         | The remap to query.                                                          |
| in  | <i>attribute</i> | The attribute to query. Use a <code>vx_remap_attribute_e</code> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                          |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.               |

**Returns**

A `vx_status_e` enumeration.

## 3.56 Object: Scalar

### 3.56.1 Detailed Description

Defines the Scalar Object interface.

#### Typedefs

- typedef struct \_vx\_scalar \* vx\_scalar  
*An opaque reference to a scalar.*

#### Enumerations

- enum vx\_scalar\_attribute\_e { VX\_SCALAR\_ATTRIBUTE\_TYPE = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ATTRIBUTE\_SCALAR << 8)) + 0x0 }  
*The scalar attributes list.*

#### Functions

- vx\_scalar VX\_API\_CALL vxCreateScalar (vx\_context context, vx\_enum data\_type, const void \*ptr)  
*Creates a reference to a scalar object. Also see [Node Parameters](#).*
- vx\_status VX\_API\_CALL vxQueryScalar (vx\_scalar scalar, vx\_enum attribute, void \*ptr, vx\_size size)  
*Queries attributes from a scalar.*
- vx\_status VX\_API\_CALL vxReadScalarValue (vx\_scalar ref, void \*ptr)  
*Gets the scalar value out of a reference.*
- vx\_status VX\_API\_CALL vxReleaseScalar (vx\_scalar \*scalar)  
*Releases a reference to a scalar object. The object may not be garbage collected until its total reference count is zero.*
- vx\_status VX\_API\_CALL vxWriteScalarValue (vx\_scalar ref, const void \*ptr)  
*Sets the scalar value in a reference.*

### 3.56.2 Typedef Documentation

**typedef struct \_vx\_scalar\* vx\_scalar**

An opaque reference to a scalar.

A scalar can be up to 64 bits wide.

See also

[vxCreateScalar](#)

Definition at line 159 of file [vx\\_types.h](#).

### 3.56.3 Enumeration Type Documentation

**enum vx\_scalar\_attribute\_e**

The scalar attributes list.

Enumerator

**VX\_SCALAR\_ATTRIBUTE\_TYPE** Queries the type of atomic that is contained in the scalar. Use a [vx\\_enum](#) parameter.

Definition at line 820 of file [vx\\_types.h](#).

### 3.56.4 Function Documentation

**vx\_scalar VX\_API\_CALL vxCreateScalar ( vx\_context context, vx\_enum data\_type, const void \* ptr )**

Creates a reference to a scalar object. Also see [Node Parameters](#).

**Parameters**

|    |                  |                                                                                                                                             |
|----|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i>   | The reference to the system context.                                                                                                        |
| in | <i>data_type</i> | The <code>vx_type_e</code> of the scalar. Must be greater than <code>VX_TYPE_INVALID</code> and less than <code>VX_TYPE_SCALAR_MAX</code> . |
| in | <i>ptr</i>       | The pointer to the initial value of the scalar.                                                                                             |

**Returns**

A scalar reference `vx_scalar`. Any possible errors preventing a successful creation should be checked using `vxGetStatus`.

**vx\_status VX\_API\_CALL vxReleaseScalar ( vx\_scalar \* scalar )**

Releases a reference to a scalar object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |               |                                       |
|----|---------------|---------------------------------------|
| in | <i>scalar</i> | The pointer to the scalar to release. |
|----|---------------|---------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                         |                                             |
|-----------------------------------------|---------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                  |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If scalar is not a <code>vx_scalar</code> . |

**vx\_status VX\_API\_CALL vxQueryScalar ( vx\_scalar scalar, vx\_enum attribute, void \* ptr, vx\_size size )**

Queries attributes from a scalar.

**Parameters**

|     |                  |                                                                                 |
|-----|------------------|---------------------------------------------------------------------------------|
| in  | <i>scalar</i>    | The scalar object.                                                              |
| in  | <i>attribute</i> | The enumeration to query. Use a <code>vx_scalar_attribute_e</code> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                             |
| in  | <i>size</i>      | The size of the container to which <i>ptr</i> points.                           |

**Returns**

A `vx_status_e` enumeration.

**vx\_status VX\_API\_CALL vxReadScalarValue ( vx\_scalar ref, void \* ptr )**

Gets the scalar value out of a reference.

**Note**

Use this in conjunction with Query APIs that return references which should be converted into values.

**Parameters**

|     |            |                                                                                           |
|-----|------------|-------------------------------------------------------------------------------------------|
| in  | <i>ref</i> | The reference from which to get the scalar value.                                         |
| out | <i>ptr</i> | An appropriate typed pointer that points to a location to which to copy the scalar value. |

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                          |                                                                         |
|------------------------------------------|-------------------------------------------------------------------------|
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the <i>ref</i> is not a valid reference.                             |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If <i>ptr</i> is NULL.                                                  |
| <code>VX_ERROR_INVALID_TYPE</code>       | If the type does not match the type in the reference or is a bad value. |

**`vx_status VX_API_CALL vxWriteScalarValue ( vx_scalar ref, const void * ptr )`**

Sets the scalar value in a reference.

**Note**

Use this in conjunction with Parameter APIs that return references to parameters that need to be altered.

**Parameters**

|    |            |                                                                                             |
|----|------------|---------------------------------------------------------------------------------------------|
| in | <i>ref</i> | The reference from which to get the scalar value.                                           |
| in | <i>ptr</i> | An appropriately typed pointer that points to a location to which to copy the scalar value. |

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                          |                                                                         |
|------------------------------------------|-------------------------------------------------------------------------|
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the <i>ref</i> is not a valid reference.                             |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If <i>ptr</i> is NULL.                                                  |
| <code>VX_ERROR_INVALID_TYPE</code>       | If the type does not match the type in the reference or is a bad value. |



## 3.57 Object: Threshold

### 3.57.1 Detailed Description

Defines the Threshold Object Interface.

#### Typedefs

- typedef struct \_vx\_threshold \* [vx\\_threshold](#)

*The Threshold Object. A thresholding object contains the types and limit values of the thresholding required.*

#### Enumerations

- enum [vx\\_threshold\\_attribute\\_e](#) {  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_TYPE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x0,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_THRESHOLD\\_VALUE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x1,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_THRESHOLD\\_LOWER](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x2,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_THRESHOLD\\_UPPER](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x3,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_TRUE\\_VALUE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x4,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_FALSE\\_VALUE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x5,  
[VX\\_THRESHOLD\\_ATTRIBUTE\\_DATA\\_TYPE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_TYPE\_THRESHOLD << 8)) + 0x6 }  
*The threshold attributes.*
- enum [vx\\_threshold\\_type\\_e](#) {  
[VX\\_THRESHOLD\\_TYPE\\_BINARY](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_THRESHOLD\_TYPE << 12)) + 0x0,  
[VX\\_THRESHOLD\\_TYPE\\_RANGE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_THRESHOLD\_TYPE << 12)) + 0x1 }  
*The Threshold types.*

#### Functions

- [vx\\_threshold](#) VX\_API\_CALL [vxCreateThreshold](#) ([vx\\_context](#) c, [vx\\_enum](#) thresh\_type, [vx\\_enum](#) data\_type)  
*Creates a reference to a threshold object of a given type.*
- [vx\\_status](#) VX\_API\_CALL [vxQueryThreshold](#) ([vx\\_threshold](#) thresh, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries an attribute on the threshold object.*
- [vx\\_status](#) VX\_API\_CALL [vxReleaseThreshold](#) ([vx\\_threshold](#) \*thresh)  
*Releases a reference to a threshold object. The object may not be garbage collected until its total reference count is zero.*
- [vx\\_status](#) VX\_API\_CALL [vxSetThresholdAttribute](#) ([vx\\_threshold](#) thresh, [vx\\_enum](#) attribute, const void \*ptr, [vx\\_size](#) size)  
*Sets attributes on the threshold object.*

### 3.57.2 Enumeration Type Documentation

enum [vx\\_threshold\\_type\\_e](#)

The Threshold types.

Enumerator

**[VX\\_THRESHOLD\\_TYPE\\_BINARY](#)** A threshold with only 1 value.

**VX\_THRESHOLD\_TYPE\_RANGE** A threshold with 2 values (upper/lower). Use with Canny Edge Detection.

Definition at line 872 of file [vx\\_types.h](#).

#### enum vx\_threshold\_attribute\_e

The threshold attributes.

Enumerator

**VX\_THRESHOLD\_ATTRIBUTE\_TYPE** The value type of the threshold. Use a [vx\\_enum](#) parameter. Will contain a [vx\\_threshold\\_type\\_e](#).

**VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_VALUE** The value of the single threshold. Use a [vx\\_int32](#) parameter.

**VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_LOWER** The value of the lower threshold. Use a [vx\\_int32](#) parameter.

**VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_UPPER** The value of the higher threshold. Use a [vx\\_int32](#) parameter.

**VX\_THRESHOLD\_ATTRIBUTE\_TRUE\_VALUE** The value of the TRUE threshold. Use a [vx\\_int32](#) parameter.

**VX\_THRESHOLD\_ATTRIBUTE\_FALSE\_VALUE** The value of the FALSE threshold. Use a [vx\\_int32](#) parameter.

**VX\_THRESHOLD\_ATTRIBUTE\_DATA\_TYPE** The data type of the threshold's value. Use a [vx\\_enum](#) parameter. Will contain a [vx\\_type\\_e](#).

Definition at line 882 of file [vx\\_types.h](#).

### 3.57.3 Function Documentation

**vx\_threshold VX\_API\_CALL vxCreateThreshold ( vx\_context c, vx\_enum thresh\_type, vx\_enum data\_type )**

Creates a reference to a threshold object of a given type.

Parameters

|    |                    |                                            |
|----|--------------------|--------------------------------------------|
| in | <i>c</i>           | The reference to the overall context.      |
| in | <i>thresh_type</i> | The type of threshold to create.           |
| in | <i>data_type</i>   | The data type of the threshold's value(s). |

Note

For OpenVX 1.0, *data\_type* can only be [VX\\_TYPE\\_UINT8](#).

Returns

An threshold reference [vx\\_threshold](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_status VX\_API\_CALL vxReleaseThreshold ( vx\_threshold \* thresh )**

Releases a reference to a threshold object. The object may not be garbage collected until its total reference count is zero.

Parameters

|    |               |                                          |
|----|---------------|------------------------------------------|
| in | <i>thresh</i> | The pointer to the threshold to release. |
|----|---------------|------------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                         |                                                       |
|-----------------------------------------|-------------------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                            |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If <i>thresh</i> is not a <code>vx_threshold</code> . |

**`vx_status VX_API_CALL vxSetThresholdAttribute ( vx_threshold thresh, vx_enum attribute, const void * ptr, vx_size size )`**

Sets attributes on the threshold object.

**Parameters**

|    |                  |                                                                                   |
|----|------------------|-----------------------------------------------------------------------------------|
| in | <i>thresh</i>    | The threshold object to set.                                                      |
| in | <i>attribute</i> | The attribute to modify. Use a <code>vx_threshold_attribute_e</code> enumeration. |
| in | <i>ptr</i>       | The pointer to the value to which to set the attribute.                           |
| in | <i>size</i>      | The size of the data pointed to by <i>ptr</i> .                                   |

**Returns**

A `vx_status_e` enumeration.

**`vx_status VX_API_CALL vxQueryThreshold ( vx_threshold thresh, vx_enum attribute, void * ptr, vx_size size )`**

Queries an attribute on the threshold object.

**Parameters**

|     |                  |                                                                                  |
|-----|------------------|----------------------------------------------------------------------------------|
| in  | <i>thresh</i>    | The threshold object to set.                                                     |
| in  | <i>attribute</i> | The attribute to query. Use a <code>vx_threshold_attribute_e</code> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                              |
| in  | <i>size</i>      | The size of the container to which <i>ptr</i> points.                            |

**Returns**

A `vx_status_e` enumeration.

## 3.58 Administrative Features

### 3.58.1 Detailed Description

Defines the Administrative Features of OpenVX.

These features are administrative in nature and require more understanding and are more complex to use.

#### Modules

- [Advanced Objects](#)
- [Advanced Framework API](#)

*Describes components that are considered to be advanced.*

## 3.59 Advanced Objects

### 3.59.1 Detailed Description

#### Modules

- [Object: Array \(Advanced\)](#)  
*Defines the advanced features of the Array Interface.*
- [Object: Node \(Advanced\)](#)  
*Defines the advanced features of the Node Interface.*
- [Object: Delay](#)  
*Defines the Delay Object interface.*
- [Object: Kernel](#)  
*Defines the Kernel Object and Interface.*
- [Object: Parameter](#)  
*Defines the Parameter Object interface.*

## 3.60 Object: Array (Advanced)

### 3.60.1 Detailed Description

Defines the advanced features of the Array Interface.

#### Functions

- `vx_enum VX_API_CALL vxRegisterUserStruct (vx_context context, vx_size size)`  
*Registers user-defined structures to the context.*

### 3.60.2 Function Documentation

**`vx_enum VX_API_CALL vxRegisterUserStruct ( vx_context context, vx_size size )`**

Registers user-defined structures to the context.

#### Parameters

|    |                |                                              |
|----|----------------|----------------------------------------------|
| in | <i>context</i> | The reference to the implementation context. |
| in | <i>size</i>    | The size of user struct in bytes.            |

#### Returns

A `vx_enum` value that is a type given to the User to refer to their custom structure when declaring a `vx_array` of that structure.

#### Return values

|                              |                                               |
|------------------------------|-----------------------------------------------|
| <code>VX_TYPE_INVALID</code> | If the namespace of types has been exhausted. |
|------------------------------|-----------------------------------------------|

#### Note

This call should only be used once within the lifetime of a context for a specific structure.

```
typedef struct _mystruct {
 vx_uint32 some_uint;
 vx_float64 some_double;
} mystruct;
#define MY_NUM_ITEMS (10)
vx_enum mytype = vxRegisterUserStruct(context, sizeof(mystruct));
vx_array array = vxCreateArray(context, mytype, MY_NUM_ITEMS);
```

## 3.61 Object: Node (Advanced)

### 3.61.1 Detailed Description

Defines the advanced features of the Node Interface.

#### Modules

- [Node: Border Modes](#)

*Defines the border mode behaviors.*

#### Functions

- [vx\\_node](#) `VX_API_CALL vxCreateGenericNode (vx_graph graph, vx_kernel kernel)`

*Creates a reference to a node object for a given kernel.*

### 3.61.2 Function Documentation

**`vx_node VX_API_CALL vxCreateGenericNode ( vx_graph graph, vx_kernel kernel )`**

Creates a reference to a node object for a given kernel.

This node has no references assigned as parameters after completion. The client is then required to set these parameters manually by [vxSetParameterByIndex](#). When clients supply their own node creation functions (for use with User Kernels), this is the API to use along with the parameter setting API.

**Parameters**

|                 |               |                                                       |
|-----------------|---------------|-------------------------------------------------------|
| <code>in</code> | <i>graph</i>  | The reference to the graph in which this node exists. |
| <code>in</code> | <i>kernel</i> | The kernel reference to associate with this new node. |

**Returns**

A node reference [vx\\_node](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**Postcondition**

Call [vxSetParameterByIndex](#) for as many parameters as needed to be set.

## 3.62 Node: Border Modes

### 3.62.1 Detailed Description

Defines the border mode behaviors.

Border Mode behavior is set as an attribute of the node, not as a direct parameter to the kernel. This allows clients to *set-and-forget* the modes of any particular node that supports border modes. All nodes shall support [VX\\_BORDER\\_MODE\\_UNDEFINED](#).

### Data Structures

- struct [vx\\_border\\_mode\\_t](#)

Use with the enumeration [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#) to set the border mode behavior of a node that supports borders. [More...](#)

### Enumerations

- enum [vx\\_border\\_mode\\_e](#) {  
[VX\\_BORDER\\_MODE\\_UNDEFINED](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_BORDER\_MODE << 12)) + 0x0,  
[VX\\_BORDER\\_MODE\\_CONSTANT](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_BORDER\_MODE << 12)) + 0x1,  
[VX\\_BORDER\\_MODE\\_REPLICATE](#) = ((( VX\_ID\_KHRONOS ) << 20) | ( VX\_ENUM\_BORDER\_MODE << 12)) + 0x2 }

The border mode list.

### 3.62.2 Data Structure Documentation

#### struct [vx\\_border\\_mode\\_t](#)

Use with the enumeration [VX\\_NODE\\_ATTRIBUTE\\_BORDER\\_MODE](#) to set the border mode behavior of a node that supports borders.

Definition at line [1376](#) of file [vx\\_types.h](#).

#### Data Fields

|                           |                |                                                                                                                                                                                |
|---------------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">vx_enum</a>   | mode           | See <a href="#">vx_border_mode_e</a> .                                                                                                                                         |
| <a href="#">vx_uint32</a> | constant_value | For the mode <a href="#">VX_BORDER_MODE_CONSTANT</a> , this value is filled into each pixel. If there are sub-channels in the pixel then this value is divided up accordingly. |

### 3.62.3 Enumeration Type Documentation

#### enum [vx\\_border\\_mode\\_e](#)

The border mode list.

#### Enumerator

**[VX\\_BORDER\\_MODE\\_UNDEFINED](#)** No defined border mode behavior is given.

**[VX\\_BORDER\\_MODE\\_CONSTANT](#)** For nodes that support this behavior, a constant value is *filled-in* when accessing out-of-bounds pixels.

**[VX\\_BORDER\\_MODE\\_REPLICATE](#)** For nodes that support this behavior, a replication of the nearest edge pixels value is given for out-of-bounds pixels.

Definition at line [1104](#) of file [vx\\_types.h](#).



## 3.63 Object: Delay

### 3.63.1 Detailed Description

Defines the Delay Object interface.

A Delay is an opaque object that contains a manually-controlled, temporally-delayed list of objects.

#### Typedefs

- typedef struct \_vx\_delay \* [vx\\_delay](#)

*The delay object. This is like a ring buffer of objects that is maintained by the OpenVX implementation.*

#### Enumerations

- enum [vx\\_delay\\_attribute\\_e](#) {  
[VX\\_DELAY\\_ATTRIBUTE\\_TYPE](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_DELAY](#) << 8)) + 0x0,  
[VX\\_DELAY\\_ATTRIBUTE\\_SLOTS](#) = ((( [VX\\_ID\\_KHRONOS](#) ) << 20) | ( [VX\\_TYPE\\_DELAY](#) << 8)) + 0x1 }

*The delay attribute list.*

#### Functions

- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxAgeDelay](#) ([vx\\_delay](#) delay)  
*Ages the internal delay ring by one. This means that once this API is called the reference from index 0 will go to index -1 and so forth until -count + 1 is reached. This last object will become 0. Once the delay has been aged, it updates the reference in any associated nodes.*
- [vx\\_delay](#) [VX\\_API\\_CALL](#) [vxCreateDelay](#) ([vx\\_context](#) context, [vx\\_reference](#) exemplar, [vx\\_size](#) slots)  
*Creates a Delay object.*
- [vx\\_reference](#) [VX\\_API\\_CALL](#) [vxGetReferenceFromDelay](#) ([vx\\_delay](#) delay, [vx\\_int32](#) index)  
*Retrieves a reference from a delay object.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxQueryDelay](#) ([vx\\_delay](#) delay, [vx\\_enum](#) attribute, void \*ptr, [vx\\_size](#) size)  
*Queries a [vx\\_delay](#) object attribute.*
- [vx\\_status](#) [VX\\_API\\_CALL](#) [vxReleaseDelay](#) ([vx\\_delay](#) \*delay)  
*Releases a reference to a delay object. The object may not be garbage collected until its total reference count is zero.*

### 3.63.2 Typedef Documentation

**typedef struct \_vx\_delay\* vx\_delay**

The delay object. This is like a ring buffer of objects that is maintained by the OpenVX implementation.

See also

[vxCreateDelay](#)

Definition at line 210 of file [vx\\_types.h](#).

### 3.63.3 Enumeration Type Documentation

**enum vx\_delay\_attribute\_e**

The delay attribute list.

Enumerator

**VX\_DELAY\_ATTRIBUTE\_TYPE** The type of reference contained in the delay. Use a [vx\\_enum](#) parameter.

**VX\_DELAY\_ATTRIBUTE\_SLOTS** The number of items in the delay. Use a [vx\\_size](#) parameter.

Definition at line 1146 of file [vx\\_types.h](#).

### 3.63.4 Function Documentation

**vx\_status VX\_API\_CALL vxQueryDelay ( vx\_delay *delay*, vx\_enum *attribute*, void \* *ptr*, vx\_size *size* )**

Queries a [vx\\_delay](#) object attribute.

**Parameters**

|     |                  |                                                                                 |
|-----|------------------|---------------------------------------------------------------------------------|
| in  | <i>delay</i>     | A pointer to a delay object.                                                    |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_delay_attribute_e</a> enumeration. |
| out | <i>ptr</i>       | The location at which to store the resulting value.                             |
| in  | <i>size</i>      | The size of the container to which <i>ptr</i> points.                           |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**vx\_status VX\_API\_CALL vxReleaseDelay ( vx\_delay \* *delay* )**

Releases a reference to a delay object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |              |                                      |
|----|--------------|--------------------------------------|
| in | <i>delay</i> | The pointer to the delay to release. |
|----|--------------|--------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                   |                                              |
|-----------------------------------|----------------------------------------------|
| <i>VX_SUCCESS</i>                 | No errors.                                   |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If delay is not a <a href="#">vx_delay</a> . |

**vx\_delay VX\_API\_CALL vxCreateDelay ( vx\_context *context*, vx\_reference *exemplar*, vx\_size *slots* )**

Creates a Delay object.

This function uses a subset of the attributes defining the metadata of the exemplar, ignoring the object. It does not alter the exemplar or keep or release the reference to the exemplar. For the definition of supported attributes see [vxSetMetaFormatAttribute](#).

**Parameters**

|    |                 |                                       |
|----|-----------------|---------------------------------------|
| in | <i>context</i>  | The reference to the system context.  |
| in | <i>exemplar</i> | The exemplar object.                  |
| in | <i>slots</i>    | The number of reference in the delay. |

**Returns**

A delay reference [vx\\_delay](#). Any possible errors preventing a successful creation should be checked using [vxGetStatus](#).

**vx\_reference VX\_API\_CALL vxGetReferenceFromDelay ( vx\_delay *delay*, vx\_int32 *index* )**

Retrieves a reference from a delay object.

**Parameters**

|    |              |                                                              |
|----|--------------|--------------------------------------------------------------|
| in | <i>delay</i> | The reference to the delay object.                           |
| in | <i>index</i> | An index into the delay from which to extract the reference. |

## Returns

[vx\\_reference](#)

## Note

The delay index is in the range  $[-count + 1, 0]$ . 0 is always the *current* object.

A reference from a delay object must not be given to its associated release API (e.g. [vxReleaseImage](#)).

Use the [vxReleaseDelay](#) only.

**vx\_status VX\_API\_CALL vxAgeDelay ( vx\_delay delay )**

Ages the internal delay ring by one. This means that once this API is called the reference from index 0 will go to index -1 and so forth until  $-count + 1$  is reached. This last object will become 0. Once the delay has been aged, it updates the reference in any associated nodes.

## Parameters

|    |              |  |
|----|--------------|--|
| in | <i>delay</i> |  |
|----|--------------|--|

## Returns

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                   |                                                                |
|-----------------------------------|----------------------------------------------------------------|
| <i>VX_SUCCESS</i>                 | Delay was aged.                                                |
| <i>VX_ERROR_INVALID_REFERENCE</i> | The value passed as delay was not a <a href="#">vx_delay</a> . |

## 3.64 Object: Kernel

### 3.64.1 Detailed Description

Defines the Kernel Object and Interface.

A Kernel in OpenVX is the abstract representation of an computer vision function, such as a “Sobel Gradient” or “Lucas Kanade Feature Tracking”. A vision function may implement many similar or identical features from other functions, but it is still considered a single unique kernel as long as it is named by the same string and enumeration and conforms to the results specified by OpenVX. Kernels are similar to function signatures in this regard.

In each of the cases, a client of OpenVX could request the kernels in nearly the same manner. There are two main approaches, which depend on the method a client calls to get the kernel reference. The first uses enumerations.

```
vx_kernel kernel = vxGetKernelByEnum(context,
VX_KERNEL_SOBEL_3x3);
vx_node node = vxCreateGenericNode(graph, kernel);
```

The second method depends on using strings to get the kernel reference.

```
vx_kernel kernel = vxGetKernelByName(context, "
org.khronos.openvx.sobel3x3");
vx_node node = vxCreateGenericNode(graph, kernel);
```

## Data Structures

- struct `vx_kernel_info_t`

*The Kernel Information Structure. This is returned by the Context to indicate which kernels are available in the OpenVX implementation. [More...](#)*

## Macros

- `#define VX_MAX_KERNEL_NAME` (256)

*Defines the maximum string length of a kernel name to be added to OpenVX.*

## Typedefs

- typedef struct `_vx_kernel` \* `vx_kernel`

*An opaque reference to the descriptor of a kernel.*

## Enumerations

- enum `vx_kernel_attribute_e` {  
`VX_KERNEL_ATTRIBUTE_PARAMETERS` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_KERNEL` << 8)) + 0x0,  
`VX_KERNEL_ATTRIBUTE_NAME` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_KERNEL` << 8)) + 0x1,  
`VX_KERNEL_ATTRIBUTE_ENUM` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_KERNEL` << 8)) + 0x2,  
`VX_KERNEL_ATTRIBUTE_LOCAL_DATA_SIZE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_KERNEL` << 8)) + 0x3,  
`VX_KERNEL_ATTRIBUTE_LOCAL_DATA_PTR` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_KERNEL` << 8)) + 0x4 }

*The kernel attributes list.*

- enum `vx_kernel_e` {  
`VX_KERNEL_INVALID` = `VX_KERNEL_BASE`(`VX_ID_KHRONOS`, `VX_LIBRARY_KHR_BASE`) + 0x0,  
`VX_KERNEL_COLOR_CONVERT` = `VX_KERNEL_BASE`(`VX_ID_KHRONOS`, `VX_LIBRARY_KHR_BASE`) + 0x1,  
`VX_KERNEL_CHANNEL_EXTRACT` = `VX_KERNEL_BASE`(`VX_ID_KHRONOS`, `VX_LIBRARY_KHR_BASE`) + 0x2,  
`VX_KERNEL_CHANNEL_COMBINE` = `VX_KERNEL_BASE`(`VX_ID_KHRONOS`, `VX_LIBRARY_KHR_BASE`) + 0x3

```

SE) + 0x3,
VX_KERNEL_SOBEL_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x4,
VX_KERNEL_MAGNITUDE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x5,
VX_KERNEL_PHASE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x6,
VX_KERNEL_SCALE_IMAGE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x7,
VX_KERNEL_TABLE_LOOKUP = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x8,
VX_KERNEL_HISTOGRAM = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x9,
VX_KERNEL_EQUALIZE_HISTOGRAM = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_
_BASE) + 0xA,
VX_KERNEL_ABSDIFF = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xB,
VX_KERNEL_MEAN_STDDEV = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0xC,
VX_KERNEL_THRESHOLD = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xD,
VX_KERNEL_INTEGRAL_IMAGE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE)
+ 0xE,
VX_KERNEL_DILATE_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0xF,
VX_KERNEL_ERODE_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x10,
VX_KERNEL_MEDIAN_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x11,
VX_KERNEL_BOX_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x12,
VX_KERNEL_GAUSSIAN_3x3 = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x13,
VX_KERNEL_CUSTOM_CONVOLUTION = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KH_
R_BASE) + 0x14,
VX_KERNEL_GAUSSIAN_PYRAMID = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_B_
ASE) + 0x15,
VX_KERNEL_ACCUMULATE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x16,
VX_KERNEL_ACCUMULATE_WEIGHTED = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_K_
HR_BASE) + 0x17,
VX_KERNEL_ACCUMULATE_SQUARE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_
_BASE) + 0x18,
VX_KERNEL_MINMAXLOC = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x19,
VX_KERNEL_CONVERTDEPTH = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x1A,
VX_KERNEL_CANNY_EDGE_DETECTOR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_K_
HR_BASE) + 0x1B,
VX_KERNEL_AND = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1C,
VX_KERNEL_OR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1D,
VX_KERNEL_XOR = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1E,
VX_KERNEL_NOT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x1F,
VX_KERNEL_MULTIPLY = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x20,
VX_KERNEL_ADD = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x21,
VX_KERNEL_SUBTRACT = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x22,
VX_KERNEL_WARP_AFFINE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x23,
VX_KERNEL_WARP_PERSPECTIVE = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_B_
ASE) + 0x24,
VX_KERNEL_HARRIS_CORNERS = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE)
+ 0x25,
VX_KERNEL_FAST_CORNERS = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) +
0x26,
VX_KERNEL_OPTICAL_FLOW_PYR_LK = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KH_
R_BASE) + 0x27,
VX_KERNEL_REMAP = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_BASE) + 0x28,
VX_KERNEL_HALFSCALE_GAUSSIAN = VX_KERNEL_BASE(VX_ID_KHRONOS, VX_LIBRARY_KHR_
_BASE) + 0x29,

```

**VX\_KERNEL\_MAX\_1\_0 }**

*The standard list of available vision kernels.*

## Functions

- **vx\_kernel** VX\_API\_CALL **vxGetKernelByEnum** (**vx\_context** context, **vx\_enum** kernel)  
*Obtains a reference to the kernel using the **vx\_kernel\_e** enumeration.*
- **vx\_kernel** VX\_API\_CALL **vxGetKernelByName** (**vx\_context** context, const **vx\_char** \*name)  
*Obtains a reference to a kernel using a string to specify the name.*
- **vx\_status** VX\_API\_CALL **vxQueryKernel** (**vx\_kernel** kernel, **vx\_enum** attribute, void \*ptr, **vx\_size** size)  
*This allows the client to query the kernel to get information about the number of parameters, enum values, etc.*
- **vx\_status** VX\_API\_CALL **vxReleaseKernel** (**vx\_kernel** \*kernel)  
*Release the reference to the kernel. The object may not be garbage collected until its total reference count is zero.*

### 3.64.2 Data Structure Documentation

**struct vx\_kernel\_info\_t**

The Kernel Information Structure. This is returned by the Context to indicate which kernels are available in the OpenVX implementation.

Definition at line 1348 of file **vx\_types.h**.

Data Fields

|                |                          |                                                                                                                                    |
|----------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>vx_enum</b> | enumeration              | The kernel enumeration value from <b>vx_kernel_e</b> (or an extension thereof).<br><br>See also<br><br><b>vxGetKernelByEnum</b>    |
| <b>vx_char</b> | name[VX_MAX_KERNEL_NAME] | The kernel name in dotted hierarchical format. e.g. "org.khronos.openvx.sobel3x3".<br><br>See also<br><br><b>vxGetKernelByName</b> |

### 3.64.3 Typedef Documentation

**typedef struct \_vx\_kernel\* vx\_kernel**

An opaque reference to the descriptor of a kernel.

See also

**vxGetKernelByName**  
**vxGetKernelByEnum**

Definition at line 174 of file **vx\_types.h**.

### 3.64.4 Enumeration Type Documentation

**enum vx\_kernel\_e**

The standard list of available vision kernels.

Each kernel listed here can be used with the **vxGetKernelByEnum** call. When programming the parameters, use

- **VX\_INPUT** for [in]
- **VX\_OUTPUT** for [out]

- `VX_BIDIRECTIONAL` for [in,out]

When programming the parameters, use

- `VX_TYPE_IMAGE` for a `vx_image` in the size field of `vxGetParameterByIndex` or `vxSetParameterByIndex` \*
- `VX_TYPE_ARRAY` for a `vx_array` in the size field of `vxGetParameterByIndex` or `vxSetParameterByIndex` \*
- or other appropriate types in `vx_type_e`.

Enumerator

**`VX_KERNEL_INVALID`** The invalid kernel is used to for conformance failure in relation to some kernel operation (Get/Release). If the kernel is executed it shall always return an error. The kernel has no parameters. To address by name use "org.khronos.openvx.invalid".

**`VX_KERNEL_COLOR_CONVERT`** The Color Space conversion kernel. The conversions are based on the `vx_df_image_e` code in the images.

See also

[Color Convert](#)

**`VX_KERNEL_CHANNEL_EXTRACT`** The Generic Channel Extraction Kernel. This kernel can remove individual color channels from an interleaved or semi-planar, planar, sub-sampled planar image. A client could extract a red channel from an interleaved RGB image or do a Luma extract from a YUV format.

See also

[Channel Extract](#)

**`VX_KERNEL_CHANNEL_COMBINE`** The Generic Channel Combine Kernel. This kernel combine multiple individual planes into a single multiplanar image of the type specified in the output image.

See also

[Channel Combine](#)

**`VX_KERNEL_SOBEL_3x3`** The Sobel 3x3 Filter Kernel.

See also

[Sobel 3x3](#)

**`VX_KERNEL_MAGNITUDE`** The Magnitude Kernel. This kernel produces a magnitude plane from two input gradients.

See also

[Magnitude](#)

**`VX_KERNEL_PHASE`** The Phase Kernel. This kernel produces a phase plane from two input gradients.

See also

[Phase](#)

**`VX_KERNEL_SCALE_IMAGE`** The Scale Image Kernel. This kernel provides resizing of an input image to an output image. The scaling factor is determined but the relative sizes of the input and output.

See also

[Scale Image](#)

**`VX_KERNEL_TABLE_LOOKUP`** The Table Lookup kernel.

See also

[TableLookup](#)

**`VX_KERNEL_HISTOGRAM`** The Histogram Kernel.



See also

[Histogram](#)

**VX\_KERNEL\_EQUALIZE\_HISTOGRAM** The Histogram Equalization Kernel.

See also

[Equalize Histogram](#)

**VX\_KERNEL\_ABSDIFF** The Absolute Difference Kernel.

See also

[Absolute Difference](#)

**VX\_KERNEL\_MEAN\_STDDEV** The Mean and Standard Deviation Kernel.

See also

[Mean and Standard Deviation](#)

**VX\_KERNEL\_THRESHOLD** The Threshold Kernel.

See also

[Thresholding](#)

**VX\_KERNEL\_INTEGRAL\_IMAGE** The Integral Image Kernel.

See also

[Integral Image](#)

**VX\_KERNEL\_DILATE\_3x3** The dilate kernel.

See also

[Dilate Image](#)

**VX\_KERNEL\_ERODE\_3x3** The erode kernel.

See also

[Erode Image](#)

**VX\_KERNEL\_MEDIAN\_3x3** The median image filter.

See also

[Median Filter](#)

**VX\_KERNEL\_BOX\_3x3** The box filter kernel.

See also

[Box Filter](#)

**VX\_KERNEL\_GAUSSIAN\_3x3** The gaussian filter kernel.

See also

[Gaussian Filter](#)

**VX\_KERNEL\_CUSTOM\_CONVOLUTION** The custom convolution kernel.

See also

[Custom Convolution](#)

**VX\_KERNEL\_GAUSSIAN\_PYRAMID** The gaussian image pyramid kernel.

See also

[Gaussian Image Pyramid](#)

**VX\_KERNEL\_ACCUMULATE** The accumulation kernel.

See also

[Accumulate](#)

**VX\_KERNEL\_ACCUMULATE\_WEIGHTED** The weighed accumulation kernel.

See also

[Accumulate Weighted](#)

**VX\_KERNEL\_ACCUMULATE\_SQUARE** The squared accumulation kernel.

See also

[Accumulate Squared](#)

**VX\_KERNEL\_MINMAXLOC** The min and max location kernel.

See also

[Min, Max Location](#)

**VX\_KERNEL\_CONVERTDEPTH** The bit-depth conversion kernel.

See also

[Convert Bit depth](#)

**VX\_KERNEL\_CANNY\_EDGE\_DETECTOR** The Canny Edge Detector.

See also

[Canny Edge Detector](#)

**VX\_KERNEL\_AND** The Bitwise And Kernel.

See also

[Bitwise AND](#)

**VX\_KERNEL\_OR** The Bitwise Inclusive Or Kernel.

See also

[Bitwise INCLUSIVE OR](#)

**VX\_KERNEL\_XOR** The Bitwise Exclusive Or Kernel.

See also

[Bitwise EXCLUSIVE OR](#)

**VX\_KERNEL\_NOT** The Bitwise Not Kernel.

See also

[Bitwise NOT](#)

**VX\_KERNEL\_MULTIPLY** The Pixelwise Multiplication Kernel.

See also

[Pixel-wise Multiplication](#)

**VX\_KERNEL\_ADD** The Addition Kernel.

See also

[Arithmetic Addition](#)

**VX\_KERNEL\_SUBTRACT** The Subtraction Kernel.

See also

[Arithmetic Subtraction](#)

**VX\_KERNEL\_WARP\_AFFINE** The Warp Affine Kernel.

See also

[Warp Affine](#)

**VX\_KERNEL\_WARP\_PERSPECTIVE** The Warp Perspective Kernel.

See also

[Warp Perspective](#)

**VX\_KERNEL\_HARRIS\_CORNERS** The Harris Corners Kernel.

See also

[Harris Corners](#)

**VX\_KERNEL\_FAST\_CORNERS** The FAST Corners Kernel.

See also

[Fast Corners](#)

**VX\_KERNEL\_OPTICAL\_FLOW\_PYR\_LK** The Optical Flow Pyramid (LK) Kernel.

See also

[Optical Flow Pyramid \(LK\)](#)

**VX\_KERNEL\_REMAP** The Remap Kernel.

See also

[Remap](#)

**VX\_KERNEL\_HALFSCALE\_GAUSSIAN** The Half Scale Gaussian Kernel.

See also

[Scale Image](#)

Definition at line 55 of file [vx\\_kernels.h](#).

#### **enum vx\_kernel\_attribute\_e**

The kernel attributes list.

Enumerator

**VX\_KERNEL\_ATTRIBUTE\_PARAMETERS** Queries a kernel for the number of parameters the kernel supports. Use a [vx\\_uint32](#) parameter.

**VX\_KERNEL\_ATTRIBUTE\_NAME** Queries the name of the kernel. Not settable. Use a [vx\\_char\[VX\\_MAX\\_KERNEL\\_NAME\]](#) array (not a [vx\\_array](#)).

**VX\_KERNEL\_ATTRIBUTE\_ENUM** Queries the enum of the kernel. Not settable. Use a [vx\\_enum](#) parameter.

**VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_SIZE** The local data area allocated with each kernel when it becomes a node. Use a [vx\\_size](#) parameter.

Note

If not set it will default to zero.

**VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_PTR** The local data pointer allocated with each kernel when it becomes a node. Use a void pointer parameter. Use a [vx\\_size](#) parameter.

Definition at line 734 of file [vx\\_types.h](#).

### **3.64.5 Function Documentation**

**vx\_kernel VX\_API\_CALL vxGetKernelByName ( vx\_context context, const vx\_char \* name )**

Obtains a reference to a kernel using a string to specify the name.

User Kernels follow a "dotted" heirarchical syntax. For example: "com.company.example.xyz". The following are strings specifying the kernel names:

```
org.khronos.openvx.color_convert
org.khronos.openvx.channel_extract
org.khronos.openvx.channel_combine
org.khronos.openvx.sobel_3x3
org.khronos.openvx.magnitude
org.khronos.openvx.phase
org.khronos.openvx.scale_image
org.khronos.openvx.table_lookup
```

```

org.khronos.openvx.histogram
org.khronos.openvx.equalize_histogram
org.khronos.openvx.absdiff
org.khronos.openvx.mean_stddev
org.khronos.openvx.threshold
org.khronos.openvx.integral_image
org.khronos.openvx.dilate_3x3
org.khronos.openvx.erode_3x3
org.khronos.openvx.median_3x3
org.khronos.openvx.box_3x3
org.khronos.openvx.gaussian_3x3
org.khronos.openvx.custom_convolution
org.khronos.openvx.gaussian_pyramid
org.khronos.openvx.accumulate
org.khronos.openvx.accumulate_weighted
org.khronos.openvx.accumulate_square
org.khronos.openvx.minmaxloc
org.khronos.openvx.convertdepth
org.khronos.openvx.canny_edge_detector
org.khronos.openvx.and
org.khronos.openvx.or
org.khronos.openvx.xor
org.khronos.openvx.not
org.khronos.openvx.multiply
org.khronos.openvx.add
org.khronos.openvx.subtract
org.khronos.openvx.warp_affine
org.khronos.openvx.warp_perspective
org.khronos.openvx.harris_corners
org.khronos.openvx.fast_corners
org.khronos.openvx.optical_flow_pyr_lk
org.khronos.openvx.remap
org.khronos.openvx.halfscale_gaussian

```

**Parameters**

|           |                |                                              |
|-----------|----------------|----------------------------------------------|
| <i>in</i> | <i>context</i> | The reference to the implementation context. |
| <i>in</i> | <i>name</i>    | The string of the name of the kernel to get. |

**Returns**

A kernel reference or zero if an error occurred.

**Return values**

|          |                                              |
|----------|----------------------------------------------|
| <i>0</i> | The kernel name is not found in the context. |
|----------|----------------------------------------------|

**Precondition**

[vxLoadKernels](#) if the kernel is not provided by the OpenVX implementation.

**Note**

User Kernels should follow a "dotted" heirarchical syntax. For example: "com.company.example.xyz".

**`vx_kernel VX_API_CALL vxGetKernelByEnum ( vx_context context, vx_enum kernel )`**

Obtains a reference to the kernel using the [vx\\_kernel\\_e](#) enumeration.

Enum values above the standard set are assumed to apply to loaded libraries.

**Parameters**

|    |                |                                                                               |
|----|----------------|-------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the implementation context.                                  |
| in | <i>kernel</i>  | A value from <a href="#">vx_kernel_e</a> or a vendor or client-defined value. |

**Returns**

A [vx\\_kernel](#).

**Return values**

|   |                                                     |
|---|-----------------------------------------------------|
| 0 | The kernel enumeration is not found in the context. |
|---|-----------------------------------------------------|

**Precondition**

[vxLoadKernels](#) if the kernel is not provided by the OpenVX implementation.

**vx\_status VX\_API\_CALL vxQueryKernel ( vx\_kernel *kernel*, vx\_enum *attribute*, void \* *ptr*, vx\_size *size* )**

This allows the client to query the kernel to get information about the number of parameters, enum values, etc.

**Parameters**

|     |                  |                                                                       |
|-----|------------------|-----------------------------------------------------------------------|
| in  | <i>kernel</i>    | The kernel reference to query.                                        |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_kernel_attribute_e</a> . |
| out | <i>ptr</i>       | The pointer to the location at which to store the resulting value.    |
| in  | <i>size</i>      | The size of the container to which <i>ptr</i> points.                 |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                    |                                                                 |
|------------------------------------|-----------------------------------------------------------------|
| <i>VX_SUCCESS</i>                  | No errors.                                                      |
| <i>VX_ERROR_INVALID_REFERENCE</i>  | If the kernel is not a <a href="#">vx_kernel</a> .              |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | If any of the other parameters are incorrect.                   |
| <i>VX_ERROR_NOT_SUPPORTED</i>      | If the attribute value is not supported in this implementation. |

**vx\_status VX\_API\_CALL vxReleaseKernel ( vx\_kernel \* *kernel* )**

Release the reference to the kernel. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |               |                                                 |
|----|---------------|-------------------------------------------------|
| in | <i>kernel</i> | The pointer to the kernel reference to release. |
|----|---------------|-------------------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                         |                                             |
|-----------------------------------------|---------------------------------------------|
| <code>VX_SUCCESS</code>                 | No errors.                                  |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If kernel is not a <code>vx_kernel</code> . |

## 3.65 Object: Parameter

### 3.65.1 Detailed Description

Defines the Parameter Object interface.

An abstract input, output, or bidirectional data object passed to a computer vision function. This object contains the signature of that parameter's usage from the kernel description. This information includes:

- *Signature Index* - The numbered index of the parameter in the signature.
- *Object Type* - e.g., `VX_TYPE_IMAGE` or `VX_TYPE_ARRAY` or some other object type from `vx_type_e`.
- *Usage Model* - e.g., `VX_INPUT`, `VX_OUTPUT`, or `VX_BIDIRECTIONAL`.
- *Presence State* - e.g., `VX_PARAMETER_STATE_REQUIRED` or `VX_PARAMETER_STATE_OPTIONAL`.

### Typedefs

- typedef struct \_vx\_parameter \* `vx_parameter`

*An opaque reference to a single parameter.*

### Enumerations

- enum `vx_direction_e` {  
`VX_INPUT` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_DIRECTION` << 12)) + 0x0,  
`VX_OUTPUT` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_DIRECTION` << 12)) + 0x1,  
`VX_BIDIRECTIONAL` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_DIRECTION` << 12)) + 0x2 }

*An indication of how a kernel will treat the given parameter.*

- enum `vx_parameter_attribute_e` {  
`VX_PARAMETER_ATTRIBUTE_INDEX` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_PARAMETER` << 8)) + 0x0,  
`VX_PARAMETER_ATTRIBUTE_DIRECTION` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_PARAMETER` << 8)) + 0x1,  
`VX_PARAMETER_ATTRIBUTE_TYPE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_PARAMETER` << 8)) + 0x2,  
`VX_PARAMETER_ATTRIBUTE_STATE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_PARAMETER` << 8)) + 0x3,  
`VX_PARAMETER_ATTRIBUTE_REF` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_TYPE_PARAMETER` << 8)) + 0x4 }

*The parameter attributes list.*

- enum `vx_parameter_state_e` {  
`VX_PARAMETER_STATE_REQUIRED` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_PARAMETER_STATE` << 12)) + 0x0,  
`VX_PARAMETER_STATE_OPTIONAL` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_PARAMETER_STATE` << 12)) + 0x1 }

*The parameter state type.*

### Functions

- `vx_parameter` `VX_API_CALL vxGetKernelParameterByIndex` (`vx_kernel` kernel, `vx_uint32` index)  
*Retrieves a `vx_parameter` from a `vx_kernel`.*
- `vx_parameter` `VX_API_CALL vxGetParameterByIndex` (`vx_node` node, `vx_uint32` index)  
*Retrieves a `vx_parameter` from a `vx_node`.*
- `vx_status` `VX_API_CALL vxQueryParameter` (`vx_parameter` param, `vx_enum` attribute, void \*ptr, `vx_size` size)  
*Allows the client to query a parameter to determine its meta-information.*
- `vx_status` `VX_API_CALL vxReleaseParameter` (`vx_parameter` \*param)

*Releases a reference to a parameter object. The object may not be garbage collected until its total reference count is zero.*

- **vx\_status** VX\_API\_CALL **vxSetParameterByIndex** (**vx\_node** node, **vx\_uint32** index, **vx\_reference** value)  
*Sets the specified parameter data for a kernel on the node.*
- **vx\_status** VX\_API\_CALL **vxSetParameterByReference** (**vx\_parameter** parameter, **vx\_reference** value)  
*Associates a parameter reference and a data reference with a kernel on a node.*

### 3.65.2 Typedef Documentation

**typedef struct \_vx\_parameter\* vx\_parameter**

An opaque reference to a single parameter.

See also

[vxGetParameterByIndex](#)

Definition at line 181 of file [vx\\_types.h](#).

### 3.65.3 Enumeration Type Documentation

**enum vx\_direction\_e**

An indication of how a kernel will treat the given parameter.

Enumerator

**VX\_INPUT** The parameter is an input only.

**VX\_OUTPUT** The parameter is an output only.

**VX\_BIDIRECTIONAL** The parameter is both an input and output.

Definition at line 558 of file [vx\\_types.h](#).

**enum vx\_parameter\_attribute\_e**

The parameter attributes list.

Enumerator

**VX\_PARAMETER\_ATTRIBUTE\_INDEX** Queries a parameter for its index value on the kernel with which it is associated. Use a [vx\\_uint32](#) parameter.

**VX\_PARAMETER\_ATTRIBUTE\_DIRECTION** Queries a parameter for its direction value on the kernel with which it is associated. Use a [vx\\_enum](#) parameter.

**VX\_PARAMETER\_ATTRIBUTE\_TYPE** Queries a parameter for its type, [vx\\_type\\_e](#) is returned. The size of the parameter is implied for plain data objects. For opaque data objects like images and arrays a query to their attributes has to be called to determine the size.

**VX\_PARAMETER\_ATTRIBUTE\_STATE** Queries a parameter for its state. A value in [vx\\_parameter\\_state\\_e](#) is returned. Use a [vx\\_enum](#) parameter.

**VX\_PARAMETER\_ATTRIBUTE\_REF** Use to extract the reference contained in the parameter. Use a [vx\\_reference](#) parameter.

Definition at line 784 of file [vx\\_types.h](#).

**enum vx\_parameter\_state\_e**

The parameter state type.

Enumerator

**VX\_PARAMETER\_STATE\_REQUIRED** Default. The parameter must be supplied. If not set, during Verify, an error is returned.

**VX\_PARAMETER\_STATE\_OPTIONAL** The parameter may be unspecified. The kernel takes care not to deference optional parameters until it is certain they are valid.

Definition at line 1090 of file [vx\\_types.h](#).



### 3.65.4 Function Documentation

**vx\_parameter VX\_API\_CALL vxGetKernelParameterByIndex ( vx\_kernel *kernel*, vx\_uint32 *index* )**

Retrieves a [vx\\_parameter](#) from a [vx\\_kernel](#).

**Parameters**

|    |               |                              |
|----|---------------|------------------------------|
| in | <i>kernel</i> | The reference to the kernel. |
| in | <i>index</i>  | The index of the parameter.  |

**Returns**

A [vx\\_parameter](#).

**Return values**

|   |                                        |
|---|----------------------------------------|
| 0 | Either the kernel or index is invalid. |
| * | The parameter reference.               |

**vx\_parameter VX\_API\_CALL vxGetParameterByIndex ( vx\_node *node*, vx\_uint32 *index* )**

Retrieves a [vx\\_parameter](#) from a [vx\\_node](#).

**Parameters**

|    |              |                                                         |
|----|--------------|---------------------------------------------------------|
| in | <i>node</i>  | The node from which to extract the parameter.           |
| in | <i>index</i> | The index of the parameter to which to get a reference. |

**Returns**

[vx\\_parameter](#)

**vx\_status VX\_API\_CALL vxReleaseParameter ( vx\_parameter \* *param* )**

Releases a reference to a parameter object. The object may not be garbage collected until its total reference count is zero.

**Parameters**

|    |              |                                          |
|----|--------------|------------------------------------------|
| in | <i>param</i> | The pointer to the parameter to release. |
|----|--------------|------------------------------------------|

**Postcondition**

After returning from this function the reference is zeroed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                            |                                                  |
|----------------------------|--------------------------------------------------|
| VX_SUCCESS                 | No errors.                                       |
| VX_ERROR_INVALID_REFERENCE | If param is not a <a href="#">vx_parameter</a> . |

**vx\_status VX\_API\_CALL vxSetParameterByIndex ( vx\_node *node*, vx\_uint32 *index*, vx\_reference *value* )**

Sets the specified parameter data for a kernel on the node.

**Parameters**

|    |              |                                     |
|----|--------------|-------------------------------------|
| in | <i>node</i>  | The node that contains the kernel.  |
| in | <i>index</i> | The index of the parameter desired. |
| in | <i>value</i> | The reference to the parameter.     |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**See also**

[vxSetParameterByReference](#)

**vx\_status VX\_API\_CALL vxSetParameterByReference ( vx\_parameter *parameter*, vx\_reference *value* )**

Associates a parameter reference and a data reference with a kernel on a node.

**Parameters**

|    |                  |                                                   |
|----|------------------|---------------------------------------------------|
| in | <i>parameter</i> | The reference to the kernel parameter.            |
| in | <i>value</i>     | The value to associate with the kernel parameter. |

**Returns**

A [vx\\_status\\_e](#) enumeration.

**See also**

[vxGetParameterByIndex](#)

**vx\_status VX\_API\_CALL vxQueryParameter ( vx\_parameter *param*, vx\_enum *attribute*, void \* *ptr*, vx\_size *size* )**

Allows the client to query a parameter to determine its meta-information.

**Parameters**

|     |                  |                                                                          |
|-----|------------------|--------------------------------------------------------------------------|
| in  | <i>param</i>     | The reference to the parameter.                                          |
| in  | <i>attribute</i> | The attribute to query. Use a <a href="#">vx_parameter_attribute_e</a> . |
| out | <i>ptr</i>       | The location at which to store the resulting value.                      |
| in  | <i>size</i>      | The size in bytes of the container to which <i>ptr</i> points.           |

**Returns**

A [vx\\_status\\_e](#) enumeration.

## 3.66 Advanced Framework API

### 3.66.1 Detailed Description

Describes components that are considered to be advanced.

Advanced topics include: extensions through User Kernels; Reflection and Introspection; Performance Tweaking through Hinting and Directives; and Debugging Callbacks.

#### Modules

- [Framework: Node Callbacks](#)  
*Allows Clients to receive a callback after a specific node has completed execution.*
- [Framework: Performance Measurement](#)  
*Defines Performance measurement and reporting interfaces.*
- [Framework: Log](#)  
*Defines the debug logging interface.*
- [Framework: Hints](#)  
*Defines the Hints Interface.*
- [Framework: Directives](#)  
*Defines the Directives Interface.*
- [Framework: User Kernels](#)  
*Defines the User Kernels, which are a method to extend OpenVX with new vision functions.*
- [Framework: Graph Parameters](#)  
*Defines the Graph Parameter API.*

## 3.67 Framework: Node Callbacks

### 3.67.1 Detailed Description

Allows Clients to receive a callback after a specific node has completed execution.

Callbacks are not guaranteed to be called *immediately* after the Node completes. Callbacks are intended to be used to create simple *early exit* conditions for Vision graphs using `vx_action_e` return values. An example of setting up a callback can be seen below:

```
vx_graph graph = vxCreateGraph(context);
status = vxGetStatus((vx_reference)graph);
if (status == VX_SUCCESS) {
 vx_uint8 lmin = 0, lmax = 0;
 vx_uint32 minCount = 0, maxCount = 0;
 vx_scalar scalars[] = {
 vxCreateScalar(context, VX_TYPE_UINT8, &lmin),
 vxCreateScalar(context, VX_TYPE_UINT8, &lmax),
 vxCreateScalar(context, VX_TYPE_UINT32, &minCount),
 vxCreateScalar(context, VX_TYPE_UINT32, &maxCount),
 };
 vx_array arrays[] = {
 vxCreateArray(context, VX_TYPE_COORDINATES2D, 1),
 vxCreateArray(context, VX_TYPE_COORDINATES2D, 1)
 };
 vx_node nodes[] = {
 vxMinMaxLocNode(graph, input, scalars[0], scalars[1], arrays[0], arrays[1],
 scalars[2], scalars[3]),
 };
 status = vxAssignNodeCallback(nodes[0], &analyze_brightness);
 // do other
}
```

Once the graph has been initialized and the callback has been installed then the callback itself will be called during graph execution.

```
#define MY_DESIRED_THRESHOLD (10)
vx_action VX_CALLBACK analyze_brightness(vx_node node) {
 // extract the max value
 vx_action action = VX_ACTION_ABANDON;
 vx_parameter pmax = vxGetParameterByIndex(node, 2); // Max Value
 if (pmax) {
 vx_scalar smax = 0;
 vxQueryParameter(pmax, VX_PARAMETER_ATTRIBUTE_REF, &smax,
 sizeof(smax));
 if (smax) {
 vx_uint8 value = 0u;
 vxReadScalarValue(smax, &value);
 if (value >= MY_DESIRED_THRESHOLD) {
 action = VX_ACTION_CONTINUE;
 }
 vxReleaseScalar(&smax);
 }
 vxReleaseParameter(&pmax);
 }
 return action;
}
```

#### Warning

This should be used with **extreme** caution as it can *ruin* optimizations in the power/performance efficiency of a graph.

The callback must return a `vx_action` code indicating how the graph processing should proceed.

- If `VX_ACTION_CONTINUE` is returned, the graph will continue execution with no changes.
- If `VX_ACTION_ABANDON` is returned, execution is unspecified for all nodes for which this node is a dominator. Nodes that are dominators of this node will have executed. Execution of any other node is unspecified.

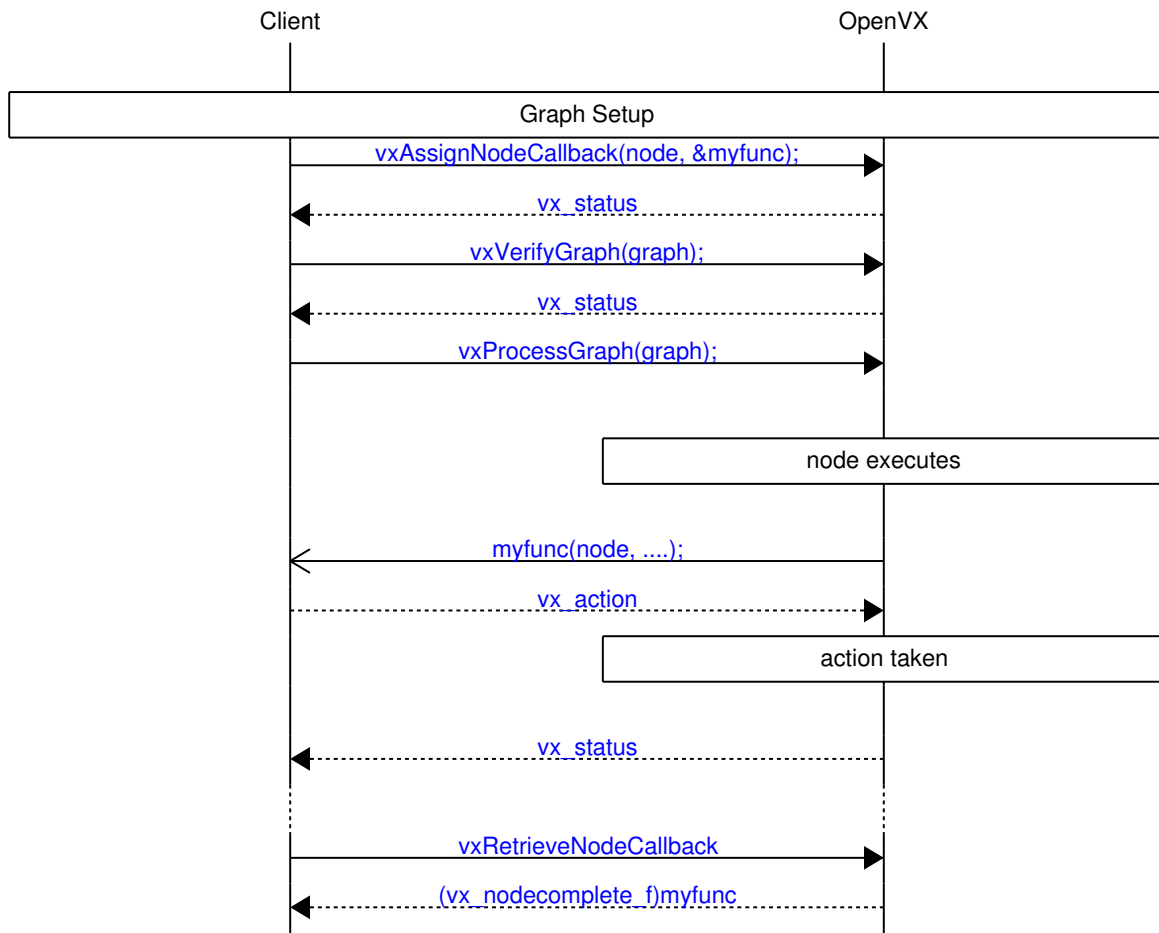


Figure 3.1: Node Callback Sequence

## Typedefs

- typedef `vx_enum vx_action`  
The formal typedef of the response from the callback.
- typedef `vx_action(* vx_nodecomplete_f)(vx_node node)`  
A callback to the client after a particular node has completed.

## Enumerations

- enum `vx_action_e` {  
`VX_ACTION_CONTINUE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_ACTION` << 12)) + 0x0,  
`VX_ACTION_ABANDON` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_ACTION` << 12)) + 0x1 }  
 A return code enumeration from a `vx_nodecomplete_f` during execution.

## Functions

- `vx_status VX_API_CALL vxAssignNodeCallback (vx_node node, vx_nodecomplete_f callback)`  
Assigns a callback to a node. If a callback already exists in this node, this function must return an error and the user may clear the callback by passing a NULL pointer as the callback.
- `vx_nodecomplete_f VX_API_CALL vxRetrieveNodeCallback (vx_node node)`  
Retrieves the current node callback function pointer set on the node.

### 3.67.2 Typedef Documentation

#### typedef vx\_enum vx\_action

The formal typedef of the response from the callback.

See also

[vx\\_action\\_e](#)

Definition at line 407 of file [vx\\_types.h](#).

#### typedef vx\_action( \* vx\_nodecomplete\_f)(vx\_node node)

A callback to the client after a particular node has completed.

See also

[vx\\_action](#)

[vxAssignNodeCallback](#)

Parameters

|           |             |                                              |
|-----------|-------------|----------------------------------------------|
| <i>in</i> | <i>node</i> | The node to which the callback was attached. |
|-----------|-------------|----------------------------------------------|

Returns

An action code from [vx\\_action\\_e](#).

Definition at line 416 of file [vx\\_types.h](#).

### 3.67.3 Enumeration Type Documentation

#### enum vx\_action\_e

A return code enumeration from a [vx\\_nodecomplete\\_f](#) during execution.

See also

[vxAssignNodeCallback](#)

Enumerator

**VX\_ACTION\_CONTINUE** Continue executing the graph with no changes.

**VX\_ACTION\_ABANDON** Stop executing the graph.

Definition at line 548 of file [vx\\_types.h](#).

### 3.67.4 Function Documentation

#### vx\_status VX\_API\_CALL vxAssignNodeCallback ( vx\_node node, vx\_nodecomplete\_f callback )

Assigns a callback to a node. If a callback already exists in this node, this function must return an error and the user may clear the callback by passing a NULL pointer as the callback.

Parameters

|           |                 |                                                                  |
|-----------|-----------------|------------------------------------------------------------------|
| <i>in</i> | <i>node</i>     | The reference to the node.                                       |
| <i>in</i> | <i>callback</i> | The callback to associate with completion of this specific node. |

Warning

This must be used with **extreme** caution as it can *ruin* optimizations in the power/performance efficiency of a graph.

Returns

A [vx\\_status\\_e](#) enumeration.



## Return values

|                                   |                                                              |
|-----------------------------------|--------------------------------------------------------------|
| <i>VX_SUCCESS</i>                 | Callback assigned.                                           |
| <i>VX_ERROR_INVALID_REFERENCE</i> | The value passed as node was not a <a href="#">vx_node</a> . |

**vx\_nodecomplete\_f VX\_API\_CALL vxRetrieveNodeCallback ( vx\_node node )**

Retrieves the current node callback function pointer set on the node.

## Parameters

|    |             |                                                      |
|----|-------------|------------------------------------------------------|
| in | <i>node</i> | The reference to the <a href="#">vx_node</a> object. |
|----|-------------|------------------------------------------------------|

## Returns

vx\_nodecomplete\_f The pointer to the callback function.

## Return values

|             |                             |
|-------------|-----------------------------|
| <i>NULL</i> | No callback is set.         |
| *           | The node callback function. |

## 3.68 Framework: Performance Measurement

### 3.68.1 Detailed Description

Defines Performance measurement and reporting interfaces.

In OpenVX, both `vx_graph` objects and `vx_node` objects track performance information. A client can query either object type using their respective `vxQuery<Object>` function with their attribute enumeration `VX_<Object>_ATTRIBUTE_PERFORMANCE` along with a `vx_perf_t` structure to obtain the performance information.

```
vx_perf_t perf;
vxQueryNode(node, VX_NODE_ATTRIBUTE_PERFORMANCE, &perf, sizeof(perf));
```

### Data Structures

- struct `vx_perf_t`

*The performance measurement structure. [More...](#)*

### 3.68.2 Data Structure Documentation

#### struct `vx_perf_t`

The performance measurement structure.

Definition at line 1328 of file `vx_types.h`.

#### Data Fields

|                        |                  |                                       |
|------------------------|------------------|---------------------------------------|
| <code>vx_uint64</code> | <code>tmp</code> | Holds the last measurement.           |
| <code>vx_uint64</code> | <code>beg</code> | Holds the first measurement in a set. |
| <code>vx_uint64</code> | <code>end</code> | Holds the last measurement in a set.  |
| <code>vx_uint64</code> | <code>sum</code> | Holds the summation of durations.     |
| <code>vx_uint64</code> | <code>avg</code> | Holds the average of the durations.   |
| <code>vx_uint64</code> | <code>min</code> | Holds the minimum of the durations.   |
| <code>vx_uint64</code> | <code>num</code> | Holds the number of measurements.     |
| <code>vx_uint64</code> | <code>max</code> | Holds the maximum of the durations.   |

## 3.69 Framework: Log

### 3.69.1 Detailed Description

Defines the debug logging interface.

The functions of the debugging interface allow clients to receive important debugging information about Open↔ VX.

See also

[vx\\_status\\_e](#) for the list of possible errors.

Figure 3.2: Log messages only can be received after the callback is installed.

### Typedefs

- typedef void(\* [vx\\_log\\_callback\\_f](#) )(vx\_context context, vx\_reference ref, vx\_status status, const vx\_char string[])

*The log callback function.*

### Functions

- void VX\_API\_CALL [vxAddLogEntry](#) (vx\_reference ref, vx\_status status, const char \*message,...)  
*Adds a line to the log.*
- void VX\_API\_CALL [vxRegisterLogCallback](#) (vx\_context context, vx\_log\_callback\_f callback, vx\_bool reentrant)

*Registers a callback facility to the OpenVX implementation to receive error logs.*

### 3.69.2 Function Documentation

**void VX\_API\_CALL vxAddLogEntry ( vx\_reference ref, vx\_status status, const char \* message, ... )**

Adds a line to the log.

Parameters

|    |         |                                                                                       |
|----|---------|---------------------------------------------------------------------------------------|
| in | ref     | The reference to add the log entry against. Some valid value must be provided.        |
| in | status  | The status code. <a href="#">VX_SUCCESS</a> status entries are ignored and not added. |
| in | message | The human readable message to add to the log.                                         |
| in | ...     | a list of variable arguments to the message.                                          |

Note

Messages may not exceed [VX\\_MAX\\_LOG\\_MESSAGE\\_LEN](#) bytes and will be truncated in the log if they exceed this limit.

**void VX\_API\_CALL vxRegisterLogCallback ( vx\_context context, vx\_log\_callback\_f callback, vx\_bool reentrant )**

Registers a callback facility to the OpenVX implementation to receive error logs.

Parameters

|    |         |                                |
|----|---------|--------------------------------|
| in | context | The overall context to OpenVX. |
|----|---------|--------------------------------|

|    |                  |                                                                                                                                                             |
|----|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>callback</i>  | The callback function. If NULL, the previous callback is removed.                                                                                           |
| in | <i>reentrant</i> | If reentrancy flag is <code>vx_true_e</code> , then the callback may be entered from multiple simultaneous tasks or threads (if the host OS supports this). |

## 3.70 Framework: Hints

### 3.70.1 Detailed Description

Defines the Hints Interface.

*Hints* are messages given to the OpenVX implementation that it may support. (These are optional.)

#### Enumerations

- enum `vx_hint_e` { `VX_HINT_SERIALIZE` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_HINT` << 12)) + 0x0 }

*These enumerations are given to the `vxHint` API to enable/disable platform optimizations and/or features. Hints are optional and usually are vendor-specific.*

#### Functions

- `vx_status` `VX_API_CALL vxHint` ( `vx_reference` reference, `vx_enum` hint )

*Provides a generic API to give platform-specific hints to the implementation.*

### 3.70.2 Enumeration Type Documentation

#### enum `vx_hint_e`

These enumerations are given to the `vxHint` API to enable/disable platform optimizations and/or features. Hints are optional and usually are vendor-specific.

See also

`vxHint`

Enumerator

**`VX_HINT_SERIALIZE`** Indicates to the implementation that the user wants to disable any parallelization techniques. Implementations may not be parallelized, so this is a hint only.

Definition at line 572 of file `vx_types.h`.

### 3.70.3 Function Documentation

`vx_status` `VX_API_CALL vxHint` ( `vx_reference` *reference*, `vx_enum` *hint* )

Provides a generic API to give platform-specific hints to the implementation.

Parameters

|    |                  |                                                                                                                                                                                                        |
|----|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>reference</i> | The reference to the object to hint at. This could be <code>vx_context</code> , <code>vx_graph</code> , <code>vx_node</code> , <code>vx_image</code> , <code>vx_array</code> , or any other reference. |
| in | <i>hint</i>      | A <code>vx_hint_e</code> <i>hint</i> to give the OpenVX context. This is a platform-specific optimization or implementation mechanism.                                                                 |

Returns

A `vx_status_e` enumeration.

Return values

|                                         |                                     |
|-----------------------------------------|-------------------------------------|
| <code>VX_SUCCESS</code>                 | No error.                           |
| <code>VX_ERROR_INVALID_REFERENCE</code> | If context or reference is invalid. |

|                                     |                               |
|-------------------------------------|-------------------------------|
| <code>VX_ERROR_NOT_SUPPORTED</code> | If the hint is not supported. |
|-------------------------------------|-------------------------------|

## 3.71 Framework: Directives

### 3.71.1 Detailed Description

Defines the Directives Interface.

*Directives* are messages given the OpenVX implementation that it must support. (These are required, i.e., non-optional.)

#### Enumerations

- enum `vx_directive_e` {  
`VX_DIRECTIVE_DISABLE_LOGGING` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_DIRECTIVE` << 12)) + 0x0,  
`VX_DIRECTIVE_ENABLE_LOGGING` = ((( `VX_ID_KHRONOS` ) << 20) | ( `VX_ENUM_DIRECTIVE` << 12)) + 0x1 }

*These enumerations are given to the `vxDirective` API to enable/disable platform optimizations and/or features. Directives are not optional and usually are vendor-specific, by defining a vendor range of directives and starting their enumeration from there.*

#### Functions

- `vx_status` `VX_API_CALL vxDirective (vx_reference reference, vx_enum directive)`

*Provides a generic API to give platform-specific directives to the implementations.*

### 3.71.2 Enumeration Type Documentation

#### enum `vx_directive_e`

These enumerations are given to the `vxDirective` API to enable/disable platform optimizations and/or features. Directives are not optional and usually are vendor-specific, by defining a vendor range of directives and starting their enumeration from there.

See also

`vxDirective`

Enumerator

**`VX_DIRECTIVE_DISABLE_LOGGING`** Disables recording information for graph debugging.

**`VX_DIRECTIVE_ENABLE_LOGGING`** Enables recording information for graph debugging.

Definition at line 587 of file `vx_types.h`.

### 3.71.3 Function Documentation

#### `vx_status` `VX_API_CALL vxDirective ( vx_reference reference, vx_enum directive )`

Provides a generic API to give platform-specific directives to the implementations.

Parameters

|    |                  |                                                                                                                                                                                                                     |
|----|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>reference</i> | The reference to the object to set the directive on. This could be <code>vx_context</code> , <code>vx_graph</code> , <code>vx_node</code> , <code>vx_image</code> , <code>vx_array</code> , or any other reference. |
| in | <i>directive</i> | The directive to set.                                                                                                                                                                                               |

Returns

A `vx_status_e` enumeration.

## Return values

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| <i>VX_SUCCESS</i>                 | No error.                           |
| <i>VX_ERROR_INVALID_REFERENCE</i> | If context or reference is invalid. |
| <i>VX_ERROR_NOT_SUPPORTED</i>     | If the directive is not supported.  |



## 3.72 Framework: User Kernels

### 3.72.1 Detailed Description

Defines the User Kernels, which are a method to extend OpenVX with new vision functions.

User Kernels can be loaded by OpenVX and included as nodes in the graph or as immediate functions (if the Client supplies the interface). User Kernels will typically be loaded and executed on HLOS/CPU compatible targets, not on remote processors or other accelerators. This specification does not mandate what constitutes compatible platforms.

Figure 3.3: Call sequence of User Kernels Installation

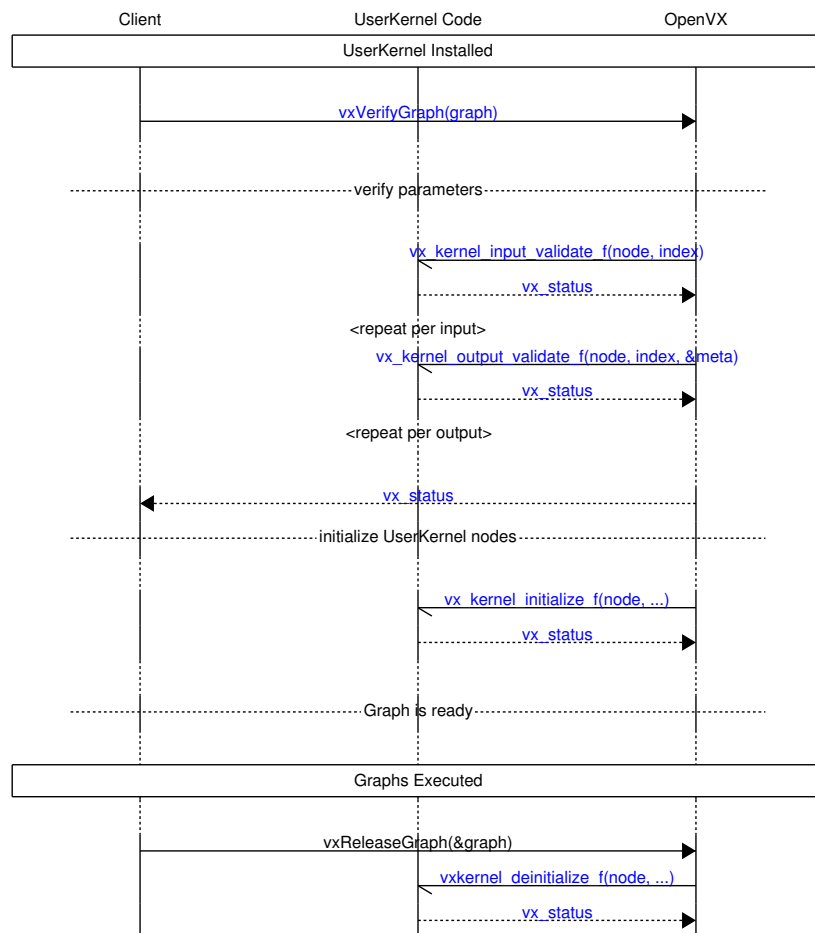


Figure 3.4: Call sequence of a Graph Verify and Release with User Kernels.

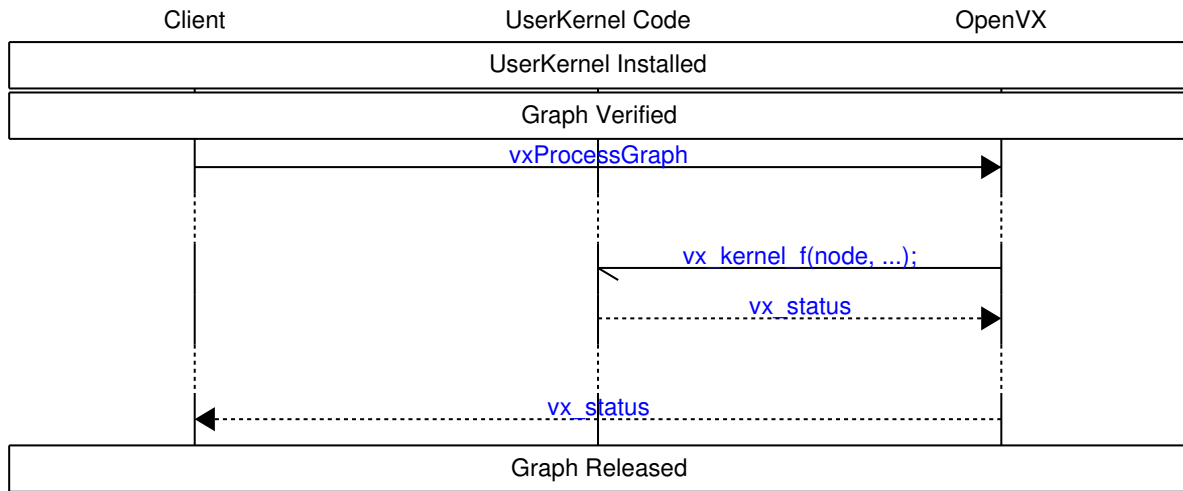


Figure 3.5: Call sequence of a Graph Execution with User Kernels

## Typedefs

- typedef `vx_status(* vx_kernel_deinitialize_f)(vx_node node, const vx_reference *parameters, vx_uint32 num)`  
The pointer to the kernel deinitializer. If the host code requires a call to deinitialize data during a node garbage collection, this function is called if not NULL.
- typedef `vx_status(* vx_kernel_f)(vx_node node, const vx_reference *parameters, vx_uint32 num)`  
The pointer to the Host side kernel.
- typedef `vx_status(* vx_kernel_initialize_f)(vx_node node, const vx_reference *parameters, vx_uint32 num)`  
The pointer to the kernel initializer. If the host code requires a call to initialize data once all the parameters have been validated, this function is called if not NULL.
- typedef `vx_status(* vx_kernel_input_validate_f)(vx_node node, vx_uint32 index)`  
The user-defined kernel node input parameter validation function.
- typedef `vx_status(* vx_kernel_output_validate_f)(vx_node node, vx_uint32 index, vx_meta_format meta)`  
The user-defined kernel node output parameter validation function. The function only needs to fill in the meta data structure.
- typedef struct `_vx_meta_format * vx_meta_format`  
This object is used by output validation functions to specify the meta data of the expected output data object. If the output object is an image, the `vx_meta_format` object can additionally store the valid region delta rectangle.
- typedef `vx_status(* vx_publish_kernels_f)(vx_context context)`  
The entry point into modules loaded by `vxLoadKernels`.

## Enumerations

- enum `vx_meta_format_attribute_e { VX_META_FORMAT_ATTRIBUTE_DELTA_RECTANGLE = ((( VX_I↵  
D_KHRONOS ) << 20) | ( VX_TYPE_META_FORMAT << 8)) + 0x0 }`  
The meta format object attributes.

## Functions

- `vx_kernel VX_API_CALL vxAddKernel (vx_context context, const vx_char name[VX_MAX_KERNEL_NA↵  
ME], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_input_validate_f input,  
vx_kernel_output_validate_f output, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit)`  
Allows users to add custom kernels to the known kernel database in OpenVX at run-time. This would primarily be used by the module function `vxPublishKernels`.
- `vx_status VX_API_CALL vxAddParameterToKernel (vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_↵  
enum data_type, vx_enum state)`

Allows users to set the signatures of the custom kernel.

- **vx\_status** VX\_API\_CALL **vxFinalizeKernel** (**vx\_kernel** kernel)

This API is called after all parameters have been added to the kernel and the kernel is ready to be used. Notice that the reference to the kernel created by **vxAddKernel** is still valid after the call to **vxFinalizeKernel**.

- **vx\_status** VX\_API\_CALL **vxLoadKernels** (**vx\_context** context, const **vx\_char** \*module)

Loads one or more kernels into the OpenVX context. This is the interface by which OpenVX is extensible. Once the set of kernels is loaded new kernels and their parameters can be queried.

- **vx\_status** VX\_API\_CALL **vxRemoveKernel** (**vx\_kernel** kernel)

Removes a non-finalized **vx\_kernel** from the **vx\_context** and releases it. Once a **vx\_kernel** has been finalized it cannot be removed.

- **vx\_status** VX\_API\_CALL **vxSetKernelAttribute** (**vx\_kernel** kernel, **vx\_enum** attribute, const void \*ptr, **vx\_size** size)

Sets kernel attributes.

- **vx\_status** VX\_API\_CALL **vxSetMetaFormatAttribute** (**vx\_meta\_format** meta, **vx\_enum** attribute, const void \*ptr, **vx\_size** size)

This function allows a user to set the attributes of a **vx\_meta\_format** object in a kernel output validator.

### 3.72.2 Typedef Documentation

**typedef struct \_vx\_meta\_format\* vx\_meta\_format**

This object is used by output validation functions to specify the meta data of the expected output data object. If the output object is an image, the **vx\_meta\_format** object can additionally store the valid region delta rectangle.

Note

when the actual output object of the user node is virtual, the information given through the **vx\_meta\_format** object allows the OpenVX framework to automatically create the data object when meta data were not specified by the application at object creation time.

Definition at line 289 of file **vx\_types.h**.

**typedef vx\_status( \* vx\_publish\_kernels\_f)(vx\_context context)**

The entry point into modules loaded by **vxLoadKernels**.

Parameters

|    |         |                                           |
|----|---------|-------------------------------------------|
| in | context | The handle to the implementation context. |
|----|---------|-------------------------------------------|

Note

The symbol exported from the user module must be **vxPublishKernels** in extern C format.

Definition at line 1189 of file **vx\_types.h**.

**typedef vx\_status( \* vx\_kernel\_f)(vx\_node node, const vx\_reference \*parameters, vx\_uint32 num)**

The pointer to the Host side kernel.

Parameters

|    |            |                                                   |
|----|------------|---------------------------------------------------|
| in | node       | The handle to the node that contains this kernel. |
| in | parameters | The array of parameter references.                |
| in | num        | The number of parameters.                         |

Definition at line 1198 of file **vx\_types.h**.

**typedef vx\_status( \* vx\_kernel\_initialize\_f)(vx\_node node, const vx\_reference \*parameters, vx\_uint32 num)**

The pointer to the kernel initializer. If the host code requires a call to initialize data once all the parameters have been validated, this function is called if not NULL.

**Parameters**

|    |                   |                                                   |
|----|-------------------|---------------------------------------------------|
| in | <i>node</i>       | The handle to the node that contains this kernel. |
| in | <i>parameters</i> | The array of parameter references.                |
| in | <i>num</i>        | The number of parameters.                         |

Definition at line 1209 of file [vx\\_types.h](#).

**typedef vx\_status( \* vx\_kernel\_deinitialize\_f)(vx\_node node, const vx\_reference \*parameters, vx\_uint32 num)**

The pointer to the kernel deinitializer. If the host code requires a call to deinitialize data during a node garbage collection, this function is called if not NULL.

**Parameters**

|    |                   |                                                   |
|----|-------------------|---------------------------------------------------|
| in | <i>node</i>       | The handle to the node that contains this kernel. |
| in | <i>parameters</i> | The array of parameter references.                |
| in | <i>num</i>        | The number of parameters.                         |

Definition at line 1220 of file [vx\\_types.h](#).

**typedef vx\_status( \* vx\_kernel\_input\_validate\_f)(vx\_node node, vx\_uint32 index)**

The user-defined kernel node input parameter validation function.

**Note**

This function is called once for each VX\_INPUT or VX\_BIDIRECTIONAL parameter index.

**Parameters**

|    |              |                                                 |
|----|--------------|-------------------------------------------------|
| in | <i>node</i>  | The handle to the node that is being validated. |
| in | <i>index</i> | The index of the parameter being validated.     |

**Returns**

An error code describing the validation status on this parameter.

**Return values**

|                                    |                                                    |
|------------------------------------|----------------------------------------------------|
| <i>VX_ERROR_INVALID_FORMAT</i>     | The parameter format was incorrect.                |
| <i>VX_ERROR_INVALID_VALUE</i>      | The value of the parameter was incorrect.          |
| <i>VX_ERROR_INVALID_DIMENSION</i>  | The dimensionality of the parameter was incorrect. |
| <i>VX_ERROR_INVALID_PARAMETERS</i> | The index was out of bounds.                       |

Definition at line 1236 of file [vx\\_types.h](#).

**typedef vx\_status( \* vx\_kernel\_output\_validate\_f)(vx\_node node, vx\_uint32 index, vx\_meta\_format meta)**

The user-defined kernel node output parameter validation function. The function only needs to fill in the meta data structure.

**Note**

This function is called once for each VX\_OUTPUT parameter index.

**Parameters**

|    |              |                                                                                                                                                                                                                                |
|----|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>node</i>  | The handle to the node that is being validated.                                                                                                                                                                                |
| in | <i>index</i> | The index of the parameter being validated.                                                                                                                                                                                    |
| in | <i>ptr</i>   | A pointer to a pre-allocated structure that the system holds. The validation function fills in the correct type, format, and dimensionality for the system to use either to create memory or to check against existing memory. |

**Returns**

An error code describing the validation status on this parameter.

**Return values**

|                                           |                             |
|-------------------------------------------|-----------------------------|
| <b><i>VX_ERROR_INVALID_PARAMETERS</i></b> | The index is out of bounds. |
|-------------------------------------------|-----------------------------|

Definition at line 1252 of file [vx\\_types.h](#).

**3.72.3 Enumeration Type Documentation****enum vx\_meta\_format\_attribute\_e**

The meta format object attributes.

**Enumerator**

***VX\_META\_FORMAT\_ATTRIBUTE\_DELTA\_RECTANGLE*** Configures a delta rectangle during kernel output parameter validation. Use a [vx\\_delta\\_rectangle\\_t](#).

Definition at line 978 of file [vx\\_types.h](#).

**3.72.4 Function Documentation****vx\_status VX\_API\_CALL vxLoadKernels ( vx\_context *context*, const vx\_char \* *module* )**

Loads one or more kernels into the OpenVX context. This is the interface by which OpenVX is extensible. Once the set of kernels is loaded new kernels and their parameters can be queried.

**Note**

When all references to loaded kernels are released, the module may be automatically unloaded.

**Parameters**

|    |                |                                                                                                                                                                                                                                                                                                                           |
|----|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>context</i> | The reference to the implementation context.                                                                                                                                                                                                                                                                              |
| in | <i>module</i>  | The short name of the module to load. On systems where there are specific naming conventions for modules, the name passed should ignore such conventions. For example: <code>libxyz.so</code> should be passed as just <code>xyz</code> and the implementation will <i>do the right thing</i> that the platform requires. |

**Note**

This API uses the system pre-defined paths for modules.

**Returns**

A [vx\\_status\\_e](#) enumeration.

## Return values

|                                          |                                                   |
|------------------------------------------|---------------------------------------------------|
| <code>VX_SUCCESS</code>                  | No errors.                                        |
| <code>VX_ERROR_INVALID_REFERENCE</code>  | If the context is not a <code>vx_context</code> . |
| <code>VX_ERROR_INVALID_PARAMETERS</code> | If any of the other parameters are incorrect.     |

See also

[vxGetKernelByName](#)

**`vx_kernel VX_API_CALL vxAddKernel ( vx_context context, const vx_char name[VX_MAX_KERNEL_NAME], vx_enum enumeration, vx_kernel_f func_ptr, vx_uint32 numParams, vx_kernel_input_validate_f input, vx_kernel_output_validate_f output, vx_kernel_initialize_f init, vx_kernel_deinitialize_f deinit )`**

Allows users to add custom kernels to the known kernel database in OpenVX at run-time. This would primarily be used by the module function `vxPublishKernels`.

## Parameters

|    |                          |                                                                                                                 |
|----|--------------------------|-----------------------------------------------------------------------------------------------------------------|
| in | <code>context</code>     | The reference to the implementation context.                                                                    |
| in | <code>name</code>        | The string to use to match the kernel.                                                                          |
| in | <code>enumeration</code> | The enumerated value of the kernel to be used by clients.                                                       |
| in | <code>func_ptr</code>    | The process-local function pointer to be invoked.                                                               |
| in | <code>numParams</code>   | The number of parameters for this kernel.                                                                       |
| in | <code>input</code>       | The pointer to <code>vx_kernel_input_validate_f</code> , which validates the input parameters to this kernel.   |
| in | <code>output</code>      | The pointer to <code>vx_kernel_output_validate_f</code> , which validates the output parameters to this kernel. |
| in | <code>init</code>        | The kernel initialization function.                                                                             |
| in | <code>deinit</code>      | The kernel de-initialization function.                                                                          |

## Returns

`vx_kernel`

## Return values

|                |                                                          |
|----------------|----------------------------------------------------------|
| <code>0</code> | Indicates that an error occurred when adding the kernel. |
| <code>*</code> | Kernel added to OpenVX.                                  |

**`vx_status VX_API_CALL vxFinalizeKernel ( vx_kernel kernel )`**

This API is called after all parameters have been added to the kernel and the kernel is *ready* to be used. Notice that the reference to the kernel created by `vxAddKernel` is still valid after the call to `vxFinalizeKernel`.

## Parameters

|    |                     |                                                                    |
|----|---------------------|--------------------------------------------------------------------|
| in | <code>kernel</code> | The reference to the loaded kernel from <code>vxAddKernel</code> . |
|----|---------------------|--------------------------------------------------------------------|

## Returns

A `vx_status_e` enumeration. If an error occurs, the kernel is not available for usage by the clients of OpenVX. Typically this is due to a mismatch between the number of parameters requested and given.

## Precondition

[vxAddKernel](#) and [vxAddParameterToKernel](#)

**`vx_status VX_API_CALL vxAddParameterToKernel ( vx_kernel kernel, vx_uint32 index, vx_enum dir, vx_enum data_type, vx_enum state )`**

Allows users to set the signatures of the custom kernel.

**Parameters**

|    |                  |                                                                                                                                                                                  |
|----|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>kernel</i>    | The reference to the kernel added with <a href="#">vxAddKernel</a> .                                                                                                             |
| in | <i>index</i>     | The index of the parameter to add.                                                                                                                                               |
| in | <i>dir</i>       | The direction of the parameter. This must be either <a href="#">VX_INPUT</a> or <a href="#">VX_OUTPUT</a> . <a href="#">VX_BIDIRECTIONAL</a> is not supported for this function. |
| in | <i>data_type</i> | The type of parameter. This must be a value from <a href="#">vx_type_e</a> .                                                                                                     |
| in | <i>state</i>     | The state of the parameter (required or not). This must be a value from <a href="#">vx_parameter_state_e</a> .                                                                   |

**Returns**

A [vx\\_status\\_e](#) enumerated value.

**Return values**

|                                            |                                                                  |
|--------------------------------------------|------------------------------------------------------------------|
| <a href="#">VX_SUCCESS</a>                 | Parameter is successfully set on kernel.                         |
| <a href="#">VX_ERROR_INVALID_REFERENCE</a> | The value passed as kernel was not a <a href="#">vx_kernel</a> . |

**Precondition**

[vxAddKernel](#)

**vx\_status VX\_API\_CALL vxRemoveKernel ( vx\_kernel kernel )**

Removes a non-finalized [vx\\_kernel](#) from the [vx\\_context](#) and releases it. Once a [vx\\_kernel](#) has been finalized it cannot be removed.

**Parameters**

|    |               |                                                                                    |
|----|---------------|------------------------------------------------------------------------------------|
| in | <i>kernel</i> | The reference to the kernel to remove. Returned from <a href="#">vxAddKernel</a> . |
|----|---------------|------------------------------------------------------------------------------------|

**Note**

Any kernel enumerated in the base standard cannot be removed; only kernels added through [vxAddKernel](#) can be removed.

**Returns**

A [vx\\_status\\_e](#) enumeration.

**Return values**

|                                            |                                    |
|--------------------------------------------|------------------------------------|
| <a href="#">VX_ERROR_INVALID_REFERENCE</a> | If an invalid kernel is passed in. |
| <a href="#">VX_ERROR_INVALID_PARAMETER</a> | If a base kernel is passed in.     |

**vx\_status VX\_API\_CALL vxSetKernelAttribute ( vx\_kernel kernel, vx\_enum attribute, const void \* ptr, vx\_size size )**

Sets kernel attributes.

**Parameters**

|    |                  |                                                                                |
|----|------------------|--------------------------------------------------------------------------------|
| in | <i>kernel</i>    | The reference to the kernel.                                                   |
| in | <i>attribute</i> | The enumeration of the attributes. See <a href="#">vx_kernel_attribute_e</a> . |

|    |             |                                                                      |
|----|-------------|----------------------------------------------------------------------|
| in | <i>ptr</i>  | The pointer to the location from which to read the attribute.        |
| in | <i>size</i> | The size in bytes of the data area indicated by <i>ptr</i> in bytes. |

**Note**

After a kernel has been passed to `vxFinalizeKernel`, no attributes can be altered.

**Returns**

A `vx_status_e` enumeration.

**`vx_status VX_API_CALL vxSetMetaFormatAttribute ( vx_meta_format meta, vx_enum attribute, const void * ptr, vx_size size )`**

This function allows a user to set the attributes of a `vx_meta_format` object in a kernel output validator.

The `vx_meta_format` object contains two types of information : data object meta data and some specific information that defines how the valid region of an image changes

The meta data attributes that can be set are identified by this list:

- `vx_image` : `VX_IMAGE_ATTRIBUTE_FORMAT`, `VX_IMAGE_ATTRIBUTE_HEIGHT`, `VX_IMAGE_ATTRIBUTE_WIDTH`
- `vx_array` : `VX_ARRAY_ATTRIBUTE_CAPACITY`, `VX_ARRAY_ATTRIBUTE_ITEMTYPE`
- `vx_pyramid` : `VX_PYRAMID_ATTRIBUTE_FORMAT`, `VX_PYRAMID_ATTRIBUTE_HEIGHT`, `VX_PYRAMID_ATTRIBUTE_WIDTH`, `VX_PYRAMID_ATTRIBUTE_LEVELS`, `VX_PYRAMID_ATTRIBUTE_SCALE`
- `vx_scalar` : `VX_SCALAR_ATTRIBUTE_TYPE`
- `vx_matrix` : `VX_MATRIX_ATTRIBUTE_TYPE`, `VX_MATRIX_ATTRIBUTE_ROWS`, `VX_MATRIX_ATTRIBUTE_COLUMNS`
- `vx_distribution` : `VX_DISTRIBUTION_ATTRIBUTE_BINS`, `VX_DISTRIBUTION_ATTRIBUTE_OFFSET`, `VX_DISTRIBUTION_ATTRIBUTE_RANGE`
- `vx_remap` : `VX_REMAP_ATTRIBUTE_SOURCE_WIDTH`, `VX_REMAP_ATTRIBUTE_SOURCE_HEIGHT`, `VX_REMAP_ATTRIBUTE_DESTINATION_WIDTH`, `VX_REMAP_ATTRIBUTE_DESTINATION_HEIGHT`
- `vx_lut` : `VX_LUT_ATTRIBUTE_TYPE`, `VX_LUT_ATTRIBUTE_COUNT`
- `vx_threshold` : `VX_THRESHOLD_ATTRIBUTE_TYPE`
- `VX_META_FORMAT_ATTRIBUTE_DELTA_RECTANGLE`

**Note**

For `vx_image`, a specific attribute can be used to specify the valid region evolution. This information is not a meta data.

**Parameters**

|    |                  |                                                                                                                                                |
|----|------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>meta</i>      | The reference to the <code>vx_meta_format</code> struct to set                                                                                 |
| in | <i>attribute</i> | Use the subset of data object attributes that define the meta data of this object or attributes from <code>vx_meta_format_attribute_e</code> . |
| in | <i>ptr</i>       | The input pointer of the value to set on the meta format object.                                                                               |
| in | <i>size</i>      | The size in bytes of the object to which <i>ptr</i> points.                                                                                    |

**Returns**

A `vx_status_e` enumeration.



## Return values

|                                   |                                                                   |
|-----------------------------------|-------------------------------------------------------------------|
| <i>VX_SUCCESS</i>                 | The attribute was set.                                            |
| <i>VX_ERROR_INVALID_REFERENCE</i> | meta was not a <code>vx_meta_format</code> .                      |
| <i>VX_ERROR_INVALID_PARAMETER</i> | size was not correct for the type needed.                         |
| <i>VX_ERROR_NOT_SUPPORTED</i>     | the object attribute was not supported on the meta format object. |
| <i>VX_ERROR_INVALID_TYPE</i>      | attribute type did not match known meta format type.              |

## 3.73 Framework: Graph Parameters

### 3.73.1 Detailed Description

Defines the Graph Parameter API.

Graph parameters allow Clients to create graphs with Client settable parameters. Clients can then create Graph creation methods (a.k.a. *Graph Factories*). When creating these factories, the client will typically not be able to use the standard Node creator functions such as `vxSobel13x3Node` but instead will use the *manual* method via `vxCreateGenericNode`.

```
vx_graph vxCornersGraphFactory(vx_context context)
{
 vx_status status = VX_SUCCESS;
 vx_uint32 i;
 vx_float32 strength_thresh = 10000.0f;
 vx_float32 r = 1.5f;
 vx_float32 sensitivity = 0.14f;
 vx_int32 window_size = 3;
 vx_int32 block_size = 3;
 vx_enum channel = VX_CHANNEL_Y;
 vx_graph graph = vxCreateGraph(context);
 if (vxGetStatus((vx_reference)graph) == VX_SUCCESS)
 {
 vx_image virts[] = {
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
 vxCreateVirtualImage(graph, 0, 0,
 VX_DF_IMAGE_VIRT),
 };
 vx_kernel kernels[] = {
 vxGetKernelByEnum(context,
 VX_KERNEL_CHANNEL_EXTRACT),
 vxGetKernelByEnum(context, VX_KERNEL_MEDIAN_3x3),
 vxGetKernelByEnum(context, VX_KERNEL_HARRIS_CORNERS),
 };
 vx_node nodes[dimof(kernels)] = {
 vxCreateGenericNode(graph, kernels[0]),
 vxCreateGenericNode(graph, kernels[1]),
 vxCreateGenericNode(graph, kernels[2]),
 };
 vx_scalar scalars[] = {
 vxCreateScalar(context, VX_TYPE_ENUM, &channel),
 vxCreateScalar(context, VX_TYPE_FLOAT32, &strength_thresh),
 vxCreateScalar(context, VX_TYPE_FLOAT32, &r),
 vxCreateScalar(context, VX_TYPE_FLOAT32, &sensitivity),
 vxCreateScalar(context, VX_TYPE_INT32, &window_size),
 vxCreateScalar(context, VX_TYPE_INT32, &block_size),
 };
 vx_parameter parameters[] = {
 vxGetParameterByIndex(nodes[0], 0),
 vxGetParameterByIndex(nodes[2], 6)
 };
 // Channel Extract
 status |= vxAddParameterToGraph(graph, parameters[0]);
 status |= vxSetParameterByIndex(nodes[0], 1, (
 vx_reference)scalars[0]);
 status |= vxSetParameterByIndex(nodes[0], 2, (
 vx_reference)virts[0]);
 // Median Filter
 status |= vxSetParameterByIndex(nodes[1], 0, (
 vx_reference)virts[0]);
 status |= vxSetParameterByIndex(nodes[1], 1, (
 vx_reference)virts[1]);
 // Harris Corners
 status |= vxSetParameterByIndex(nodes[2], 0, (
 vx_reference)virts[1]);
 status |= vxSetParameterByIndex(nodes[2], 1, (
 vx_reference)scalars[1]);
 status |= vxSetParameterByIndex(nodes[2], 2, (
 vx_reference)scalars[2]);
 status |= vxSetParameterByIndex(nodes[2], 3, (
 vx_reference)scalars[3]);
 status |= vxSetParameterByIndex(nodes[2], 4, (
 vx_reference)scalars[4]);
 status |= vxSetParameterByIndex(nodes[2], 5, (
 vx_reference)scalars[5]);
 status |= vxAddParameterToGraph(graph, parameters[1]);

 for (i = 0; i < dimof(scalars); i++)
 {
 vxReleaseScalar(&scalars[i]);
 }
 for (i = 0; i < dimof(virts); i++)
```

```

 {
 vxReleaseImage(&virt[s[i]]);
 }
 for (i = 0; i < dimof(kernels); i++)
 {
 vxReleaseKernel(&kernels[i]);
 }
 for (i = 0; i < dimof(nodes); i++)
 {
 vxReleaseNode(&nodes[i]);
 }
 for (i = 0; i < dimof(parameters); i++)
 {
 vxReleaseParameter(¶meters[i]);
 }
}
return graph;
}

```

Some data are contained in these Graphs and do not become exposed to Clients of the factory. This allows ISVs or Vendors to create custom IP or IP-sensitive factories that Clients can use but may not be able to determine what is inside the factory. As the graph contains internal references to the data, the objects will not be freed until the graph itself is released.

## Functions

- **vx\_status** VX\_API\_CALL **vxAddParameterToGraph** (**vx\_graph** graph, **vx\_parameter** parameter)  
*Adds the given parameter extracted from a vx\_node to the graph.*
- **vx\_parameter** VX\_API\_CALL **vxGetGraphParameterByIndex** (**vx\_graph** graph, **vx\_uint32** index)  
*Retrieves a vx\_parameter from a vx\_graph.*
- **vx\_status** VX\_API\_CALL **vxSetGraphParameterByIndex** (**vx\_graph** graph, **vx\_uint32** index, **vx\_reference** value)  
*Sets a reference to the parameter on the graph. The implementation must set this parameter on the originating node as well.*

### 3.73.2 Function Documentation

**vx\_status** VX\_API\_CALL **vxAddParameterToGraph** ( **vx\_graph** graph, **vx\_parameter** parameter )

Adds the given parameter extracted from a vx\_node to the graph.

Parameters

|    |           |                                                            |
|----|-----------|------------------------------------------------------------|
| in | graph     | The graph reference that contains the node.                |
| in | parameter | The parameter reference to add to the graph from the node. |

Returns

A vx\_status\_e enumeration.

Return values

|                            |                                               |
|----------------------------|-----------------------------------------------|
| VX_SUCCESS                 | Parameter added to Graph.                     |
| VX_ERROR_INVALID_REFERENCE | The parameter is not a valid vx_parameter.    |
| VX_ERROR_INVALID_PARAMETER | The parameter is of a node not in this graph. |

**vx\_status** VX\_API\_CALL **vxSetGraphParameterByIndex** ( **vx\_graph** graph, **vx\_uint32** index, **vx\_reference** value )

Sets a reference to the parameter on the graph. The implementation must set this parameter on the originating node as well.

**Parameters**

|    |              |                                        |
|----|--------------|----------------------------------------|
| in | <i>graph</i> | The graph reference.                   |
| in | <i>index</i> | The parameter index.                   |
| in | <i>value</i> | The reference to set to the parameter. |

**Returns**

A `vx_status_e` enumeration.

**Return values**

|                                         |                                                                         |
|-----------------------------------------|-------------------------------------------------------------------------|
| <code>VX_SUCCESS</code>                 | Parameter set to Graph.                                                 |
| <code>VX_ERROR_INVALID_REFERENCE</code> | The value is not a valid <code>vx_reference</code> .                    |
| <code>VX_ERROR_INVALID_PARAMETER</code> | The parameter index is out of bounds or the dir parameter is incorrect. |

**vx\_parameter VX\_API\_CALL vxGetGraphParameterByIndex ( vx\_graph *graph*, vx\_uint32 *index* )**

Retrieves a `vx_parameter` from a `vx_graph`.

**Parameters**

|    |              |                             |
|----|--------------|-----------------------------|
| in | <i>graph</i> | The graph.                  |
| in | <i>index</i> | The index of the parameter. |

**Returns**

`vx_parameter` reference.

**Return values**

|                |                                |
|----------------|--------------------------------|
| <code>0</code> | if the index is out of bounds. |
| <code>*</code> | The parameter reference.       |

# Bibliography

- [1] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm, 2000. [83](#)
- [2] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986. [46](#)
- [3] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006. [26](#), [64](#)
- [4] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:105–119, October 2010. [26](#), [64](#)

# Index

- Absolute Difference, [28](#)
- Accumulate, [29](#)
- Accumulate Squared, [30](#)
- Accumulate Weighted, [32](#)
- Administrative Features, [192](#)
- Advanced Objects, [193](#)
- Arithmetic Addition, [34](#)
- Arithmetic Subtraction, [36](#)
- Basic Features, [106](#)
  - VX\_CHANNEL\_0, [118](#)
  - VX\_CHANNEL\_1, [118](#)
  - VX\_CHANNEL\_2, [118](#)
  - VX\_CHANNEL\_3, [118](#)
  - VX\_CHANNEL\_A, [118](#)
  - VX\_CHANNEL\_B, [118](#)
  - VX\_CHANNEL\_G, [118](#)
  - VX\_CHANNEL\_R, [118](#)
  - VX\_CHANNEL\_U, [118](#)
  - VX\_CHANNEL\_V, [118](#)
  - VX\_CHANNEL\_Y, [118](#)
  - VX\_CONVERT\_POLICY\_SATURATE, [117](#)
  - VX\_CONVERT\_POLICY\_WRAP, [117](#)
  - VX\_DF\_IMAGE\_IYUV, [117](#)
  - VX\_DF\_IMAGE\_NV12, [117](#)
  - VX\_DF\_IMAGE\_NV21, [117](#)
  - VX\_DF\_IMAGE\_RGB, [117](#)
  - VX\_DF\_IMAGE\_RGBX, [117](#)
  - VX\_DF\_IMAGE\_S16, [118](#)
  - VX\_DF\_IMAGE\_S32, [118](#)
  - VX\_DF\_IMAGE\_U16, [118](#)
  - VX\_DF\_IMAGE\_U32, [118](#)
  - VX\_DF\_IMAGE\_U8, [117](#)
  - VX\_DF\_IMAGE\_UYVY, [117](#)
  - VX\_DF\_IMAGE\_VIRT, [117](#)
  - VX\_DF\_IMAGE\_YUV4, [117](#)
  - VX\_DF\_IMAGE\_YUYV, [117](#)
  - VX\_ENUM\_ACCESSOR, [117](#)
  - VX\_ENUM\_ACTION, [116](#)
  - VX\_ENUM\_BORDER\_MODE, [117](#)
  - VX\_ENUM\_CHANNEL, [117](#)
  - VX\_ENUM\_COLOR\_RANGE, [117](#)
  - VX\_ENUM\_COLOR\_SPACE, [116](#)
  - VX\_ENUM\_COMPARISON, [117](#)
  - VX\_ENUM\_CONVERT\_POLICY, [117](#)
  - VX\_ENUM\_DIRECTION, [116](#)
  - VX\_ENUM\_DIRECTIVE, [116](#)
  - VX\_ENUM\_HINT, [116](#)
  - VX\_ENUM\_IMPORT\_MEM, [117](#)
  - VX\_ENUM\_INTERPOLATION, [116](#)
  - VX\_ENUM\_NORM\_TYPE, [117](#)
  - VX\_ENUM\_OVERFLOW, [116](#)
  - VX\_ENUM\_PARAMETER\_STATE, [117](#)
  - VX\_ENUM\_ROUND\_POLICY, [117](#)
  - VX\_ENUM\_TERM\_CRITERIA, [117](#)
  - VX\_ENUM\_THRESHOLD\_TYPE, [117](#)
  - VX\_ERROR\_GRAPH\_ABANDONED, [115](#)
  - VX\_ERROR\_GRAPH\_SCHEDULED, [115](#)
  - VX\_ERROR\_INVALID\_DIMENSION, [115](#)
  - VX\_ERROR\_INVALID\_FORMAT, [115](#)
  - VX\_ERROR\_INVALID\_GRAPH, [115](#)
  - VX\_ERROR\_INVALID\_LINK, [115](#)
  - VX\_ERROR\_INVALID\_MODULE, [116](#)
  - VX\_ERROR\_INVALID\_NODE, [115](#)
  - VX\_ERROR\_INVALID\_PARAMETERS, [116](#)
  - VX\_ERROR\_INVALID\_REFERENCE, [116](#)
  - VX\_ERROR\_INVALID\_SCOPE, [115](#)
  - VX\_ERROR\_INVALID\_TYPE, [115](#)
  - VX\_ERROR\_INVALID\_VALUE, [115](#)
  - VX\_ERROR\_MULTIPLE\_WRITERS, [115](#)
  - VX\_ERROR\_NO\_MEMORY, [116](#)
  - VX\_ERROR\_NO\_RESOURCES, [116](#)
  - VX\_ERROR\_NOT\_ALLOCATED, [116](#)
  - VX\_ERROR\_NOT\_COMPATIBLE, [116](#)
  - VX\_ERROR\_NOT\_IMPLEMENTED, [116](#)
  - VX\_ERROR\_NOT\_SUFFICIENT, [116](#)
  - VX\_ERROR\_NOT\_SUPPORTED, [116](#)
  - VX\_ERROR\_OPTIMIZED\_AWAY, [116](#)
  - VX\_ERROR\_REFERENCE\_NONZERO, [115](#)
  - VX\_FAILURE, [116](#)
  - VX\_ID\_AMD, [119](#)
  - VX\_ID\_ARM, [119](#)
  - VX\_ID\_AXIS, [119](#)
  - VX\_ID\_BDTI, [119](#)
  - VX\_ID\_BROADCOM, [119](#)
  - VX\_ID\_CEVA, [119](#)
  - VX\_ID\_COGNIVUE, [119](#)
  - VX\_ID\_DEFAULT, [119](#)
  - VX\_ID\_FREESCALE, [119](#)
  - VX\_ID\_IMAGINATION, [119](#)
  - VX\_ID\_INTEL, [119](#)
  - VX\_ID\_ITSEEZ, [119](#)
  - VX\_ID\_KHRONOS, [119](#)
  - VX\_ID\_MARVELL, [119](#)
  - VX\_ID\_MEDIATEK, [119](#)
  - VX\_ID\_MOVIDIUS, [119](#)
  - VX\_ID\_NVIDIA, [119](#)
  - VX\_ID\_QUALCOMM, [119](#)
  - VX\_ID\_RENESAS, [119](#)

- VX\_ID\_SAMSUNG, [119](#)
- VX\_ID\_ST, [119](#)
- VX\_ID\_SYNOPSIS, [119](#)
- VX\_ID\_TI, [119](#)
- VX\_ID\_VIDEANTIS, [119](#)
- VX\_ID\_VIVANTE, [119](#)
- VX\_ID\_XILINX, [119](#)
- VX\_INTERPOLATION\_TYPE\_AREA, [119](#)
- VX\_INTERPOLATION\_TYPE\_BILINEAR, [119](#)
- VX\_INTERPOLATION\_TYPE\_NEAREST\_NEIGHBOR, [119](#)
- VX\_STATUS\_MIN, [115](#)
- VX\_SUCCESS, [116](#)
- VX\_TYPE\_ARRAY, [115](#)
- VX\_TYPE\_BOOL, [114](#)
- VX\_TYPE\_CHAR, [114](#)
- VX\_TYPE\_CONTEXT, [114](#)
- VX\_TYPE\_CONVOLUTION, [115](#)
- VX\_TYPE\_COORDINATES2D, [114](#)
- VX\_TYPE\_COORDINATES3D, [114](#)
- VX\_TYPE\_DELAY, [115](#)
- VX\_TYPE\_DF\_IMAGE, [114](#)
- VX\_TYPE\_DISTRIBUTION, [115](#)
- VX\_TYPE\_ENUM, [114](#)
- VX\_TYPE\_ERROR, [115](#)
- VX\_TYPE\_FLOAT32, [114](#)
- VX\_TYPE\_FLOAT64, [114](#)
- VX\_TYPE\_GRAPH, [114](#)
- VX\_TYPE\_IMAGE, [115](#)
- VX\_TYPE\_INT16, [114](#)
- VX\_TYPE\_INT32, [114](#)
- VX\_TYPE\_INT64, [114](#)
- VX\_TYPE\_INT8, [114](#)
- VX\_TYPE\_INVALID, [114](#)
- VX\_TYPE\_KERNEL, [115](#)
- VX\_TYPE\_KEYPOINT, [114](#)
- VX\_TYPE\_LUT, [115](#)
- VX\_TYPE\_MATRIX, [115](#)
- VX\_TYPE\_META\_FORMAT, [115](#)
- VX\_TYPE\_NODE, [115](#)
- VX\_TYPE\_OBJECT\_MAX, [115](#)
- VX\_TYPE\_PARAMETER, [115](#)
- VX\_TYPE\_PYRAMID, [115](#)
- VX\_TYPE\_RECTANGLE, [114](#)
- VX\_TYPE\_REFERENCE, [114](#)
- VX\_TYPE\_REMAP, [115](#)
- VX\_TYPE\_SCALAR, [115](#)
- VX\_TYPE\_SCALAR\_MAX, [114](#)
- VX\_TYPE\_SIZE, [114](#)
- VX\_TYPE\_STRUCT\_MAX, [114](#)
- VX\_TYPE\_THRESHOLD, [115](#)
- VX\_TYPE\_UINT16, [114](#)
- VX\_TYPE\_UINT32, [114](#)
- VX\_TYPE\_UINT64, [114](#)
- VX\_TYPE\_UINT8, [114](#)
- VX\_TYPE\_USER\_STRUCT\_END, [114](#)
- VX\_TYPE\_USER\_STRUCT\_START, [114](#)
- VX\_TYPE\_VENDOR\_OBJECT\_END, [115](#)
- VX\_TYPE\_VENDOR\_OBJECT\_START, [115](#)
- VX\_TYPE\_VENDOR\_STRUCT\_END, [114](#)
- VX\_TYPE\_VENDOR\_STRUCT\_START, [114](#)
- vx\_false\_e, [114](#)
- vx\_true\_e, [114](#)
- Box Filter, [45](#)
- Canny Edge Detector, [46](#)
  - VX\_NORM\_L1, [47](#)
  - VX\_NORM\_L2, [47](#)
- Channel Combine, [49](#)
- Channel Extract, [51](#)
- Color Convert, [53](#)
- Convert Bit depth, [57](#)
- Custom Convolution, [59](#)
- Dilate Image, [61](#)
- Equalize Histogram, [62](#)
- Erode Image, [63](#)
- Fast Corners, [64](#)
- Framework: Directives
  - VX\_DIRECTIVE\_DISABLE\_LOGGING, [227](#)
  - VX\_DIRECTIVE\_ENABLE\_LOGGING, [227](#)
- Framework: Hints
  - VX\_HINT\_SERIALIZE, [225](#)
- Framework: Node Callbacks
  - VX\_ACTION\_ABANDON, [220](#)
  - VX\_ACTION\_CONTINUE, [220](#)
- Framework: User Kernels
  - VX\_META\_FORMAT\_ATTRIBUTE\_DELTA\_RECTANGLE, [233](#)
- Gaussian Filter, [67](#)
- Gaussian Image Pyramid, [72](#)
- Harris Corners, [68](#)
- Histogram, [71](#)
- Integral Image, [74](#)
- Magnitude, [76](#)
- Mean and Standard Deviation, [78](#)
- Median Filter, [80](#)
- Min, Max Location, [81](#)
- Node: Border Modes
  - VX\_BORDER\_MODE\_CONSTANT, [196](#)
  - VX\_BORDER\_MODE\_REPLICATE, [196](#)
  - VX\_BORDER\_MODE\_UNDEFINED, [196](#)
- Object: Array
  - VX\_ARRAY\_ATTRIBUTE\_CAPACITY, [141](#)
  - VX\_ARRAY\_ATTRIBUTE\_ITEMSIZE, [141](#)
  - VX\_ARRAY\_ATTRIBUTE\_ITEMTYPE, [141](#)
  - VX\_ARRAY\_ATTRIBUTE\_NUMITEMS, [141](#)
- Object: Context
  - VX\_CONTEXT\_ATTRIBUTE\_CONVOLUTION\_MAXIMUM\_DIMENSION, [126](#)

- VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS, 126
- VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS\_SIZE, 126
- VX\_CONTEXT\_ATTRIBUTE\_IMMEDIATE\_BORDER\_MODE, 126
- VX\_CONTEXT\_ATTRIBUTE\_IMPLEMENTATION, 126
- VX\_CONTEXT\_ATTRIBUTE\_MODULES, 125
- VX\_CONTEXT\_ATTRIBUTE\_OPTICAL\_FLOW\_WINDOW\_MAXIMUM\_DIMENSION, 126
- VX\_CONTEXT\_ATTRIBUTE\_REFERENCES, 126
- VX\_CONTEXT\_ATTRIBUTE\_UNIQUE\_KERNEL\_TABLE, 126
- VX\_CONTEXT\_ATTRIBUTE\_UNIQUE\_KERNELS, 125
- VX\_CONTEXT\_ATTRIBUTE\_VENDOR\_ID, 125
- VX\_CONTEXT\_ATTRIBUTE\_VERSION, 125
- VX\_IMPORT\_TYPE\_HOST, 126
- VX\_IMPORT\_TYPE\_NONE, 126
- VX\_READ\_AND\_WRITE, 127
- VX\_READ\_ONLY, 127
- VX\_ROUND\_POLICY\_TO\_NEAREST\_EVEN, 127
- VX\_ROUND\_POLICY\_TO\_ZERO, 127
- VX\_TERM\_CRITERIA\_BOTH, 127
- VX\_TERM\_CRITERIA\_EPSILON, 127
- VX\_TERM\_CRITERIA\_ITERATIONS, 127
- VX\_WRITE\_ONLY, 127
- Object: Convolution
  - VX\_CONVOLUTION\_ATTRIBUTE\_COLUMNS, 147
  - VX\_CONVOLUTION\_ATTRIBUTE\_ROWS, 147
  - VX\_CONVOLUTION\_ATTRIBUTE\_SCALE, 147
  - VX\_CONVOLUTION\_ATTRIBUTE\_SIZE, 148
- Object: Delay
  - VX\_DELAY\_ATTRIBUTE\_SLOTS, 197
  - VX\_DELAY\_ATTRIBUTE\_TYPE, 197
- Object: Distribution
  - VX\_DISTRIBUTION\_ATTRIBUTE\_BINS, 151
  - VX\_DISTRIBUTION\_ATTRIBUTE\_DIMENSIONS, 150
  - VX\_DISTRIBUTION\_ATTRIBUTE\_OFFSET, 150
  - VX\_DISTRIBUTION\_ATTRIBUTE\_RANGE, 150
  - VX\_DISTRIBUTION\_ATTRIBUTE\_SIZE, 151
  - VX\_DISTRIBUTION\_ATTRIBUTE\_WINDOW, 151
- Object: Graph
  - VX\_GRAPH\_ATTRIBUTE\_NUMNODES, 132
  - VX\_GRAPH\_ATTRIBUTE\_NUMPARAMETERS, 132
  - VX\_GRAPH\_ATTRIBUTE\_PERFORMANCE, 132
  - VX\_GRAPH\_ATTRIBUTE\_STATUS, 132
- Object: Image
  - VX\_CHANNEL\_RANGE\_FULL, 157
  - VX\_CHANNEL\_RANGE\_RESTRICTED, 157
  - VX\_COLOR\_SPACE\_BT601\_525, 157
  - VX\_COLOR\_SPACE\_BT601\_625, 157
  - VX\_COLOR\_SPACE\_BT709, 157
  - VX\_COLOR\_SPACE\_DEFAULT, 157
  - VX\_COLOR\_SPACE\_NONE, 157
  - VX\_IMAGE\_ATTRIBUTE\_FORMAT, 156
  - VX\_IMAGE\_ATTRIBUTE\_HEIGHT, 156
  - VX\_IMAGE\_ATTRIBUTE\_PLANES, 156
  - VX\_IMAGE\_ATTRIBUTE\_RANGE, 157
  - VX\_IMAGE\_ATTRIBUTE\_SIZE, 157
  - VX\_IMAGE\_ATTRIBUTE\_SPACE, 156
  - VX\_IMAGE\_ATTRIBUTE\_WIDTH, 156
- Object: Kernel
  - VX\_KERNEL\_ABSDIFF, 205
  - VX\_KERNEL\_ACCUMULATE, 205
  - VX\_KERNEL\_ACCUMULATE\_SQUARE, 206
  - VX\_KERNEL\_ACCUMULATE\_WEIGHTED, 205
  - VX\_KERNEL\_ADD, 206
  - VX\_KERNEL\_AND, 206
  - VX\_KERNEL\_ATTRIBUTE\_ENUM, 207
  - VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_PTR, 207
  - VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_SIZE, 207
  - VX\_KERNEL\_ATTRIBUTE\_NAME, 207
  - VX\_KERNEL\_ATTRIBUTE\_PARAMETERS, 207
  - VX\_KERNEL\_BOX\_3x3, 205
  - VX\_KERNEL\_CANNY\_EDGE\_DETECTOR, 206
  - VX\_KERNEL\_CHANNEL\_COMBINE, 204
  - VX\_KERNEL\_CHANNEL\_EXTRACT, 204
  - VX\_KERNEL\_COLOR\_CONVERT, 204
  - VX\_KERNEL\_CONVERTDEPTH, 206
  - VX\_KERNEL\_CUSTOM\_CONVOLUTION, 205
  - VX\_KERNEL\_DILATE\_3x3, 205
  - VX\_KERNEL\_EQUALIZE\_HISTOGRAM, 205
  - VX\_KERNEL\_ERODE\_3x3, 205
  - VX\_KERNEL\_FAST\_CORNERS, 207
  - VX\_KERNEL\_GAUSSIAN\_3x3, 205
  - VX\_KERNEL\_GAUSSIAN\_PYRAMID, 205
  - VX\_KERNEL\_HALFSCALE\_GAUSSIAN, 207
  - VX\_KERNEL\_HARRIS\_CORNERS, 206
  - VX\_KERNEL\_HISTOGRAM, 204
  - VX\_KERNEL\_INTEGRAL\_IMAGE, 205
  - VX\_KERNEL\_INVALID, 204
  - VX\_KERNEL\_MAGNITUDE, 204
  - VX\_KERNEL\_MEAN\_STDDEV, 205
  - VX\_KERNEL\_MEDIAN\_3x3, 205
  - VX\_KERNEL\_MINMAXLOC, 206
  - VX\_KERNEL\_MULTIPLY, 206
  - VX\_KERNEL\_NOT, 206
  - VX\_KERNEL\_OPTICAL\_FLOW\_PYR\_LK, 207
  - VX\_KERNEL\_OR, 206
  - VX\_KERNEL\_PHASE, 204
  - VX\_KERNEL\_REMAP, 207
  - VX\_KERNEL\_SCALE\_IMAGE, 204
  - VX\_KERNEL\_SOBEL\_3x3, 204
  - VX\_KERNEL\_SUBTRACT, 206
  - VX\_KERNEL\_TABLE\_LOOKUP, 204
  - VX\_KERNEL\_THRESHOLD, 205
  - VX\_KERNEL\_WARP\_AFFINE, 206
  - VX\_KERNEL\_WARP\_PERSPECTIVE, 206
  - VX\_KERNEL\_XOR, 206
- Object: LUT



- VX\_LUT\_ATTRIBUTE\_COUNT, [171](#)
- VX\_LUT\_ATTRIBUTE\_SIZE, [171](#)
- VX\_LUT\_ATTRIBUTE\_TYPE, [171](#)
- Object: Matrix
  - VX\_MATRIX\_ATTRIBUTE\_COLUMNS, [175](#)
  - VX\_MATRIX\_ATTRIBUTE\_ROWS, [175](#)
  - VX\_MATRIX\_ATTRIBUTE\_SIZE, [175](#)
  - VX\_MATRIX\_ATTRIBUTE\_TYPE, [175](#)
- Object: Node
  - VX\_NODE\_ATTRIBUTE\_BORDER\_MODE, [138](#)
  - VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_PTR, [138](#)
  - VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_SIZE, [138](#)
  - VX\_NODE\_ATTRIBUTE\_PERFORMANCE, [138](#)
  - VX\_NODE\_ATTRIBUTE\_STATUS, [138](#)
- Object: Parameter
  - VX\_BIDIRECTIONAL, [212](#)
  - VX\_INPUT, [212](#)
  - VX\_OUTPUT, [212](#)
  - VX\_PARAMETER\_ATTRIBUTE\_DIRECTION, [212](#)
  - VX\_PARAMETER\_ATTRIBUTE\_INDEX, [212](#)
  - VX\_PARAMETER\_ATTRIBUTE\_REF, [212](#)
  - VX\_PARAMETER\_ATTRIBUTE\_STATE, [212](#)
  - VX\_PARAMETER\_ATTRIBUTE\_TYPE, [212](#)
  - VX\_PARAMETER\_STATE\_OPTIONAL, [212](#)
  - VX\_PARAMETER\_STATE\_REQUIRED, [212](#)
- Object: Pyramid
  - VX\_PYRAMID\_ATTRIBUTE\_FORMAT, [179](#)
  - VX\_PYRAMID\_ATTRIBUTE\_HEIGHT, [179](#)
  - VX\_PYRAMID\_ATTRIBUTE\_LEVELS, [179](#)
  - VX\_PYRAMID\_ATTRIBUTE\_SCALE, [179](#)
  - VX\_PYRAMID\_ATTRIBUTE\_WIDTH, [179](#)
- Object: Reference
  - VX\_REF\_ATTRIBUTE\_COUNT, [122](#)
  - VX\_REF\_ATTRIBUTE\_TYPE, [122](#)
- Object: Remap
  - VX\_REMAP\_ATTRIBUTE\_DESTINATION\_HEIGHT, [182](#)
  - VX\_REMAP\_ATTRIBUTE\_DESTINATION\_WIDTH, [182](#)
  - VX\_REMAP\_ATTRIBUTE\_SOURCE\_HEIGHT, [182](#)
  - VX\_REMAP\_ATTRIBUTE\_SOURCE\_WIDTH, [182](#)
- Object: Scalar
  - VX\_SCALAR\_ATTRIBUTE\_TYPE, [186](#)
- Object: Threshold
  - VX\_THRESHOLD\_ATTRIBUTE\_DATA\_TYPE, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_FALSE\_VALUE, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_LOWER, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_UPPER, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_VALUE, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_TRUE\_VALUE, [190](#)
  - VX\_THRESHOLD\_ATTRIBUTE\_TYPE, [190](#)
  - VX\_THRESHOLD\_TYPE\_BINARY, [189](#)
  - VX\_THRESHOLD\_TYPE\_RANGE, [189](#)
- Objects, [121](#)
- Phase, [87](#)
- Pixel-wise Multiplication, [89](#)
- Remap, [91](#)
- Scale Image, [93](#)
- Sobel 3x3, [97](#)
- Thresholding, [100](#)
- VX\_ACTION\_ABANDON
  - Framework: Node Callbacks, [220](#)
- VX\_ACTION\_CONTINUE
  - Framework: Node Callbacks, [220](#)
- VX\_ARRAY\_ATTRIBUTE\_CAPACITY
  - Object: Array, [141](#)
- VX\_ARRAY\_ATTRIBUTE\_ITEMSIZE
  - Object: Array, [141](#)
- VX\_ARRAY\_ATTRIBUTE\_ITEMTYPE
  - Object: Array, [141](#)
- VX\_ARRAY\_ATTRIBUTE\_NUMITEMS
  - Object: Array, [141](#)
- VX\_BIDIRECTIONAL
  - Object: Parameter, [212](#)
- VX\_BORDER\_MODE\_CONSTANT
  - Node: Border Modes, [196](#)
- VX\_BORDER\_MODE\_REPLICATE
  - Node: Border Modes, [196](#)
- VX\_BORDER\_MODE\_UNDEFINED
  - Node: Border Modes, [196](#)
- VX\_CHANNEL\_0
  - Basic Features, [118](#)
- VX\_CHANNEL\_1
  - Basic Features, [118](#)
- VX\_CHANNEL\_2
  - Basic Features, [118](#)
- VX\_CHANNEL\_3
  - Basic Features, [118](#)
- VX\_CHANNEL\_A
  - Basic Features, [118](#)
- VX\_CHANNEL\_B
  - Basic Features, [118](#)
- VX\_CHANNEL\_G
  - Basic Features, [118](#)
- VX\_CHANNEL\_R
  - Basic Features, [118](#)
- VX\_CHANNEL\_RANGE\_FULL
  - Object: Image, [157](#)
- VX\_CHANNEL\_RANGE\_RESTRICTED
  - Object: Image, [157](#)
- VX\_CHANNEL\_U
  - Basic Features, [118](#)

- VX\_CHANNEL\_V
  - Basic Features, [118](#)
- VX\_CHANNEL\_Y
  - Basic Features, [118](#)
- VX\_COLOR\_SPACE\_BT601\_525
  - Object: Image, [157](#)
- VX\_COLOR\_SPACE\_BT601\_625
  - Object: Image, [157](#)
- VX\_COLOR\_SPACE\_BT709
  - Object: Image, [157](#)
- VX\_COLOR\_SPACE\_DEFAULT
  - Object: Image, [157](#)
- VX\_COLOR\_SPACE\_NONE
  - Object: Image, [157](#)
- VX\_CONTEXT\_ATTRIBUTE\_CONVOLUTION\_MAXIMUM\_DIMENSION
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_EXTENSIONS\_SIZE
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_IMMEDIATE\_BORDER\_MODE
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_IMPLEMENTATION
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_MODULES
  - Object: Context, [125](#)
- VX\_CONTEXT\_ATTRIBUTE\_OPTICAL\_FLOW\_WINDOW\_MAXIMUM\_DIMENSION
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_REFERENCES
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_UNIQUE\_KERNEL\_TABLE
  - Object: Context, [126](#)
- VX\_CONTEXT\_ATTRIBUTE\_UNIQUE\_KERNELS
  - Object: Context, [125](#)
- VX\_CONTEXT\_ATTRIBUTE\_VENDOR\_ID
  - Object: Context, [125](#)
- VX\_CONTEXT\_ATTRIBUTE\_VERSION
  - Object: Context, [125](#)
- VX\_CONVERT\_POLICY\_SATURATE
  - Basic Features, [117](#)
- VX\_CONVERT\_POLICY\_WRAP
  - Basic Features, [117](#)
- VX\_CONVOLUTION\_ATTRIBUTE\_COLUMNS
  - Object: Convolution, [147](#)
- VX\_CONVOLUTION\_ATTRIBUTE\_ROWS
  - Object: Convolution, [147](#)
- VX\_CONVOLUTION\_ATTRIBUTE\_SCALE
  - Object: Convolution, [147](#)
- VX\_CONVOLUTION\_ATTRIBUTE\_SIZE
  - Object: Convolution, [148](#)
- VX\_DELAY\_ATTRIBUTE\_SLOTS
  - Object: Delay, [197](#)
- VX\_DELAY\_ATTRIBUTE\_TYPE
  - Object: Delay, [197](#)
- VX\_DF\_IMAGE\_IYUV
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_NV12
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_NV21
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_RGB
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_RGBX
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_S16
  - Basic Features, [118](#)
- VX\_DF\_IMAGE\_S32
  - Basic Features, [118](#)
- VX\_DF\_IMAGE\_U16
  - Basic Features, [118](#)
- VX\_DF\_IMAGE\_U32
  - Basic Features, [118](#)
- VX\_DF\_IMAGE\_U8
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_UYVY
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_VIRT
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_YUV4
  - Basic Features, [117](#)
- VX\_DF\_IMAGE\_YUYV
  - Basic Features, [117](#)
- VX\_DIRECTIVE\_DISABLE\_LOGGING
  - Framework: Directives, [227](#)
- VX\_DIRECTIVE\_ENABLE\_LOGGING
  - Framework: Directives, [227](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_BINS
  - Object: Distribution, [151](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_DIMENSIONS
  - Object: Distribution, [150](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_OFFSET
  - Object: Distribution, [150](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_RANGE
  - Object: Distribution, [150](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_SIZE
  - Object: Distribution, [151](#)
- VX\_DISTRIBUTION\_ATTRIBUTE\_WINDOW
  - Object: Distribution, [151](#)
- VX\_ENUM\_ACCESSOR
  - Basic Features, [117](#)
- VX\_ENUM\_ACTION
  - Basic Features, [116](#)
- VX\_ENUM\_BORDER\_MODE
  - Basic Features, [117](#)
- VX\_ENUM\_CHANNEL
  - Basic Features, [117](#)
- VX\_ENUM\_COLOR\_RANGE
  - Basic Features, [117](#)
- VX\_ENUM\_COLOR\_SPACE
  - Basic Features, [116](#)
- VX\_ENUM\_COMPARISON
  - Basic Features, [117](#)

- VX\_ENUM\_CONVERT\_POLICY
  - Basic Features, [117](#)
- VX\_ENUM\_DIRECTION
  - Basic Features, [116](#)
- VX\_ENUM\_DIRECTIVE
  - Basic Features, [116](#)
- VX\_ENUM\_HINT
  - Basic Features, [116](#)
- VX\_ENUM\_IMPORT\_MEM
  - Basic Features, [117](#)
- VX\_ENUM\_INTERPOLATION
  - Basic Features, [116](#)
- VX\_ENUM\_NORM\_TYPE
  - Basic Features, [117](#)
- VX\_ENUM\_OVERFLOW
  - Basic Features, [116](#)
- VX\_ENUM\_PARAMETER\_STATE
  - Basic Features, [117](#)
- VX\_ENUM\_ROUND\_POLICY
  - Basic Features, [117](#)
- VX\_ENUM\_TERM\_CRITERIA
  - Basic Features, [117](#)
- VX\_ENUM\_THRESHOLD\_TYPE
  - Basic Features, [117](#)
- VX\_ERROR\_GRAPH\_ABANDONED
  - Basic Features, [115](#)
- VX\_ERROR\_GRAPH\_SCHEDULED
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_DIMENSION
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_FORMAT
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_GRAPH
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_LINK
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_MODULE
  - Basic Features, [116](#)
- VX\_ERROR\_INVALID\_NODE
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_PARAMETERS
  - Basic Features, [116](#)
- VX\_ERROR\_INVALID\_REFERENCE
  - Basic Features, [116](#)
- VX\_ERROR\_INVALID\_SCOPE
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_TYPE
  - Basic Features, [115](#)
- VX\_ERROR\_INVALID\_VALUE
  - Basic Features, [115](#)
- VX\_ERROR\_MULTIPLE\_WRITERS
  - Basic Features, [115](#)
- VX\_ERROR\_NO\_MEMORY
  - Basic Features, [116](#)
- VX\_ERROR\_NO\_RESOURCES
  - Basic Features, [116](#)
- VX\_ERROR\_NOT\_ALLOCATED
  - Basic Features, [116](#)
- VX\_ERROR\_NOT\_COMPATIBLE
  - Basic Features, [116](#)
- VX\_ERROR\_NOT\_IMPLEMENTED
  - Basic Features, [116](#)
- VX\_ERROR\_NOT\_SUFFICIENT
  - Basic Features, [116](#)
- VX\_ERROR\_NOT\_SUPPORTED
  - Basic Features, [116](#)
- VX\_ERROR\_OPTIMIZED\_AWAY
  - Basic Features, [116](#)
- VX\_ERROR\_REFERENCE\_NONZERO
  - Basic Features, [115](#)
- VX\_FAILURE
  - Basic Features, [116](#)
- VX\_GRAPH\_ATTRIBUTE\_NUMNODES
  - Object: Graph, [132](#)
- VX\_GRAPH\_ATTRIBUTE\_NUMPARAMETERS
  - Object: Graph, [132](#)
- VX\_GRAPH\_ATTRIBUTE\_PERFORMANCE
  - Object: Graph, [132](#)
- VX\_GRAPH\_ATTRIBUTE\_STATUS
  - Object: Graph, [132](#)
- VX\_HINT\_SERIALIZE
  - Framework: Hints, [225](#)
- VX\_ID\_AMD
  - Basic Features, [119](#)
- VX\_ID\_ARM
  - Basic Features, [119](#)
- VX\_ID\_AXIS
  - Basic Features, [119](#)
- VX\_ID\_BDTI
  - Basic Features, [119](#)
- VX\_ID\_BROADCOM
  - Basic Features, [119](#)
- VX\_ID\_CEVA
  - Basic Features, [119](#)
- VX\_ID\_COGNIVUE
  - Basic Features, [119](#)
- VX\_ID\_DEFAULT
  - Basic Features, [119](#)
- VX\_ID\_FREESCALE
  - Basic Features, [119](#)
- VX\_ID\_IMAGINATION
  - Basic Features, [119](#)
- VX\_ID\_INTEL
  - Basic Features, [119](#)
- VX\_ID\_ITSEEZ
  - Basic Features, [119](#)
- VX\_ID\_KHRONOS
  - Basic Features, [119](#)
- VX\_ID\_MARVELL
  - Basic Features, [119](#)
- VX\_ID\_MEDIATEK
  - Basic Features, [119](#)
- VX\_ID\_MOVIDIUS
  - Basic Features, [119](#)
- VX\_ID\_NVIDIA
  - Basic Features, [119](#)

- VX\_ID\_QUALCOMM
  - Basic Features, [119](#)
- VX\_ID\_RENESAS
  - Basic Features, [119](#)
- VX\_ID\_SAMSUNG
  - Basic Features, [119](#)
- VX\_ID\_ST
  - Basic Features, [119](#)
- VX\_ID\_SYNOPSYS
  - Basic Features, [119](#)
- VX\_ID\_TI
  - Basic Features, [119](#)
- VX\_ID\_VIDEANTIS
  - Basic Features, [119](#)
- VX\_ID\_VIVANTE
  - Basic Features, [119](#)
- VX\_ID\_XILINX
  - Basic Features, [119](#)
- VX\_IMAGE\_ATTRIBUTE\_FORMAT
  - Object: Image, [156](#)
- VX\_IMAGE\_ATTRIBUTE\_HEIGHT
  - Object: Image, [156](#)
- VX\_IMAGE\_ATTRIBUTE\_PLANES
  - Object: Image, [156](#)
- VX\_IMAGE\_ATTRIBUTE\_RANGE
  - Object: Image, [157](#)
- VX\_IMAGE\_ATTRIBUTE\_SIZE
  - Object: Image, [157](#)
- VX\_IMAGE\_ATTRIBUTE\_SPACE
  - Object: Image, [156](#)
- VX\_IMAGE\_ATTRIBUTE\_WIDTH
  - Object: Image, [156](#)
- VX\_IMPORT\_TYPE\_HOST
  - Object: Context, [126](#)
- VX\_IMPORT\_TYPE\_NONE
  - Object: Context, [126](#)
- VX\_INPUT
  - Object: Parameter, [212](#)
- VX\_INTERPOLATION\_TYPE\_AREA
  - Basic Features, [119](#)
- VX\_INTERPOLATION\_TYPE\_BILINEAR
  - Basic Features, [119](#)
- VX\_INTERPOLATION\_TYPE\_NEAREST\_NEIGHBOR
  - Basic Features, [119](#)
- VX\_KERNEL\_ABSDIFF
  - Object: Kernel, [205](#)
- VX\_KERNEL\_ACCUMULATE
  - Object: Kernel, [205](#)
- VX\_KERNEL\_ACCUMULATE\_SQUARE
  - Object: Kernel, [206](#)
- VX\_KERNEL\_ACCUMULATE\_WEIGHTED
  - Object: Kernel, [205](#)
- VX\_KERNEL\_ADD
  - Object: Kernel, [206](#)
- VX\_KERNEL\_AND
  - Object: Kernel, [206](#)
- VX\_KERNEL\_ATTRIBUTE\_ENUM
  - Object: Kernel, [207](#)
- VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_PTR
  - Object: Kernel, [207](#)
- VX\_KERNEL\_ATTRIBUTE\_LOCAL\_DATA\_SIZE
  - Object: Kernel, [207](#)
- VX\_KERNEL\_ATTRIBUTE\_NAME
  - Object: Kernel, [207](#)
- VX\_KERNEL\_ATTRIBUTE\_PARAMETERS
  - Object: Kernel, [207](#)
- VX\_KERNEL\_BOX\_3x3
  - Object: Kernel, [205](#)
- VX\_KERNEL\_CANNY\_EDGE\_DETECTOR
  - Object: Kernel, [206](#)
- VX\_KERNEL\_CHANNEL\_COMBINE
  - Object: Kernel, [204](#)
- VX\_KERNEL\_CHANNEL\_EXTRACT
  - Object: Kernel, [204](#)
- VX\_KERNEL\_COLOR\_CONVERT
  - Object: Kernel, [204](#)
- VX\_KERNEL\_CONVERTDEPTH
  - Object: Kernel, [206](#)
- VX\_KERNEL\_CUSTOM\_CONVOLUTION
  - Object: Kernel, [205](#)
- VX\_KERNEL\_DILATE\_3x3
  - Object: Kernel, [205](#)
- VX\_KERNEL\_EQUALIZE\_HISTOGRAM
  - Object: Kernel, [205](#)
- VX\_KERNEL\_ERODE\_3x3
  - Object: Kernel, [205](#)
- VX\_KERNEL\_FAST\_CORNERS
  - Object: Kernel, [207](#)
- VX\_KERNEL\_GAUSSIAN\_3x3
  - Object: Kernel, [205](#)
- VX\_KERNEL\_GAUSSIAN\_PYRAMID
  - Object: Kernel, [205](#)
- VX\_KERNEL\_HALFSCALE\_GAUSSIAN
  - Object: Kernel, [207](#)
- VX\_KERNEL\_HARRIS\_CORNERS
  - Object: Kernel, [206](#)
- VX\_KERNEL\_HISTOGRAM
  - Object: Kernel, [204](#)
- VX\_KERNEL\_INTEGRAL\_IMAGE
  - Object: Kernel, [205](#)
- VX\_KERNEL\_INVALID
  - Object: Kernel, [204](#)
- VX\_KERNEL\_MAGNITUDE
  - Object: Kernel, [204](#)
- VX\_KERNEL\_MEAN\_STDDEV
  - Object: Kernel, [205](#)
- VX\_KERNEL\_MEDIAN\_3x3
  - Object: Kernel, [205](#)
- VX\_KERNEL\_MINMAXLOC
  - Object: Kernel, [206](#)
- VX\_KERNEL\_MULTIPLY
  - Object: Kernel, [206](#)
- VX\_KERNEL\_NOT
  - Object: Kernel, [206](#)
- VX\_KERNEL\_OPTICAL\_FLOW\_PYR\_LK
  - Object: Kernel, [207](#)

- VX\_KERNEL\_OR
  - Object: Kernel, [206](#)
- VX\_KERNEL\_PHASE
  - Object: Kernel, [204](#)
- VX\_KERNEL\_REMAP
  - Object: Kernel, [207](#)
- VX\_KERNEL\_SCALE\_IMAGE
  - Object: Kernel, [204](#)
- VX\_KERNEL\_SOBEL\_3x3
  - Object: Kernel, [204](#)
- VX\_KERNEL\_SUBTRACT
  - Object: Kernel, [206](#)
- VX\_KERNEL\_TABLE\_LOOKUP
  - Object: Kernel, [204](#)
- VX\_KERNEL\_THRESHOLD
  - Object: Kernel, [205](#)
- VX\_KERNEL\_WARP\_AFFINE
  - Object: Kernel, [206](#)
- VX\_KERNEL\_WARP\_PERSPECTIVE
  - Object: Kernel, [206](#)
- VX\_KERNEL\_XOR
  - Object: Kernel, [206](#)
- VX\_LUT\_ATTRIBUTE\_COUNT
  - Object: LUT, [171](#)
- VX\_LUT\_ATTRIBUTE\_SIZE
  - Object: LUT, [171](#)
- VX\_LUT\_ATTRIBUTE\_TYPE
  - Object: LUT, [171](#)
- VX\_MATRIX\_ATTRIBUTE\_COLUMNS
  - Object: Matrix, [175](#)
- VX\_MATRIX\_ATTRIBUTE\_ROWS
  - Object: Matrix, [175](#)
- VX\_MATRIX\_ATTRIBUTE\_SIZE
  - Object: Matrix, [175](#)
- VX\_MATRIX\_ATTRIBUTE\_TYPE
  - Object: Matrix, [175](#)
- VX\_META\_FORMAT\_ATTRIBUTE\_DELTA\_RECTANGLE
  - Framework: User Kernels, [233](#)
- VX\_NODE\_ATTRIBUTE\_BORDER\_MODE
  - Object: Node, [138](#)
- VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_PTR
  - Object: Node, [138](#)
- VX\_NODE\_ATTRIBUTE\_LOCAL\_DATA\_SIZE
  - Object: Node, [138](#)
- VX\_NODE\_ATTRIBUTE\_PERFORMANCE
  - Object: Node, [138](#)
- VX\_NODE\_ATTRIBUTE\_STATUS
  - Object: Node, [138](#)
- VX\_NORM\_L1
  - Canny Edge Detector, [47](#)
- VX\_NORM\_L2
  - Canny Edge Detector, [47](#)
- VX\_OUTPUT
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_ATTRIBUTE\_DIRECTION
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_ATTRIBUTE\_INDEX
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_ATTRIBUTE\_REF
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_ATTRIBUTE\_STATE
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_ATTRIBUTE\_TYPE
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_STATE\_OPTIONAL
  - Object: Parameter, [212](#)
- VX\_PARAMETER\_STATE\_REQUIRED
  - Object: Parameter, [212](#)
- VX\_PYRAMID\_ATTRIBUTE\_FORMAT
  - Object: Pyramid, [179](#)
- VX\_PYRAMID\_ATTRIBUTE\_HEIGHT
  - Object: Pyramid, [179](#)
- VX\_PYRAMID\_ATTRIBUTE\_LEVELS
  - Object: Pyramid, [179](#)
- VX\_PYRAMID\_ATTRIBUTE\_SCALE
  - Object: Pyramid, [179](#)
- VX\_PYRAMID\_ATTRIBUTE\_WIDTH
  - Object: Pyramid, [179](#)
- VX\_READ\_AND\_WRITE
  - Object: Context, [127](#)
- VX\_READ\_ONLY
  - Object: Context, [127](#)
- VX\_REF\_ATTRIBUTE\_COUNT
  - Object: Reference, [122](#)
- VX\_REF\_ATTRIBUTE\_TYPE
  - Object: Reference, [122](#)
- VX\_REMAP\_ATTRIBUTE\_DESTINATION\_HEIGHT
  - Object: Remap, [182](#)
- VX\_REMAP\_ATTRIBUTE\_DESTINATION\_WIDTH
  - Object: Remap, [182](#)
- VX\_REMAP\_ATTRIBUTE\_SOURCE\_HEIGHT
  - Object: Remap, [182](#)
- VX\_REMAP\_ATTRIBUTE\_SOURCE\_WIDTH
  - Object: Remap, [182](#)
- VX\_ROUND\_POLICY\_TO\_NEAREST\_EVEN
  - Object: Context, [127](#)
- VX\_ROUND\_POLICY\_TO\_ZERO
  - Object: Context, [127](#)
- VX\_SCALAR\_ATTRIBUTE\_TYPE
  - Object: Scalar, [186](#)
- VX\_STATUS\_MIN
  - Basic Features, [115](#)
- VX\_SUCCESS
  - Basic Features, [116](#)
- VX\_TERM\_CRITERIA\_BOTH
  - Object: Context, [127](#)
- VX\_TERM\_CRITERIA\_EPSILON
  - Object: Context, [127](#)
- VX\_TERM\_CRITERIA\_ITERATIONS
  - Object: Context, [127](#)
- VX\_THRESHOLD\_ATTRIBUTE\_DATA\_TYPE
  - Object: Threshold, [190](#)
- VX\_THRESHOLD\_ATTRIBUTE\_FALSE\_VALUE
  - Object: Threshold, [190](#)

- VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_LOW↔  
ER  
Object: Threshold, [190](#)
- VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_UPP↔  
ER  
Object: Threshold, [190](#)
- VX\_THRESHOLD\_ATTRIBUTE\_THRESHOLD\_VAL↔  
UE  
Object: Threshold, [190](#)
- VX\_THRESHOLD\_ATTRIBUTE\_TRUE\_VALUE  
Object: Threshold, [190](#)
- VX\_THRESHOLD\_ATTRIBUTE\_TYPE  
Object: Threshold, [190](#)
- VX\_THRESHOLD\_TYPE\_BINARY  
Object: Threshold, [189](#)
- VX\_THRESHOLD\_TYPE\_RANGE  
Object: Threshold, [189](#)
- VX\_TYPE\_ARRAY  
Basic Features, [115](#)
- VX\_TYPE\_BOOL  
Basic Features, [114](#)
- VX\_TYPE\_CHAR  
Basic Features, [114](#)
- VX\_TYPE\_CONTEXT  
Basic Features, [114](#)
- VX\_TYPE\_CONVOLUTION  
Basic Features, [115](#)
- VX\_TYPE\_COORDINATES2D  
Basic Features, [114](#)
- VX\_TYPE\_COORDINATES3D  
Basic Features, [114](#)
- VX\_TYPE\_DELAY  
Basic Features, [115](#)
- VX\_TYPE\_DF\_IMAGE  
Basic Features, [114](#)
- VX\_TYPE\_DISTRIBUTION  
Basic Features, [115](#)
- VX\_TYPE\_ENUM  
Basic Features, [114](#)
- VX\_TYPE\_ERROR  
Basic Features, [115](#)
- VX\_TYPE\_FLOAT32  
Basic Features, [114](#)
- VX\_TYPE\_FLOAT64  
Basic Features, [114](#)
- VX\_TYPE\_GRAPH  
Basic Features, [114](#)
- VX\_TYPE\_IMAGE  
Basic Features, [115](#)
- VX\_TYPE\_INT16  
Basic Features, [114](#)
- VX\_TYPE\_INT32  
Basic Features, [114](#)
- VX\_TYPE\_INT64  
Basic Features, [114](#)
- VX\_TYPE\_INT8  
Basic Features, [114](#)
- VX\_TYPE\_INVALID  
Basic Features, [114](#)
- VX\_TYPE\_KERNEL  
Basic Features, [115](#)
- VX\_TYPE\_KEYPOINT  
Basic Features, [114](#)
- VX\_TYPE\_LUT  
Basic Features, [115](#)
- VX\_TYPE\_MATRIX  
Basic Features, [115](#)
- VX\_TYPE\_META\_FORMAT  
Basic Features, [115](#)
- VX\_TYPE\_NODE  
Basic Features, [115](#)
- VX\_TYPE\_OBJECT\_MAX  
Basic Features, [115](#)
- VX\_TYPE\_PARAMETER  
Basic Features, [115](#)
- VX\_TYPE\_PYRAMID  
Basic Features, [115](#)
- VX\_TYPE\_RECTANGLE  
Basic Features, [114](#)
- VX\_TYPE\_REFERENCE  
Basic Features, [114](#)
- VX\_TYPE\_REMAP  
Basic Features, [115](#)
- VX\_TYPE\_SCALAR  
Basic Features, [115](#)
- VX\_TYPE\_SCALAR\_MAX  
Basic Features, [114](#)
- VX\_TYPE\_SIZE  
Basic Features, [114](#)
- VX\_TYPE\_STRUCT\_MAX  
Basic Features, [114](#)
- VX\_TYPE\_THRESHOLD  
Basic Features, [115](#)
- VX\_TYPE\_UINT16  
Basic Features, [114](#)
- VX\_TYPE\_UINT32  
Basic Features, [114](#)
- VX\_TYPE\_UINT64  
Basic Features, [114](#)
- VX\_TYPE\_UINT8  
Basic Features, [114](#)
- VX\_TYPE\_USER\_STRUCT\_END  
Basic Features, [114](#)
- VX\_TYPE\_USER\_STRUCT\_START  
Basic Features, [114](#)
- VX\_TYPE\_VENDOR\_OBJECT\_END  
Basic Features, [115](#)
- VX\_TYPE\_VENDOR\_OBJECT\_START  
Basic Features, [115](#)
- VX\_TYPE\_VENDOR\_STRUCT\_END  
Basic Features, [114](#)
- VX\_TYPE\_VENDOR\_STRUCT\_START  
Basic Features, [114](#)
- VX\_WRITE\_ONLY  
Object: Context, [127](#)
- Vision Functions, [25](#)

- [vx\\_border\\_mode\\_t, 196](#)
- [vx\\_coordinates2d\\_t, 111](#)
- [vx\\_coordinates3d\\_t, 111](#)
- [vx\\_delta\\_rectangle\\_t, 112](#)
- [vx\\_false\\_e](#)
  - Basic Features, [114](#)
- [vx\\_imagepatch\\_addressing\\_t, 154](#)
- [vx\\_kernel\\_info\\_t, 203](#)
- [vx\\_keypoint\\_t, 112](#)
- [vx\\_perf\\_t, 222](#)
- [vx\\_rectangle\\_t, 112](#)
- [vx\\_true\\_e](#)
  - Basic Features, [114](#)
- Warp Affine, [102](#)
- Warp Perspective, [104](#)