OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems, and handheld devices.

Specification documents and online reference are available at www.khronos.org/opencl.

OpenCL

KHRONOS GROUP

[n.n.n] and purple text: sections and text in the OpenCL API 2.2 Spec.
[n.n.n] and green text: sections and text in the OpenCL C++ 2.2 Spec.
[n.n.n] and brown text: sections and text in the OpenCL C 2.0 Spec.
[n.n.n] and blue text: sections and text in the OpenCL Extension 2.2 Spec.

OpenCL API

OpenCL API Reference

Section and table references are to the OpenCL API 2.2 specification.

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. Items in blue apply only when the appropriate extension is enabled (see Extensions on page 21 of this reference guide).

Querying platform info & devices [4.1-2] [9.16.9]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*, cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clIcdGetPlatformIDsKHR** (cl_uint *num_entries*, cl_platform_id * *platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*, cl_platform_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}, CL_PLATFORM_HOST_TIMER_RESOLUTION, CL_PLATFORM_ICD_SUFFIX_KHR [Table 4.1]

cl_int **clGetDeviceIDs** (cl_platform_id *platform*, cl_device_type *device_type*, cl_uint *num_entries*, cl_device_id **devices*, cl_uint **num_devices*)

device_type: [Table 4.2]
CL_DEVICE_TYPE_{ACCELERATOR, ALL, CPU}, CL_DEVICE_TYPE_{CUSTOM, DEFAULT, GPU}

cl_int **clGetDeviceInfo** (cl_device_id *device*, cl_device_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 4.3]
CL_DEVICE_ADDRESS_BITS, CL_DEVICE_AVAILABLE, CL_DEVICE_BUILT_IN_KERNELS, CL_DEVICE_COMPILER_AVAILABLE, CL_DEVICE_{DOUBLE, HALF, SINGLE}_FP_CONFIG, CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_EXTENSIONS, CL_DEVICE_ERROR_CORRECTION_SUPPORT, CL_DEVICE_EXECUTION_CAPABILITIES, CL_DEVICE_GLOBAL_MEM_CACHE_{SIZE, TYPE}, CL_DEVICE_GLOBAL_MEM_{CACHELINE_SIZE, SIZE}, CL_DEVICE_GLOBAL_VARIABLE_PREFERRED_TOTAL_SIZE, CL_DEVICE_IL_VERSION, CL_DEVICE_IMAGE_MAX_{ARRAY, BUFFER}_SIZE, CL_DEVICE_IMAGE_SUPPORT, CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT}, CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}, CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT, CL_DEVICE_IMAGE_PITCH_ALIGNMENT, CL_DEVICE_LINKER_AVAILABLE, CL_DEVICE_LOCAL_MEM_{TYPE, SIZE}, CL_DEVICE_MAX_{CLOCK_FREQUENCY, PIPE_ARGS}, CL_DEVICE_MAX_{COMPUTE_UNITS, SAMPLERS}, CL_DEVICE_MAX_CONSTANT_{ARGS, BUFFER_SIZE}, CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE, CL_DEVICE_MAX_{MEM_ALLOC, PARAMETER}_SIZE, CL_DEVICE_MAX_NUM_SUB_GROUPS, CL_DEVICE_MAX_ON_DEVICE_{QUEUES, EVENTS}, CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS, CL_DEVICE_MAX_READ_WRITE_IMAGE_ARGS, CL_DEVICE_MAX_SUB_GROUPS, CL_DEVICE_MAX_WORK_GROUP_SIZE, CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES}, CL_DEVICE_MEM_BASE_ADDR_ALIGN, CL_DEVICE_NAME, CL_DEVICE_NATIVE_VECTOR_WIDTH_{CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT}, CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT, CL_DEVICE_{OPENCL_C_VERSION, PARENT_DEVICE}, CL_DEVICE_PARTITION_AFFINITY_DOMAIN, CL_DEVICE_PARTITION_MAX_SUB_DEVICES, CL_DEVICE_PARTITION_{PROPERTIES, TYPE}, CL_DEVICE_PIPE_MAX_ACTIVE_RESERVATIONS, CL_DEVICE_PIPE_MAX_PACKET_SIZE, CL_DEVICE_{PLATFORM, PRINTF_BUFFER_SIZE}, CL_DEVICE_PREFERRED_*Y*_ATOMIC_ALIGNMENT (where *Y* may be LOCAL, GLOBAL, PLATFORM), CL_DEVICE_PREFERRED_VECTOR_WIDTH_*Z* (where *Z* may be CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT), CL_DEVICE_PREFERRED_INTEROP_USER_SYNC, CL_DEVICE_PROFILE, CL_DEVICE_PROFILING_TIMER_RESOLUTION, CL_DEVICE_SPIR_VERSIONS, CL_DEVICE_SUB_GROUP_INDEPENDENT_FORWARD_PROGRESS CL_DEVICE_QUEUE_ON_{DEVICE, HOST}_PROPERTIES, CL_DEVICE_QUEUE_ON_DEVICE_MAX_SIZE, CL_DEVICE_QUEUE_ON_DEVICE_PREFERRED_SIZE, CL_DEVICE_{REFERENCE_COUNT, VENDOR_ID}, CL_DEVICE_SVM_CAPABILITIES, CL_DEVICE_TERMINATE_CAPABILITY_KHR, CL_DEVICE_{TYPE, VENDOR}, CL_DEVICE_VENDOR_ID, CL_{DEVICE, DRIVER}_VERSION, CL_DEVICE_MAX_NAMED_BARRIER_COUNT_KHR

cl_int **clGetDeviceAndHostTimer** (cl_device_id *device*, cl_ulong **device_timestamp*, cl_ulong **host_timestamp*)

cl_int **clGetHostTimer** (cl_device_id *device*, cl_ulong **host_timestamp*)

Partitioning a device [4.3]

cl_int **clCreateSubDevices** (cl_device_id *in_device*, const cl_device_partition_property **properties*, cl_uint *num_devices*, cl_device_id **out_devices*, cl_uint **num_devices_ret*)

properties: [Table 4.4] CL_DEVICE_PARTITION_EQUALLY, CL_DEVICE_PARTITION_BY_COUNTS, CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN

cl_int **clRetainDevice** (cl_device_id *device*)

cl_int **clReleaseDevice** (cl_device_id *device*)

Contexts [4.4]

cl_context **clCreateContext** (const cl_context_properties **properties*, cl_uint *num_devices*, const cl_device_id **devices*, void (CL_CALLBACK**pfn_notify*)(const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*), void **user_data*, cl_int **errcode_ret*)

properties: [Table 4.5]
NULL or CL_CONTEXT_PLATFORM, CL_CONTEXT_INTEROP_USER_SYNC, CL_CONTEXT_{D3D10, D3D11}_DEVICE_KHR, CL_CONTEXT_ADAPTER_{D3D9, D3D9EX}_KHR, CL_CONTEXT_ADAPTER_DXVA_KHR, CL_CONTEXT_MEMORY_INITIALIZE_KHR, CL_CONTEXT_TERMINATE_KHR, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (const cl_context_properties **properties*, cl_device_type *device_type*, void (CL_CALLBACK **pfn_notify*)(const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*), void **user_data*, cl_int **errcode_ret*)

properties: See **clCreateContext**

device_type: See **clGetDeviceIDs**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

cl_int **clGetContextInfo** (cl_context *context*, cl_context_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name:
CL_CONTEXT_*X* where *X* may be REFERENCE_COUNT, DEVICES, NUM_DEVICES, PROPERTIES, D3D10_PREFER_SHARED_RESOURCES_KHR, D3D11_PREFER_SHARED_RESOURCES_KHR [Table 4.6]

cl_int **clTerminateContextKHR** (cl_context *context*)

Get CL extension function pointers [9.2]

void* **clGetExtensionFunctionAddressForPlatform** (cl_platform_id *platform*, const char **funcname*)

The OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for __kernel functions in a program and calls that allow you to enqueue commands to a command-queue such as executing a kernel, reading, or writing a memory object.

Command queues [5.1]

cl_command_queue **clCreateCommandQueueWithProperties** (cl_context *context*, cl_device_id *device*, const cl_command_queue_properties **properties*, cl_int **errcode_ret*)

**properties:* Points to a zero-terminated list of properties and their values: [Table 5.1] CL_QUEUE_SIZE, CL_QUEUE_PROPERTIES (bitfield which may be set to an OR of CL_QUEUE_* where * may be: OUT_OF_ORDER_EXEC_MODE_ENABLE, PROFILING_ENABLE, ON_DEVICE[_DEFAULT]), CL_QUEUE_THROTTLE_{HIGH, MED, LOW}_KHR (requires the *cl_khr_throttle_hint* extension), CL_QUEUE_PRIORITY_KHR (bitfield which may be one of CL_QUEUE_PRIORITY_HIGH_KHR, CL_QUEUE_PRIORITY_MED_KHR, CL_QUEUE_PRIORITY_LOW_KHR (requires the *cl_khr_priority_hints* extension))

cl_int **clSetDefaultDeviceCommandQueue** (cl_context *context*, cl_device_id *device*, cl_command_queue *command_queue*)

cl_int **clRetainCommandQueue** (cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (cl_command_queue *command_queue*, cl_command_queue_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.2]
CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE[_DEFAULT], CL_QUEUE_SIZE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

OpenCL API

Buffer Objects

Elements of buffer objects are stored sequentially and accessed using a pointer by a kernel executing on a device.

Create buffer objects [5.2.1]

cl_mem **clCreateBuffer** (
cl_context *context*, cl_mem_flags *flags*, size_t *size*,
void *host_ptr, cl_int *errcode_ret)

flags: [Table 5.3] CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_HOST_NO_ACCESS, CL_MEM_HOST_{READ, WRITE}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateSubBuffer** (
cl_mem *buffer*, cl_mem_flags *flags*, cl_buffer_create_type *buffer_create_type*,
const void *buffer_create_info, cl_int *errcode_ret)

flags: See **clCreateBuffer**

buffer_create_type: CL_BUFFER_CREATE_TYPE_REGION

Read, write, copy, & fill buffer objects [5.2.2-3]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*,
size_t *offset*, size_t *size*, void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*,
const size_t *buffer_origin, const size_t *host_origin, const size_t *region,
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*,
size_t *host_slice_pitch*, void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*,
size_t *offset*, size_t *size*, const void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*,
const size_t *buffer_origin, const size_t *host_origin, const size_t *region,
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*,
size_t *host_slice_pitch*, const void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueFillBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, const void *pattern,
size_t *pattern_size*, size_t *offset*, size_t *size*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBuffer** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*,
size_t *src_offset*, size_t *dst_offset*, size_t *size*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBufferRect** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*,
const size_t *src_origin, const size_t *dst_origin, const size_t *region,
size_t *src_row_pitch*, size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

Map buffer objects [5.2.4]

void * **clEnqueueMapBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_map*,
cl_map_flags *map_flags*, size_t *offset*, size_t *size*,
cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list, cl_event *event,
cl_int *errcode_ret)

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

Image Formats [5.3.1.1]

Supported combinations of image_channel_order and image_channel_data_type.

Built-in support [Table 5.8]

CL_R (read or write): CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}
CL_DEPTH (read or write): CL_FLOAT, CL_UNORM_INT16
CL_DEPTH_STENCIL (read only): CL_FLOAT, CL_UNORM_INT24 (Requires the extension *cl_khr_gl_depth_images*)
CL_RG (read or write): CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}
CL_RGBA (read or write): CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_UNORM_INT_101010_2, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}
CL_BGRA (read or write): CL_UNORM_INT8
CL_sRGBA (read only): CL_UNORM_INT8 (Requires the extension *cl_khr_srgb_image_writes*)

Supported image channel order values [Table 5.6]

CL_R, CL_A (read and write): CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_INTENSITY: CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8|16}
CL_DEPTH_STENCIL: Only used if extension *cl_khr_gl_depth_images* is enabled and channel data type = CL_UNORM_INT24 or CL_FLOAT
CL_LUMINANCE: CL_UNORM_INT{8,16}, CL_[HALF_]FLOAT, CL_SNORM_INT{8,16}
CL_RG, CL_RA: CL_[HALF_]FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16, 32} , CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_RGB: CL_UNORM_SHORT_{555,565} , CL_UNORM_INT_101010
CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8
CL_BGRA: CL_{SIGNED, UNSIGNED}_INT8, CL_SNORM_INT8

Image Objects

Items in blue apply when the appropriate extension is enabled.

Create image objects [5.3.1]

cl_mem **clCreateImage** (
cl_context *context*, cl_mem_flags *flags*, const cl_image_format *image_format,
const cl_image_desc *image_desc, void *host_ptr, cl_int *errcode_ret)

flags: See **clCreateBuffer**

Query list of supported image formats [5.3.2]

cl_int **clGetSupportedImageFormats** (
cl_context *context*, cl_mem_flags *flags*, cl_mem_object_type *image_type*,
cl_uint *num_entries*, cl_image_format *image_formats,
cl_uint *num_image_formats)

flags: See **clCreateBuffer**

image_type: CL_MEM_OBJECT_IMAGE{1D, 2D, 3D}, CL_MEM_OBJECT_IMAGE1D_BUFFER, CL_MEM_OBJECT_IMAGE{1D, 2D}_ARRAY

Read, write, copy, & fill image objects [5.3.3-4]

cl_int **clEnqueueReadImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_read*,
const size_t *origin, const size_t *region, size_t *row_pitch*, size_t *slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list*,
cl_event *event)

cl_int **clEnqueueWriteImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_write*,
const size_t *origin, const size_t *region, size_t *input_row_pitch*,
size_t *input_slice_pitch*, const void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueFillImage** (
cl_command_queue *command_queue*, cl_mem *image*, const void *fill_color*,
const size_t *origin, const size_t *region,cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyImage** (
cl_command_queue *command_queue*, cl_mem *src_image*, cl_mem *dst_image*,
const size_t *src_origin, const size_t *dst_origin, const size_t *region,
cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list*, cl_event *event)

Copy between image & buffer objects [5.3.5]

cl_int **clEnqueueCopyImageToBuffer** (
cl_command_queue *command_queue*, cl_mem *src_image*, cl_mem *dst_buffer*,
const size_t *src_origin, const size_t *region, size_t *dst_offset*,
cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list*, cl_event *event)

cl_int **clEnqueueCopyBufferToImage** (
cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_image*,
size_t *src_offset*, const size_t *dst_origin, const size_t *region,
cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list*, cl_event *event)

Map and unmap image objects [5.3.6]

void * **clEnqueueMapImage** (
cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_map*,
cl_map_flags *map_flags*, const size_t *origin, const size_t *region,
size_t *image_row_pitch*, size_t *image_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event, cl_int *errcode_ret)

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

Query image objects [5.3.7]

cl_int **clGetImageInfo** (
cl_mem *image*, cl_image_info *param_name*, size_t *param_value_size*,
void *param_value, size_t *param_value_size_ret)

param_name: [Table 5.10] CL_IMAGE_FORMAT, CL_IMAGE_{ARRAY, ELEMENT}_SIZE, CL_IMAGE_{ROW, SLICE}_PITCH, CL_IMAGE_{HEIGHT, WIDTH, DEPTH}, CL_IMAGE_NUM_{SAMPLES, MIP_LEVELS}, CL_IMAGE_DX9_MEDIA_PLANE_KHR, CL_IMAGE_{D3D10, D3D11}_SUBRESOURCE_KHR

Pipes

A pipe is a memory object that stores data organized as a FIFO. Pipe objects can only be accessed using built-in functions that read from and write to a pipe. Pipe objects are not accessible from the host.

Create pipe objects [5.4.1]

cl_mem **clCreatePipe** (cl_context *context*,
cl_mem_flags *flags*, cl_uint *pipe_packet_size*,
cl_uint *pipe_max_packets*,
const cl_pipe_properties *properties*,
cl_int *errcode_ret*)

flags: 0 or CL_MEM_{READ, WRITE}_ONLY,
CL_MEM_{READ_WRITE, HOST_NO_ACCESS}

Pipe object queries [5.4.2]

cl_int **clGetPipeInfo** (cl_mem *pipe*,
cl_pipe_info *param_name*, size_t *param_value_size*,
void *param_value, size_t *param_value_size_ret)

param_name:
CL_PIPE_PACKET_SIZE, CL_PIPE_MAX_PACKETS

Shared Virtual Memory [5.6]

Shared Virtual Memory (SVM) allows the host and kernels executing on devices to directly share complex, pointer-containing data structures such as trees and linked lists.

SVM sharing granularity

void* **clSVMAlloc** (
cl_context *context*, cl_svm_mem_flags *flags*,
size_t *size*, cl_uint *alignment*)

flags: [Table 5.14]
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_SVM_FINE_GRAIN_BUFFER,
CL_MEM_SVM_ATOMICS

void **clSVMFree** (cl_context *context*, void *svm_pointer)

Enqueuing SVM operations

cl_int **clEnqueueSVMFree** (
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*, void *sym_pointers[],
void (CL_CALLBACK*pfn_free_func)(
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*,
void *sym_pointers[], void *user_data),
void *user_data, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueSVMMemcpy** (
cl_command_queue *command_queue*,
cl_bool *blocking_copy*, void *dst_ptr,
const void *src_ptr, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueSVMMemFill** (
cl_command_queue *command_queue*,
void *svm_ptr, const void *pattern,
size_t *pattern_size*, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueSVMMap** (
cl_command_queue *command_queue*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
void *svm_ptr, size_t *size*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueSVMUnmap** (
cl_command_queue *command_queue*,
void *svm_ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueSVMMigrateMem** (
cl_command_queue *command_queue*,
cl_uint *num_svm_pointers*, const void **svm_pointers*,
const size_t *sizes, cl_mem_migration_flags *flags*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

Flush and Finish [5.15]

cl_int **clFlush** (cl_command_queue *command_queue*)
cl_int **clFinish** (cl_command_queue *command_queue*)

Memory Objects

A memory object is a handle to a reference counted region of global memory. Includes buffer objects, image objects, and pipe objects. Items in blue apply when the appropriate extension is enabled.

Memory objects [5.5.1, 5.5.2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (
cl_mem *memobj*, void (CL_CALLBACK *pfn_notify)
(cl_mem *memobj*, void *user_data),
void *user_data)

cl_int **clEnqueueUnmapMemObject** (
cl_command_queue *command_queue*,
cl_mem *memobj*, void *mapped_ptr,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

Migrate memory objects [5.5.4]

cl_int **clEnqueueMigrateMemObjects** (
cl_command_queue *command_queue*,
cl_uint *num_mem_objects*,
const cl_mem *mem_objects,
cl_mem_migration_flags *flags*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

flags: CL_MIGRATE_MEM_OBJECT_HOST,
CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query memory object [5.5.5]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void *param_value, size_t *param_value_size_ret)

param_name: [Table 5.13]
CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_CONTEXT, CL_MEM_OFFSET,
CL_MEM_{MAP, REFERENCE}_COUNT,
CL_MEM_ASSOCIATED_MEMOBJECT,
CL_MEM_USES_SVM_POINTER,
CL_MEM_{D3D10, D3D11}_RESOURCE_KHR,
CL_MEM_DX9_MEDIA_ADAPTER_TYPE_KHR,
CL_MEM_DX9_MEDIA_SURFACE_INFO_KHR

Sampler Objects [5.7]

Items in blue require the *cl_khr_mipmap_image* extension.

cl_sampler
clCreateSamplerWithProperties (cl_context *context*,
const cl_sampler_properties *sampler_properties,
cl_int *errcode_ret)

sampler_properties: [Table 5.15]
CL_SAMPLER_NORMALIZED_COORDS,
CL_SAMPLER_{ADDRESSING, FILTER}_MODE,
CL_SAMPLER_MIP_FILTER_MODE,
CL_SAMPLER_LOD_{MIN, MAX}

cl_int **clRetainSampler** (cl_sampler *sampler*)
cl_int **clReleaseSampler** (cl_sampler *sampler*)

cl_int **clGetSamplerInfo** (cl_sampler *sampler*,
cl_sampler_info *param_name*,
size_t *param_value_size*, void *param_value,
size_t *param_value_size_ret)

param_name: CL_SAMPLER_REFERENCE_COUNT,
CL_SAMPLER_{CONTEXT, FILTER_MODE},
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_NORMALIZED_COORDS [Table 5.16]

Sampler declaration fields [6.13.14.1]

The sampler can be passed as an argument to the kernel using **clSetKernelArg**, or declared in the outermost scope of kernel functions, or it can be a constant variable of type sampler_t declared in the program source.

const sampler_t <*sampler-name*> =
<*normalized-mode*> | <*address-mode*> | <*filter-mode*>

normalized-mode:
CLK_NORMALIZED_COORDS_{TRUE, FALSE}

address-mode:
CLK_ADDRESS_*X*, where *X* may be NONE, REPEAT,
CLAMP, CLAMP_TO_EDGE, MIRRORED_REPEAT

filter-mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR

Program Objects

An OpenCL program consists of a set of kernels that are identified as functions declared with the __kernel qualifier in the program source.

Create program objects [5.8.1]

cl_program **clCreateProgramWithSource** (
cl_context *context*, cl_uint *count*, const char **strings*,
const size_t *lengths*, cl_int *errcode_ret)

cl_program **clCreateProgramWithIL** (cl_context *context*,
const void *il*, size_t *length*, cl_int *errcode_ret)

cl_program **clCreateProgramWithBinary** (
cl_context *context*, cl_uint *num_devices*,
const cl_device_id *device_list*, const size_t *lengths*,
const unsigned char **binaries*,
cl_int *binary_status*, cl_int *errcode_ret)

cl_program **clCreateProgramWithBuiltInKernels** (
cl_context *context*, cl_uint *num_devices*,
const cl_device_id *device_list*,
const char *kernel_names*, cl_int *errcode_ret)

Retain and release program objects [5.8.2]

cl_int **clRetainProgram** (cl_program *program*)
cl_int **clReleaseProgram** (cl_program *program*)

cl_int **clSetProgramReleaseCallback**
(cl_program *program*, void (CL_CALLBACK*pfn_notify)
(cl_program *prog*, void *user_data),
void *user_data)

Set SPIR-V specialization constants [5.8.3]

cl_int **clSetProgramSpecializationConstant** (
cl_program *program*, cl_uint *spec_id*, size_t *spec_size*,
const void *spec_value)

Building program executables [5.8.4]

cl_int **clBuildProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id *device_list*,
const char *options, void (CL_CALLBACK*pfn_notify)
(cl_program *program*, void *user_data),
void *user_data)

Separate compilation and linking [5.8.5]

cl_int **clCompileProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id *device_list*,
const char *options, cl_uint *num_input_headers*,
const cl_program *input_headers*,
const char **header_include_names*,
void (CL_CALLBACK*pfn_notify)
(cl_program *program*, void *user_data),
void *user_data)

cl_program **clLinkProgram** (cl_context *context*,
cl_uint *num_devices*, const cl_device_id *device_list*,
const char *options, cl_uint *num_input_programs*,
const cl_program *input_programs*,
void (CL_CALLBACK*pfn_notify)
(cl_program *program*, void *user_data),
void *user_data, cl_int *errcode_ret)

Unload the OpenCL compiler [5.8.8]

cl_int **clUnloadPlatformCompiler** (
cl_platform_id *platform*)

Query program objects [5.8.9]

cl_int **clGetProgramInfo** (cl_program *program*,
cl_program_info *param_name*,
size_t *param_value_size*, void *param_value*,
size_t *param_value_size_ret)

param_name: [Table 5.17]
CL_PROGRAM_{IL, REFERENCE_COUNT},
CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES},
CL_PROGRAM_{NUM_KERNELS, KERNEL_NAMES},
CL_PROGRAM_SCOPE_GLOBAL_{C,D}TORS_PRESENT

cl_int **clGetProgramBuildInfo** (
cl_program *program*, cl_device_id *device*,
cl_program_build_info *param_name*,
size_t *param_value_size*, void *param_value*,
size_t *param_value_size_ret)

param_name: [Table 5.18]
CL_PROGRAM_BINARY_TYPE,
CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG},
CL_PROGRAM_BUILD_GLOBAL_VARIABLE_TOTAL_SIZE

(Continued on next page >)

OpenCL API

Program Objects (continued)

Compiler options [5.8.6]

Preprocessor:
*(-D processed in order for **clBuildProgram** or **clCompileProgram**)*
-D *name* -D *name=definition* -I dir

Math intrinsics:
-cl-single-precision-constant
-cl-denorms-are-zero
-cl-fp32-correctly-rounded-divide-sqrt

Optimization options:
-cl-opt-disable -cl-mad-enable
-cl-no-signed-zeros -cl-finite-math-only
-cl-unsafe-math-optimizations -cl-fast-relaxed-math
-cl-uniform-work-group-size

Warning request/suppress:
-w -Werror

Control OpenCL C and C++ language version:
-cl-std=CL1.1 OpenCL 1.1 specification
-cl-std=CL1.2 OpenCL 1.2 specification
-cl-std=CL2.0 OpenCL 2.0 specification
-cl-std=C++ OpenCL C++ specification

Query kernel argument information:
-cl-kernel-arg-info

Debugging options:
-g Generate additional errors for built-in functions that allow you to enqueue commands on a device

SPIR binary options:
Requires the *cl_khr_spir* extension.
-x spir Indicate that binary is in SPIR format
-spir-std=*x* *x* is SPIR spec version, e.g.: 1.2

Double and half-precision floating-point in C++:
-cl-fp16-enable Enable full half data type support cl_khr_fp16 macro.
-cl-fp64-enable Enable full half data type support cl_khr_fp64 macro.

Linker options [5.8.7]

Library linking options:
-create-library -enable-link-options

Program linking options:
-cl-denorms-are-zero -cl-no-signed-zeroes
-cl-finite-math-only -cl-fast-relaxed-math
-cl-unsafe-math-optimizations

Kernel Objects

A kernel object encapsulates the specific __kernel function and the argument values to be used when executing it.

Create kernel objects [5.9.1]

cl_kernel **clCreateKernel** (cl_program *program*, const char **kernel_name*, cl_int **errcode_ret*)

cl_int **clCreateKernelsInProgram** (cl_program *program*, cl_uint *num_kernels*, cl_kernel **kernels*, cl_uint **num_kernels_ret*)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel arguments and queries [5.9.2-4]

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*, size_t *arg_size*, const void **arg_value*)

cl_int **clSetKernelArgSVMPointer** (cl_kernel *kernel*, cl_uint *arg_index*, const void **arg_value*)

cl_int **clSetKernelExecInfo** (cl_kernel *kernel*, cl_kernel_exec_info *param_name*, size_t *param_value_size*, const void **param_value*)

param_name: CL_KERNEL_EXEC_INFO_SVM_PTRS, CL_KERNEL_EXEC_INFO_SVM_FINE_GRAIN_SYSTEM

cl_kernel **clCloneKernel** (cl_kernel *source_kernel*, cl_int **errcode_ret*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*, cl_kernel_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.20] CL_KERNEL_{FUNCTION_NAME, NUM_ARGS}, CL_KERNEL_REFERENCE_COUNT, CL_KERNEL_{ATTRIBUTES, CONTEXT, PROGRAM}

cl_int **clGetKernelWorkGroupInfo** (cl_kernel *kernel*, cl_device_id *device*, cl_kernel_work_group_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_KERNEL_GLOBAL_WORK_SIZE, CL_KERNEL_[COMPILE_]WORK_GROUP_SIZE, CL_KERNEL_{COMPILE, MAX}_NUM_SUB_GROUPS, CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

cl_int **clGetKernelArgInfo** (cl_kernel *kernel*, cl_uint *arg_indx*, cl_kernel_arg_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.23] CL_KERNEL_ARG_NAME, CL_KERNEL_ARG_{ACCESS, ADDRESS}_QUALIFIER, CL_KERNEL_ARG_TYPE_{NAME, QUALIFIER}

cl_int **clGetKernelSubGroupInfo** (cl_kernel *kernel*, cl_device_id *device*, cl_kernel_sub_group_info *param_name*, size_t *input_value_size*, const void **input_value*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.22] CL_KERNEL_LOCAL_SIZE_FOR_SUB_GROUP_COUNT, CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDRANGE, CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE

Execute kernels [5.10]

cl_int **clEnqueueNDRangeKernel** (cl_command_queue *command_queue*, cl_kernel *kernel*, cl_uint *work_dim*, const size_t **global_work_offset*, const size_t **global_work_size*, const size_t **local_work_size*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueNativeKernel** (cl_command_queue *command_queue*, void (CL_CALLBACK **user_func*)(void *), void **args*, size_t *cb_args*, cl_uint *num_mem_objects*, const cl_mem **mem_list*, const void ***args_mem_loc*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

Event Objects

Event objects can be used to refer to a kernel execution command, and read, write, map, and copy commands on memory objects or user events.

Event objects [5.11]

cl_event **clCreateUserEvent** (cl_context *context*, cl_int **errcode_ret*)

cl_int **clSetUserEventStatus** (cl_event *event*, cl_int *execution_status*)

cl_int **clWaitForEvents** (cl_uint *num_events*, const cl_event **event_list*)

cl_int **clGetEventInfo** (cl_event *event*, cl_event_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.24] CL_EVENT_COMMAND_{QUEUE, TYPE}, CL_EVENT_{CONTEXT, REFERENCE_COUNT}, CL_EVENT_COMMAND_EXECUTION_STATUS

cl_int **clRetainEvent** (cl_event *event*)

cl_int **clReleaseEvent** (cl_event *event*)

cl_int **clSetEventCallback** (cl_event *event*, cl_int *command_exec_callback_type*, void (CL_CALLBACK **pfn_event_notify*) (cl_event *event*, cl_int *event_command_exec_status*, void **user_data*), void **user_data*)

Markers, barriers, & waiting for events [5.12]

cl_int **clEnqueueMarkerWithWaitList** (cl_command_queue *command_queue*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueBarrierWithWaitList** (cl_command_queue *command_queue*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

Profiling operations [5.14]

cl_int **clGetEventProfilingInfo** (cl_event *event*, cl_profiling_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: [Table 5.25] CL_PROFILING_COMMAND_{COMPLETE, QUEUED}, CL_PROFILING_COMMAND_{SUBMIT, START, END}

Memory Model: SVM [3.3.3]

OpenCL extends the global memory region into host memory through a shared virtual memory (SVM) mechanism. Three types of SVM in OpenCL:

- **Coarse-Grained buffer SVM**: (Required) Sharing at the granularity of regions of OpenCL buffer memory objects.
- **Fine-Grained buffer SVM**: (Optional) Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects.
- **Fine-Grained system SVM**: Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory.

Summary of SVM options in OpenCL [3.3.3, Table 3-2]

SVM	Granularity of sharing	Memory allocation	Mechanisms to enforce consistency	Explicit updates between host and device?
Non-SVM buffers	OpenCL Memory objects (buffer)	clCreateBuffer	Host synchronization points on the same or between devices.	Yes, through Map and Unmap commands.
Coarse-Grained buffer SVM	OpenCL Memory objects (buffer)	clSVMAlloc	Host synchronization points between devices	Yes, through Map and Unmap commands.
Fine Grained buffer SVM	Bytes within OpenCL Memory objects (buffer)	clSVMAlloc	Synchronization points plus atomics (if supported)	No
Fine-Grained system SVM	Bytes within Host memory (system)	Host memory allocation mechanisms (e.g. malloc)	Synchronization points plus atomics (if supported)	No

OpenCL C++ Language Reference

Section and table references are to the OpenCL 2.2 C++ Language specification.

Supported Data Types [3.1]

Header <*opencl_def*>

cl_* types have exactly the same size as their host counterparts defined in <*cl_platform.h*> file. Half types require *cl_khr_fp16*. Double types require that *cl_khr_fp64* be enabled and that CL_DEVICE_DOUBLE_FP_CONFIG is not zero.

Built-in scalar data types

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
double	cl_double	64-bit IEEE 754
half	cl_half	16-bit float (storage only)
void	void	empty set of values

Built-in vector data types

n is 2, 3, 4, 8, or 16. The half*n* vector data type is required to be supported as a data storage format.

OpenCL Type	API Type	Description
bool*n*		
[u]char*n*	cl_[u]char*n*	8-bit [un]signed
[u]short*n*	cl_ [u]short*n*	16-bit [un]signed
[u]int*n*	cl_ [u]int*n*	32-bit [un]signed
[u]long*n*	cl_ [u]long*n*	64-bit [un]signed
float*n*	cl_float*n*	32-bit float
double*n*	cl_double*n*	64-bit float
half*n*	cl_ half*n*	16-bit float

Other types [3.7.1, 3.8.1]

Header <*opencl_image*>

Image and sampler types require CL_DEVICE_IMAGE_SUPPORT is CL_TRUE. See header <opencl_pipe> for pipe type. See header <opencl_device_queue> for device_queue type.

Type in OpenCL C++	API type for application
cl::sampler	cl_sampler
cl::image[1d, 2d, 3d] cl::image1d_[buffer, array] cl::image2d_ms cl::image2d_array[_ms] cl::image2d_depth[_ms] cl::image2d_array_depth[_ms]	cl_image
cl::pipe	cl_pipe
cl::device_queue	cl_queue

half wrapper [3.6.1]

Header <*opencl_half*> OpenCL C++ implements a wrapper class for the built-in half data type. The class methods perform implicit vload_half and vstore_half operations from Vector Data Load and Store Functions section.

fp16(const half &*r*) noexcept;	Constructs an object with a half built-in type.
fp16(const float &*r*) noexcept;	Constructs an object with a float built-in type.
fp16(const double &*r*) noexcept;	Constructs an object with a double built-in type.

ndrange [3.13.6]

Header <*opencl_device_queue*> The ndrange type is used to represent the size of the enqueued workload with a dimension from 1 to 3.

```
struct ndrange {
  explicit ndrange(size_t global_work_size) noexcept;
  ndrange(size_t global_work_size, size_t local_work_size
          noexcept;
  ndrange(size_t global_work_offset, size_t global_work_size,
          size_t local_work_size) noexcept;
  template <size_t N>
  ndrange(const size_t (&global_work_size)[N]) noexcept;
  template <size_t N>
  ndrange(const size_t (&global_work_size)[N],
          const size_t (&global_work_size)[N]) noexcept;
  template <size_t N>
  ndrange(const size_t (&global_work_offset)[N],
          const size_t (&global_work_size)[N],
          const size_t (&global_work_size)[N]) noexcept;
};
```

Example

```
#include <opencl_device_queue>
#include <opencl_work_item>
using namespace cl;
kernel void foo(device_queue q) {
  q.enqueue_kernel(cl::enqueue_policy::no_wait, cl::ndrange( 1 ),
                   [](){ uint tid = get_global_id(0); } );
}
```

Preprocessor Directives & Macros [2.7]

```
#pragma OPENCL FP_CONTRACT on-off-switch
  on-off-switch: ON, OFF, or DEFAULT
#pragma OPENCL EXTENSION extensionname : behavior
#pragma OPENCL EXTENSION all : behavior
```

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_CPP_VERSION__	Integer version number, e.g: 100
__func__	Current function name

OpenCL C++ and C++ 14

The OpenCL C++ programming language is based on the ISO/IEC JTC1 SC22 WG21 N3690 language (a.k.a. C++14) specification with specific restrictions and exceptions. Section numbers denoted here with § refer to the C++ 14 specification.

- Implicit conversions for pointer types follow the rules described in the C++ 14 specification.
- Conversions between integer types follow the conversion rules specified in the C++14 specification except for specific out-of-range behavior and saturated conversions.
- The preprocessing directives defined by the C++14 specification are supported.
- Macro names defined by the C++14 specification but not currently supported by OpenCL are reserved for future use.
- OpenCL C++ standard library implements modified version of the C++ 14 numeric limits library.
- OpenCL C++ implements the following parts of the C++ 14 iterator library: Primitives, iterator operations, predefined iterators, and range access.
- The OpenCL C++ kernel language doesn't support variadic functions and variable length arrays.
- OpenCL C++ library implements most of the C++14 tuples except for allocator related traits (§ 20.4.2.8).
- OpenCL C++ supports type traits defined in the C++ 14 specification with additions and changes to the following:
  - UnaryTypeTraits (§ 3.15.1)
  - BinaryTypeTraits (§ 3.15.2)
  - TransformationTraits (§ 3.15.3)
- OpenCL C++ standard library implements most C++ 14 tuples excluding allocator related traits.
- C++14 features not supported by OpenCL C++:
  - the dynamic_cast operator (§ 5.2.7)
  - type identification (§ 5.2.8)
  - recursive function calls (§ 5.2.2, item 9) unless they are a compile-time constant expression
  - non-placement new and delete operators (§ 5.3.4, 5.3.5)
  - goto statement (§ 6.6)
  - register and thread_local storage qualifiers (§ 7.1.1)
  - virtual function qualifier (§ 7.1.2)
  - function pointers (§ 8.3.5, 8.5.3) unless they are a compile-time constant expression
  - virtual functions and abstract classes (§ 10.3, 10.4)
  - exception handling (§ 15)
  - the C++ standard library (§ 17 ... 30)
  - asm declaration (§ 7.4)
  - no implicit lambda to function pointer conversion (§ 5.1.2)

Qualifiers and Optional Attributes

Function Qualifier [2.6.1]

__kernel, kernel

Type and Variable Attributes [2.8]

[[cl::aligned(X)]] **[[cl::aligned]]**

Specifies a minimum alignment (in bytes) for variables of the specified type.

[[cl::packed]]

Specifies that each member of the structure or union is placed to minimize the memory required.

Kernel Function Attributes [2.8.3]

[[cl::work_group_size_hint(X, Y, Z)]]

A hint to the compiler to specify the value most likely to be specified by the *local_work_size* argument to **clEnqueueNDRangeKernel**.

[[cl::required_work_group_size(X, Y, Z)]]

The work-group size that must be used as the *local_work_size* argument to **clEnqueueNDRangeKernel**.

[[cl::required_num_sub_groups(X)]]

The number of sub-groups that must be generated by a kernel launch.

[[cl::vec_type_hint(<type>)]]

A hint to the compiler as a representation of the computational width of the kernel.

Kernel Parameter Attribute [2.8.4]

[[cl::max_size(*n*)]]

The value of the attribute specifies the maximum size in bytes of the corresponding memory object.

Loop Attributes [2.8.5]

[[cl::unroll_hint(*n*)]] **[[cl::unroll_hint]]**

Used to specify that a loop (for, while, and do loops) can be unrolled.

[[cl::ivdep(len)]] **[[cl::ivdep]]**

A hint to indicate that the compiler may assume there are no memory dependencies across loop iterations in order to autovectorize consecutive iterations of loop.

Conversions and Reinterpretation

Header <*opencl_convert*>

Conversion types [3.2]

Conversions are available for the scalar types bool, char, uchar, short, ushort, int, uint, long, ulong, half (if *cl_khr_fp16* extension is enabled), float, double (if *cl_khr_fp64* is enabled), and derived vector types.

```
template <class T, rounding_mode rmode, class U>
T convert_cast(U const& arg);

template <class T, rounding_mode rmode>
T convert_cast(T const& arg);

// and more...
```

Rounding modes [3.2.3]

::rte	to nearest even	::rtz	toward zero
::rtp	toward + infinity	::rtn	toward - infinity

Reinterpreting types [3.3]

Header <*opencl_reinterpret*>

Supported data types except bool and void may be reinterpreted as another data type of the same size using the **as_type** function for scalar and vector data types.

```
template <class T, class U>
T as_type(U const& arg);
```

Vector Component Addressing [2.1.2.3]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.r, v.s0	v.y, v.g, v.s1														
float3 v;	v.x, v.r, v.s0	v.y, v.g, v.s1	v.z, v.b, v.s2													
float4 v;	v.x, v.r, v.s0	v.y, v.g, v.s1	v.z, v.b, v.s2	v.w, v.a, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalences

Numeric indices are preceded by the letter s. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3 *	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Address Spaces Library

Header <*opencl_memory*>

Explicit address space storage classes [3.4.2]

global<T> class
Can only be used to declare variables at program scope, with static specifier, extern specifier, or passed as a kernel argument.

local<T> class
Can only be used to declare variables at kernel function scope, program scope, with static keyword, extern specifier, or passed as a kernel argument.

priv<T> class
Cannot be used to declare variables in the program scope, with static specifier, or extern specifier.

constant<T> class
Can only be used to declare variables at program scope, with static specifier, extern specifier, or passed as a kernel argument.

Explicit address space pointer classes [3.4.3]

The explicit address space pointer classes can be converted to and from pointers with compatible address spaces, qualifiers, and types. Local, global, and private pointers can be converted to standard C++ pointers.

```
typedef T element_type;
typedef ptrdiff_t difference_type;
typedef add_global_t<T>& reference;
typedef const add_global_t<T>& const_reference;
typedef add_global_t<T>* pointer;
typedef const add_global_t<T>* const_pointer;
```

The following pointer classes are defined in the header file <*opencl_memory*>:

template <class T> class global_ptr
template <class T> class local_ptr
template <class T> class private_ptr
template <class T> class constant_ptr

Non-member functions [3.4.3.9]

In each of the partial declarations below, the placeholder *Q* may be replaced with global, local, private, or constant. The omitted initial part of each declaration is:

template<class T, class U>

bool **operator==**(const *Q*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;
bool **operator!=**(const *OP*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;
bool **operator<**(const *Q*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;
bool **operator>**(const *Q*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;
bool **operator<=**(const *Q*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;
bool **operator>=**(const *Q*_ptr<T> &*a*, const *Q*_ptr<U> &*b*) noexcept;

In each of the partial declarations below, the omitted initial part of the declaration is:

template<class T>

bool **operator==**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator==**(nullptr_t, const *Q*_ptr<T> &x) noexcept;
bool **operator!=**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator!=**(nullptr_t, const *Q*_ptr global_ptr<T> & *x*) noexcept;
bool **operator<**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator<**(nullptr_t, const *Q*_ptr<T> & *x*) noexcept;
bool **operator>**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator>**(nullptr_t, const *Q*_ptr<T> & *x*) noexcept;
bool **operator<=**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator<=**(nullptr_t, const *Q*_ptr<T> & *x*) noexcept;
bool **operator>=**(const *Q*_ptr<T> &x, nullptr_t) noexcept;
bool **operator>=**(nullptr_t, const *Q*_ptr<T> & *x*) noexcept;
void **swap**(*Q*_ptr<T>& *a*, *Q*_ptr<T>& *b*) noexcept;

Pointer class constructors [3.4.3.5]

Q may be global, local, private, or constant.

constexpr ***Q*_ptr**() noexcept;	Construct an object which points to nothing
explicit ***Q*_ptr**(pointer *p*) noexcept;	Construct an object which points to *p*
***Q*_ptr**(const *Q*_ptr &) noexcept;	Copy constructor
***Q*_ptr**(*Q*_ptr &&*r*) noexcept;	Move constructor
constexpr ***Q*_ptr**(nullptr_t) noexcept;	Construct an object initialized with nullptr

Pointer class assignment operators [3.4.3.6]

Q may be global, local, private, or constant.

*Q*_ptr &**operator=**(const *Q*_ptr &*r*) noexcept;	Copy assignment operator
*Q*_ptr &**operator=**(*Q*_ptr &&*r*) noexcept;	Move assignment operator
*Q*_ptr &**operator=**(pointer *r*) noexcept;	Assign *r* pointer to the stored pointer
*Q*_ptr &**operator=**(nullptr_t) noexcept;	Assign nullptr to the stored pointer

Pointer class observers [3.4.3.7]

Q may be global, local, private, or constant.

add_lvalue_reference_t<add_*Q*<T>> **operator***() const noexcept;	Return **get()*
pointer **operator->**() const noexcept;	Return *get()*
reference **operator[]**(size_t *pos*) const noexcept;	Return *get()[pos]*
pointer **get**() const noexcept;	Return the stored pointer
explicit **operator bool**() const noexcept;	Return *get()!=nullptr*

Pointer class modifiers [3.4.3.8]

Q may be global, local, private, or constant.

pointer **release**() noexcept;	Assign nullptr to the stored pointer, returns the value that **get()** had at start
void **reset**(pointer *p* = pointer()) noexcept; void **reset**(pointer *p*) noexcept;	Assign *p* to the stored pointer
void **reset**(nullptr_t *p* = nullptr) noexcept;	Equivalent to reset(pointer())
void **swap**(*Q*_ptr& *r*) noexcept;	Invokes swap on the stored pointers.

*Q*_ptr **&operator++**() noexcept; *Q*_ptr **&operator--**() noexcept;	Prefix [in/de]crement stored pointer by one.
*Q*_ptr **operator++**(int) noexcept; *Q*_ptr **operator--**(int) noexcept;	Postfix [in/de]crement stored pointer by one.
*Q*_ptr **&operator+=**(difference_type *r*) noexcept;	Adds *r* to the stored pointer and returns **this*.
*Q*_ptr **&operator-=**(difference_type *r*) noexcept;	Subtracts *r* to the stored pointer and returns **this*.
*Q*_ptr **operator+**(difference_type *r*) noexcept;	[Adds/subtracts] *r* to the stored pointer and returns the value **this* has at the start of the operation.
*Q*_ptr **operator-**(difference_type *r*) noexcept;	

Other address space functions [3.4.4]

template <class T> mem_fence **get_mem_fence** (T **ptr*);	Return the *mem_fence* value for *ptr*.
template<class T, class U> T **dynamic_as_cast**(U **ptr*);	Returns a pointer to a region in the address space pointer class specified in T

Atomic Operations Library [3.24]

Header <*opencl_atomic*>

```
template<class T> struct atomic;
template<> struct atomic<integral>;
template<class T> struct atomic<T*>;
```

enum memory_order
memory_order_*x* where *x* may be relaxed, acquire, acq_rel, seq_cst, release

enum memory_scope
memory_scope_*x* where *x* may be work_item, work_group, sub_group, all_svm_devices, device

Atomic types [3.24.4]

Combined members from struct atomic, including specializations for integers (atomic<integral>) and pointers (atomic<T*>). For struct atomic<integral>, replace *T* with integral. For struct atomic<T*>, replace *T* with T*.
The pointer specialization is available if __INTPTR_WIDTH__== 32, or both extensions *cl_khr_int64_[base, extended]_atomics* are enabled and __INTPTR_WIDTH__== 64.

bool **is_lock_free**() const [volatile] noexcept;
void **store**(*T*, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
T **load**(memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) const [volatile] noexcept;
operator ***T***() const [volatile] noexcept;
T **exchange**(*T*, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
bool **compare_exchange_[weak, strong]**(*T*&, *T*, memory_order, memory_order, memory_scope) [volatile] noexcept;
bool **compare_exchange_[weak, strong]**(*T*&, *T*, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;
atomic() noexcept = default; constexpr **atomic**(*T*) noexcept;
T **operator=**(*T*) [volatile] noexcept;

(Continued on next page >)

Atomic Operations Library (continued)

Members available in specializations atomic<integral> and atomic<T*>. For struct atomic<integral>, replace *T* with integral, and for struct atomic<T*>, replace *T* with T*. ***op*** may be one of add, sub, and, or, xor, min, or max.

T **fetch_*op***(*T*, memory_order = memory_order_seq_cst, memory_scope = memory_scope_device) [volatile] noexcept;

Ti **operator[++, --]**([int]) [volatile] noexcept;

Ti **operator[+, -, &, |, ^]=**(*Ti*) [volatile] noexcept;

Atomic types

Pointer specializations indicated with a dot are available when these extensions are enabled:

- *cl_khr_fp64*
- *both cl_khr_int64_[base, extended]_atomics*

using atomic_[u]int = atomic<[u]int>;
using atomic_float = atomic<float>;
● using atomic_[u]long = atomic<[u]long>;
● && ● using atomic_double = atomic<double>;

Available if __INTPTR_WIDTH__ == 32, or both extensions *cl_khr_int64_[base, extended]_atomics* are enabled and __INTPTR_WIDTH__ == 64.

using atomic_intptr_t = atomic<intptr_t>;
using atomic_uintptr_t = atomic<uintptr_t>;

Available if __SIZE_WIDTH__ == 32, or both extensions *cl_khr_int64_[base, extended]_atomics* are enabled and __SIZE_WIDTH__ == 64:

using atomic_size_t = atomic<size_t>;

Available if __PTRDIFF_WIDTH__ == 32, or both extensions *cl_khr_int64_[base, extended]_atomics* are enabled and __PTRDIFF_WIDTH__ == 64:

using atomic_ptrdiff_t = atomic<ptrdiff_t>;

Members of struct atomic_flag:

atomic_flag() noexcept = default;

bool **test_and_set**(memory_order = *memory_order_seq_cst*, memory_scope = *memory_scope_device*) [volatile] noexcept;

void **clear**(memory_order = *memory_order_seq_cst*, memory_scope = *memory_scope_device*) [volatile] noexcept;

Non-member functions:

bool **atomic_flag_test_and_set**([volatile]atomic_flag*) noexcept;

bool **atomic_flag_test_and_set_explicit**([volatile]atomic_flag*, memory_order, memory_scope) noexcept;

void **atomic_flag_clear**([volatile]atomic_flag*) noexcept;

void **atomic_flag_clear_explicit**([volatile]atomic_flag*, memory_order, memory_scope) noexcept;

Fences [3.24.6]

void **atomic_fence**(mem_fence *flags*, memory_order *order*, memory_scope *scope*) noexcept;

flags: mem_fence::global, mem_fence::local, mem_fence::image or a combination of these values ORed together

scope: memory_scope_*x* where *x* may be all_svm_devices, device, work_group, sub_group, work_item

Images and Samplers Library [3.11]

Header <*opencl_image*>

struct sampler;

template <addressing_mode A, normalized_coordinates C, filtering_mode F> constexpr sampler **make_sampler**();

template <class T, image_access A, image_dim Dim, bool Depth, bool Array, bool MS> struct image;

Image types [3.11.2]

T is the type of value returned when reading or sampling from given image, or the type of color used to write to image.

using image1d = image<T, A, image_dim::image_1d, false, false, false>;
using image1d_buffer = image<T, A, image_dim::image_buffer, false, false, false>;
using image1d_array = image<T, A, image_dim::image_1d, false, true, false>;

using image2d = image<T, A, image_dim::image_2d, false, false, false>;
using image2d_depth = image<T, A, image_dim::image_2d, true, false, false>;
using image2d_array = image<T, A, image_dim::image_2d, false, true, false>;
using image2d_array_depth = image<T, A, image_dim:: image_2d, true, true, false>;

using image3d = image<T, A, image_dim::image_3d, false, false, false>;

The extensions *cl_khr_gl_msaa_sharing* and *cl_khr_gl_depth_images* add the following functions.

using image2d_ms = image<T, A, image_dim::image_2d, false, false, true>;
using image2d_array_ms = image<T, A, image_dim::image_2d, false, true, true>;
using image2d_depth_ms = image<T, A, image_dim::image_2d, true, false, true>;
using image2d_array_depth_ms = image<T, A, image_dim::image_2d, true, true, true>;

Image element types [3.11.4]

In OpenCL terminology, images are classified as depth images, which have the Depth template parameter set to true, or normal images, which have the Depth template parameter set to false. Half types are only available if *cl_khr_fp16* extension is enabled.

depth images	Non-multisample depth image types: float, half For multi-sample 2D and multi-sample 2D array images, only valid type: float
normal images	Valid types: float4, int4, uint4, and half4 For multi-sample 2D and multi-sample 2D array images, only valid types: float4, int4 and uint4

Image dimension [3.11.5]

template <image_dim Dim>
struct image_dim_num;

enum image_dim
image_1d, image_2d, image_3d, image_buffer

Members of class image

Members indicated with a dot are available when these extensions are enabled:

- *cl_khr_mipmap_image[_writes]*
- *cl_khr_gl_msaa_sharing* and *cl_khr_gl_depth_images*

For images specified with image_dim::image1d and image_dim::buffer

int **width**() const noexcept;
● int **width**(float *lod*) const noexcept;

For images specified with image_dim::image2d

int **[width, height]**() const noexcept;
● int **[width, height]**(float *lod*) const noexcept;
● int **num_samples**() const noexcept;

For images specified with image_dim::image3d

int **[width, height, depth]**() const noexcept;
● int **[width, height, depth]**(float *lod*) const noexcept;

For arrayed images

int **array_size**() const noexcept;
● int **array_size**(int *lod*) const noexcept;

Image access [3.11.6]

enum image_access
sample, read, write, read_write

Members of class image

The non-multisample image template class specializations present different sets of methods based on their access parameter. Members indicated with a dot are available when these extensions are enabled:

- *cl_khr_mipmap_image*
- ● *cl_khr_mipmap_image_writes*
- ● *cl_khr_gl_msaa_sharing* and *cl_khr_gl_depth_images*

For images specified with image_access::read

element_type image::**read**(integer_coord *coord*) const noexcept;
pixel image::**operator[]**(integer_coord *coord*) const noexcept;
element_type image::pixel::operator **element_type**() const noexcept;
● element_type image::**read**(integer_coord *coord*, int *sample*) noexcept;

For images specified with image_access::write

void image::**write**(integer_coord *coord*, element_type *color*) noexcept;
image::pixel image::**operator[]**(integer_coord *coord*) noexcept;
image::pixel & image::pixel::**operator=**(element_type *color*) noexcept;
● void image::**write**(integer_coord *coord*, element_type *color*, int *lod*) noexcept;

For images specified with image_access::read_write

element_type image::**read**(integer_coord *coord*) const noexcept;
void image::**write**(integer_coord *coord*, element_type *color*) noexcept;
image::pixel image::**operator[]**(integer_coord *coord*) noexcept;
element_type image::pixel::operator **element_type**( ) const noexcept;
image::pixel & image::pixel::**operator=**(element_type *color*) noexcept;

For images specified with image_access::sample

element_type image::**read**(integer_coord *coord*) const noexcept;
element_type image::**sample**(const sampler &*s*, integer_coord *coord*) const noexcept;
element_type image::**sample**(const sampler &*s*, float_coord *coord*) const noexcept;
image::pixel image::**operator[]**(integer_coord *coord*) const noexcept;
element_type image::pixel::operator **element_type**( ) const noexcept;
● element_type image::**sample**(const sampler &*s*, float_coord *coord*, float *lod*) const noexcept;
● element_type image::**sample**(const sampler &*s*, integer_coord *coord*, gradient_coord *gradient_x*, gradient_coord *gradient_y*) const noexcept;

Common image methods [3.11.7]

Each image type implements this set of common members. Member indicated with a dot is available when these extensions are enabled:

- *cl_khr_mipmap_image[_writes]*

image_channel_type image::**data_type**() const noexcept;
image_channel_order image::**order**() const noexcept;
● int image::**miplevels**() const noexcept;

enum image_channel_type
snorm_int8, snorm_int16, unorm_int8, unorm_int16, unorm_int24, unorm_short_565, unorm_short_555, unorm_short_101010, unorm_short_101010_2, sint8, sint16, sint32, uint8, uint16, uint32, float16, float32

enum image_channel_order
a, r, rx, rg, rgx, ra, rgb, rgbx, rgba, argb, bgra, intensity, luminance, abgr, srgb, srgbx, srgba, sbgra, depth, depth_stencil

(Continued on next page >)

Images and Samplers Library (continued)

Other image methods [3.11.8]

Members indicated with a dot are available when these extensions are enabled:

- *cl_khr_mipmap_image*
- *cl_khr_mipmap_image* or *cl_khr_mipmap_image_writes*
- *cl_ khr_gl_msaa_sharing* and *cl_khr_gl_depth_images*

element_type image::**sample**(const sampler &*s*, float_coord *coord*) const noexcept;
element_type image::**sample**(const sampler &*s*, integer_coord *coord*) const noexcept;
• element_type image::**sample**(const sampler &*s*, float_coord *coord*, float *lod*) const noexcept;
• element_type image::**sample**(const sampler &*s*, float_coord *coord*, gradient_coord *gradient_x*, gradient_coord *gradient_y*) const noexcept;

element_type image::**read**(integer_coord *coord*) const noexcept;
• void image::**read**(integer_coord *coord*, int *sample*) noexcept;

void image::**write**(integer_coord *coord*, element_type *color*) noexcept;
• void image::**write**(integer_coord *coord*, element_type *color*, int *lod*) noexcept;

pixel **operator[]**(integer_coord *coord*) noexcept;
pixel **operator[]**(integer_coord *coord*) const noexcept;

element_type pixel::**operator element_type**() const noexcept;

pixel & pixel::**operator=**(element_type *color*) noexcept;

int **width**() const noexcept;
• int **width**(int *lod*) const noexcept;

int **height**() const noexcept;
• int **height**(int *lod*) const noexcept;

int **depth**() const noexcept;
• int **depth**(int *lod*) const noexcept;

int **array_size**() const noexcept;
• int **array_size**(int *lod*) const noexcept;

image_channel_type image::**data_type**() const noexcept;

image_channel_order image::**order**() const noexcept;

integer_coord **size**() const noexcept;
• int **miplevels**() const noexcept;
• int **num_samples**() const noexcept;

Sampler [3.11.9]

Acquire a sampler inside of a kernel by passing it as a kernel parameter from host using **clSetKernelArg**, or creating it using the **make_sampler** function in the kernel code.

enum addressing_mode
mirrored_repeat, repeat, clamp_to_edge, clamp, none

enum normalized_coordinates
normalized, unnormalized

enum normalized_coordinates
nearest, linear

Pipes Library

Header <*opencl_pipe*>

Use pipe and pipe_storage template classes as a communication channel between kernels.

```
enum class pipe_access { read, write };
template <class T, pipe_access Access = pipe_access::read> struct pipe;
template<cl::pipe_access Access = pipe_access::read, class T, size_t N> pipe<T, Access>
```

class pipe methods [3.8.4]

When pipe_access is:	Member function	Description
read	bool **read**(T& *ref*) const noexcept;	Read packet from pipe into *ref*.
write	bool **write**(const T& *ref*) noexcept;	Write packet specified by *ref* to pipe.
read	reservation<memory_scope_work_item> **reserve**(uint *num_packets*) const noexcept;	Reserve *num_packets* entries for reading/writing from/to pipe.
write	reservation<memory_scope_work_item> **reserve**(uint *num_packets*) noexcept;	
read	reservation<memory_scope_work_group> **work_group_reserve**(uint *num_packets*) const noexcept;	
write	reservation<memory_scope_work_group> **work_group_reserve**(uint *num_packets*) noexcept;	
read	reservation <memory_scope_sub_group> **sub_group_reserve**(uint *num_packets*) const noexcept;	
write	reservation <memory_scope_sub_group> **sub_group_reserve**(uint *num_packets*) noexcept;	
read, write	uint **num_packets**() const noexcept;	Returns current number of packets that have been written to but not yet been read from the pipe.
read, write	uint **max_packets**() const noexcept;	Returns max. number of packets specified when pipe was created.

When pipe_access is:	Member function	Description
read	bool pipe::reservation::**read**(uint *index*, T& *ref*) const noexcept;	Read packet from the reserved area of the pipe referred to by index into *ref*.
write	bool pipe::reservation::**write**(uint *index*, const T& ref) noexcept;	Write packet specified by *ref* to the reserved area of the pipe referred to by *index*.
read	void pipe::reservation::**commit**() const noexcept;	Indicates that all reads/writes to num_packets associated with reservation are completed.
write	bool pipe::reservation::**commit**() noexcept;	
read	bool pipe::reservation::**is_valid**();	Return true if reservation is a valid reservation ID.
write	bool pipe::reservation::**is_valid**() const noexcept;	
read, write	explicit pipe::reservation::**operator bool**() const noexcept;	

Non-member functions

template<pipe_access Access = pipe_access::read, class T, size_t N> pipe<T, Access> **make_pipe**(const pipe_storage <T, N>& *ps*);	Constructs a read only or write only pipe from pipe_storage object.

pipe_storage class [3.8.5]

N in the following declaration specifies the maximum number of packets which can be held by an object.

```
template <class T, size_t N> struct pipe_storage;
```

Members of struct pipe_storage:

pipe_storage(); **pipe_storage**(const pipe_storage&) = default; **pipe_storage**(pipe_storage&&) = default; template<pipe_access Access = pipe_access::read> pipe<T, Access> **get**() noexcept;	Constructs a read only or write only pipe from pipe_storage object.

Device Enqueue Library [3.13]

Header <*opencl_device_queue*>

Allows a kernel to independently enqueue the same device, without host interaction.

enum enqueue_policy
no_wait, wait_kernel, wait_work_group

enum event_status
submitted, complete, error

enum enqueue_status
success, failure, invalid_queue, invalid_ndrange, invalid_event_wait_list, queue_full, invalid_arg_size, event_allocation_failure, out_of_resources

enum event_profiling_info
exec_time

Members of struct device_queue [3.13.3]

struct device_queue: marker_type;

enqueue_status **enqueue_marker**(uint *num_events_in_wait_list*, const event **event_wait_list*, event **event_ret*) noexcept;	Enqueues a marker to device queue after a list of events specified by event_wait_list completes.
template <class Fun , class... Args> enqueue_status **enqueue_kernel**(enqueue_policy *policy*, const ndrange &*ndrange*, Fun *fun*, Args... *args*) noexcept;	Enqueue functor or lambda fun on the device with specified policy over the specified ndrange.
template <class Fun, class... Args> enqueue_status **enqueue_kernel**(enqueue_policy *policy*, uint *num_events_in_wait_list*, const event **event_wait_list*, event **event_ret*, const ndrange &*ndrange*, Fun *fun*, Args... *args*) noexcept;	Enqueues functor or lambda fun in the same way as the overload above with the exception for the passed event list.
device_queue(const device_queue&) = default; **device_queue**(device_queue&&) = default;	Constructors

Members of struct event [3.13.4]

struct event;

bool **is_valid**() const noexcept;	Returns true if event object is a valid event.
explicit **operator bool**() const noexcept;	Returns true if event object is a valid event.
void **retain**() noexcept;	Increments the event reference count.
void **release**() noexcept;	Decrements the event reference count.
void **set_status**(event_status *status*) noexcept;	Sets the execution status of a user event.
void **profiling_info**(event_profiling_info *name*, global_ptr<long> *value*) noexcept;	Captures the profiling information for functions that are enqueued as commands.

(Continued on next page >)

Device Enqueue Library (continued)

Non-member functions [3.13.5]

device_queue **get_default_device_queue**();	Returns the default device queue.
event **make_user_event**();	Creates, returns, and sets the execution status of the user event to event_status::submitted.
template <class Fun, class... Args>uint **get_kernel_work_group_size**(Fun *fun*, Args... args);	Provides a mechanism to query the maximum work-group size that can be used to execute a functor
template <class Fun, class... Args> uint **get_kernel_preferred_work_group_size_multiple**(Fun *fun*, Args... *args*);	Returns the preferred multiple of work-group size for launch.
template <class Fun, class... Args> uint **get_kernel_sub_group_count_for_ndrange**(const ndrange & *ndrange*, Fun *fun*, Args... *args*);	Returns the number of sub-groups in each work-group of the dispatch
template <class Fun, class... Args> uint **get_kernel_max_sub_group_size_for_ndrange**(const ndrange & *ndrange*, Fun *fun*, Args... *args*);	Returns the maximum sub-group size for a functor.
template <class Fun, class... Args> uint **get_kernel_local_size_for_sub_group_count**(uint *num_sub_groups*, Fun *fun*, Args... *args*);	Returns a valid local size that would produce the requested number of sub-groups such that each sub-group is complete with no partial sub-groups
template <class Fun, class... Args> uint **get_kernel_max_num_sub_groups**(Fun *fun*, Args... *args*);	Provides a mechanism to query the maximum number of sub-groups that can be used to execute the passed functor on the current device.

Enqueue enums [3.13.7-11]

enum enqueue_policy
no_wait, wait_kernel, wait_work_group

enum enqueue_status
success, failure, invalid_queue, invalid_ndrange, invalid_event_wait_list, queue_full, invalid_arg_size, event_allocation_failure, out_of_resources

enum event_status
submitted, complete, error

enum event_profiling_info
exec_time

Synchronization Functions [3.16]

Header <*opencl_synchronization*>

struct work_group_named_barrier: marker_type;

Barriers [3.16.2]

void **work_group_barrier**(mem_fence *flags*, memory_scope *scope* = memory_scope_work_group);	Work-items in a work-group must execute this before any can continue
void **sub_group_barrier**(mem_fence *flags*, memory_scope *scope* = memory_scope_work_group);	Work-items in a sub-group must execute this before any can continue

flags: mem_fence::global, mem_fence::local, mem_fence::image or a combination of these values ORed together

scope: memory_scope_*x* where *x* may be all_svm_devices, device, work_group, sub_group, work_item

Named barriers [3.16.3]

Members from struct work_group_named_barrier. work_group_named_barrier requires the *cl_khr_sub_group_named_barrier* extension be enabled.

work_group_named_barrier(uint *sub_group_count*);	Initialize a new named barrier object to synchronize sub_group_count sub-groups in the current work-group.
work_group_named_barrier(const work_group_named_barrier&) = default;	
work_group_named_barrier(work_group_named_barrier&&) = default;	
wait(mem_fence *flags*, memory_scope scope = memory_scope_work_group) const noexcept;	All work-items in a sub-group executing the kernel on a processor must execute this method before any are allowed to continue.

Work-Item Functions [3.14]

Header <*opencl_work_item*>

Query the number of dimensions, global, and local work size specified to **clEnqueueNDRangeKernel**, and global and local identifier of each work-item when this kernel is executed on a device.

uint **get_work_dim**();	Number of dimensions in use
size_t **get_global_size**(uint *dimindx*);	Number of global work-items
size_t **get_global_id**(uint *dimindx*);	Global work-item ID value
size_t **get_local_size**(uint *dimindx*);	Number of local work-items if kernel executed with uniform work-group size
size_t **get_enqueued_local_size**(uint *dimindx*);	Number of local work-items
size_t **get_local_id**(uint *dimindx*);	Local work-item ID
size_t **get_num_groups**(uint *dimindx*);	Number of work-groups
size_t **get_group_id**(uint *dimindx*);	Work-group ID
size_t **get_global_offset**(uint *dimindx*);	Global offset
size_t **get_global_linear_id**();	Work-items 1-dimensional global ID
size_t **get_local_linear_id**();	Work-items 1-dimensional local ID
size_t **get_sub_group_size**();	Number of work-items in the subgroup
size_t **get_max_sub_group_size**();	Maximum size of a subgroup
size_t **get_num_sub_groups**();	Number of subgroups
size_t **get_enqueued_num_sub_groups**();	If kernel executed with a uniform work-group size, results are same as for **get_num_sub_groups**.
size_t **get_sub_group_id**();	Sub-group ID
size_t **get_sub_group_local_id**();	Unique work-item ID

Workgroup Functions [3.15]

Header <*opencl_work_group*>

Logical operations [3.15.2]

bool **work_group_all** (bool *predicate*) bool **work_group_any** (bool *predicate*)	Evaluates predicate for all work-items in the work-group and returns true if predicate evaluates to true for all/any work-items in the work-group.
bool **sub_group_all** (bool *predicate*) bool **sub_group_any** (bool *predicate*)	Evaluates predicate for all work-items in the sub-group and returns a non-zero value if predicate evaluates to non-zero for all/any work-items in the sub-group.

Broadcast functions [3.15.3]

T is type int, uint, long, ulong, or float, double (if *cl_khr_fp64* is enabled) or half (if *cl_khr_fp16* is enabled).

T **work_group_broadcast**(*T a*, size_t *local_id*); *T* **work_group_broadcast**(*T a*, size_t *local_id_x*, size_t *local_id_y*); *T* **work_group_broadcast**(*T a*, size_t *local_id_x*, size_t *local_id_y*, size_t *local_id_z*);	Broadcast the value of a for work-item identified by local_id to all work-items in the work-group.
T **sub_group_broadcast**(T *x*, size_t *sub_group_local_id*);	Broadcast the value of x for work-item returned by get_sub_group_local_id to all work-items in the sub-group.

Numeric operations [3.15.4]

enum work_group_op
add, min, max

T is type int, uint, long, ulong, or float, double (if *cl_khr_fp64* is enabled) or half (if *cl_khr_fp16* is enabled).

template <work_group_op *op*> *T* **work_group_reduce**(*T x*);	Return result of reduction operation <*op*> for all values of x specified by work-items in a work-group.
template <work_group_op *op*> *T* **work_group_scan_[ex, in]clusive**(*T x*);	Perform an exclusive/inclusive scan operation <*op*> of all values specified by work-items in the work-group.
template <work_group_op *op*> *T* **sub_group_reduce**(*T x*);	Return result of reduction operation <*op*> for all values of x specified by work-items in a sub-group.
template <work_group_op *op*> *T* **sub_group_scan_[ex, in]clusive**(*T x*);	Perform an exclusive/inclusive scan operation <*op*> of all values specified by work-items in a sub-group. The scan results are returned for each work-item.

Math Functions [3.19]

Header <*opencl_math*>

Vector versions of the math functions operate component-wise. The description is per-component. T is half*n* (if *cl_khr_fp16* is enabled), float*n*, or double*n* (if *cl_khr_fp64* is enabled), where *n* is 2, 3, 4, 8, or 16. T*f* may only be float*n*. All angles are in radians.

Trigonometric functions [3.19.2]

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	Compute acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	Compute asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	Compute atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	Compute atan2 (*y*, *x*) / π
T **cos** (*T x*) T*f* native_math::**cos**(T*f x*); T*f* half_math::**cos**(T*f x*);	Cosine, *x* is an angle
T **cosh** (*T x*)	Hyperbolic cosine
T **cospi** (*T x*)	Compute cos (π *x*)
T **sin** (*T x*) T*f* native_math::**sin**(T*f x*); T*f* half_math::**sin**(T*f x*);	Sine, *x* is an angle
T **sincos** (*T x*, *T *cosval*)	Sine and cosine of *x*
T **sinh** (*T x*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **tan** (*T x*) T*f* native_math::**tan**(T*f x*); T*f* half_math::**tan**(T*f x*);	Tangent
T **tanh** (*T x*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)

Power functions [3.19.3]

T **cbrt** (*T*)	Cube root
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y*
float*n* **pown** (*T x*, int*n y*)	Compute *x* *y*, where *y* is an integer
T **powr** (*T x*, *T y*) T*f* native_math::**powr**(T*f x*, T*f y*); T*f* half_math::**powr**(T*f x*, T*f y*);	Compute *x* *y*, where *x* is >= 0
T*f* **rootn** (*T x*, int*n y*)	Compute *x* to the power of 1/*y*
T **rsqrt** (*T*) T*f* native_math::**rsqrt**(T*f x*); T*f* half_math::**rsqrt**(T*f x*);	Inverse square root
T **sqrt** (*T*) T*f* native_math::**sqrt**(T*f x*); T*f* half_math::**sqrt**(T*f x*);	Square root

Logarithmic functions [3.19.4]

int*n* **ilogb** (*T x*)	Return exponent as an integer value
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, int*n *signp*)	Log gamma function
T **log** (*T*) T*f* native_math::**log**(T*f x*); T*f* half_math::**log**(T*f x*);	Natural logarithm
T **log2** (*T*) T*f* native_math::**log2**(T*f x*); T*f* half_math::**log2**(T*f x*);	Base 2 logarithm
T **log10** (*T*) T*f* native_math::**log10**(T*f x*); T*f* half_math::**log10**(T*f x*);	Base 10 logarithm
T **log1p** (*T x*)	Compute loge (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*

Exponential functions [3.19.5]

T **exp** (*T x*) T*f* native_math::**exp**(T*f x*); T*f* half_math::**exp**(T*f x*);	Exponential base-e exp. of *x*
T **exp2** (*T*) T*f* native_math::**exp2**(T*f x*); T*f* half_math::**exp2**(T*f x*);	Exponential base 2
T **exp10** (*T x*) T*f* native_math::**exp10**(T*f x*); T*f* half_math::**exp10**(T*f x*);	Exponential base 10
T **expm1** (*T x*)	Compute ex -1.0
T **ldexp** (*T x*, int*n k*)	*x* * 2k

Floating point functions [3.19.6]

T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **floor** (*T*)	Round to integer toward infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *T *iptr*)	Fractional value in *x*
T **frexp** (*T x*, int*n *exp*)	Extract mantissa and exponent
T **modf** (*T x*, *T *iptr*)	Decompose floating-point number
float*n* **nan** (uint*n nancode*) double*n* **nan** (ulong*n nancode*) half*n* **nan** (ushort*n nancode*)	Quiet NaN
T **nextafter** (*T x*, *T y*)	Next representable floating-point value after *x* in the direction of *y*
T **remainder** (*T x*, *T y*)	Floating point remainder
T **remquo** (*T x*, *T y*, int*n *quo*)	Remainder and quotient
T **rint** (*T x*)	Round to nearest even integer
T **round** (*T x*)	Integral value nearest to *x* rounding
T **trunc** (*T x*)	Return integral value nearest to *x* rounding halfway cases away from zero.

Comparison functions [3.19.7]

T **fdim** (*T x*, *T y*)	Positive difference between *x* and *y*
T **fmax** (*T x*, *T y*)	Return *y* if *x* < *y*, else returns *x*
T **fmin** (*T x*, *T y*)	Return *y* if *y* < *x*, else returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*
T **minmag** (*T x*, *T y*)	Minimum magnitude of *x* and *y*

Other functions [3.19.8]

T*f* native_math::**divide**(T*f x*, T*f y*); T*f* half_math::**divide**(T*f x*, T*f y*);	Compute *x* / *y*
T **erfc** (*T*)	Complementary error function.
T **erf** (*T x*)	Calculates error function of *T*
T **fabs** (*T x*)	Absolute value
T **hypot** (*T x*, *T y*)	Square root of *x*2 + *y*2
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T*f* native_math::**recip**(T*f x*); T*f* half_math::**recip**(T*f x*);	Reciprocal
T **tgamma** (*T x*)	Gamma function

Integer Built-in Functions [3.20]

Header <*opencl_integer*>

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*, *where n is 2, 3, 4, 8, or 16. Tu* is the unsigned version of *T. Ts*c is the scalar version of *T.*

bitwise functions [3.20.2]

T **clz** (*T x*)	Number of leading 0-bits in *x*
T **ctz** (*T x*)	Number of trailing 0-bits in *x*
T **popcount** (*T x*)	Number of non-zero bits in *x*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
For ***upsample****, return type is scalar when the parameters are scalar.*	
short[*n*] **upsample** (char[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]= (([u]short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort[*n*] **upsample** (uchar[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int[*n*] **upsample** (short[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint[*n*] **upsample** (ushort[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long[*n*] **upsample** (int[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong[*n*] **upsample** (uint[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

numeric functions [3.20.3]

Tu **abs** (*T x*)	| *x* |
Tu **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *Tsc min*, *Tsc max*)	min(max(*x*, *minval*), *maxval*)
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *Tsc y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*) *T* **min** (*T x*, *Tsc y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	High half of the product of *x* and *y*
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result

24-bit operations [3.20.4]

The following fast integer functions optimize the performance of kernels. In these functions, *T* is type int, uint, int*n* or uint*n*, where *n* is 2, 3, 4, 8, or 16.

int*n* **mad24** (*T x*, *T y*, *T z*)	Multiply 24-bit integer values *x*, *y*, add 32-bit int. result to 32-bit integer *z*
T **mul24** (*T x*, *T y*)	Multiply 24-bit integer values *x* and *y*

Common Functions [3.17]

Header <*opencl_common*>

These functions are implemented using the round to nearest even rounding mode. Vector versions operate component-wise. *Ts* is type float, optionally double (if *cl_khr_fp64* is enabled), or half if *cl_khr_fp16* is enabled. *Tn* is the vector form of *Ts, where n is 2, 3, 4, 8, or 16. T* is *Ts* and *Tn*.

T **clamp** (*T x*, *T min*, *T max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*)	Max of *x* and *y*
T **min** (*T x*, *T y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

Geometric Functions [3.18]

Header <*opencl_geometric*>

These functions use the round to nearest even rounding mode. Vector versions operate component-wise. *Ts* is scalar type float, double if *cl_khr_fp64* is enabled, or half if *cl_khr_fp16* is enabled. *Tn* is the vector form of *Ts* with 2, 3, or 4 components.

float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product
Ts **distance** (*T p0*, *T p1*)	Vector distance
Ts **dot** (*T p0*, *T p1*)	Dot product
Ts **length** (*T p*)	Vector length
T **normalize** (*T p*)	Normal vector length 1

Vector Data Load/Store [3.22]

Header <opencl_vector_load_store>

T is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double (if *cl_khr_fp64* is enabled), or half (if *cl_khr_fp16* is enabled). *Tn* refers to the vector form of type *T*, where *n* is 2, 3, 4, 8, or 16.

template <size_t N, class T> make_vector_t<T, N> **vload**(size_t *offset*, const T* *p*); template <size_t N, class T> make_vector_t<T, N> **vload**(size_t *offset*, const constant_ptr<T> *p*);	Read vector data from address (*p* + (*offset* * *n*))
template <size_t N> make_vector_t<float, N> **vload_half**(size_t *offset*, const half* *p*); template <size_t N> make_vector_t<float, N> **vload_half**(size_t*offset*, const constant_ptr<half> *p*);	Read a half*n* from address (*p* + (*offset* * *n*))
template <size_t N> make_vector<float, N> **vloada_half**(size_t *offset*, const half* *p*); template <size_t N> make_vector<float, N> **vloada_half**(size_t *offset*, const constant_ptr<half> *p*);	Read half vector from (*p* + (*offset* * *n*)). For half3, read from (*p* + (*offset* * 4)).
template <class T> void **vstore**(T *data*, size_t *offset*, vector_element_t<T>* *p*);	Write vector data to address (*p* + (*offset* * *n*)
template <rounding_mode rmode = rounding_mode::rte, class T> void **vstore_half**(T *data*, size_t *offset*, half* *p*);	Write a half to address (*p* + *offset*)
template <rounding_mode rmode = rounding_mode::rte, class T> void **vstorea_half**(T *data*, size_t *offset*, half* *p*);	Write a half vector to address (*p* + (*offset* * *n*))

Array Library [3.25]

Header <*opencl_array*>

template<class T, size_t N> struct array;

Iterators from struct array

[const_]iterator **begin**() [const] noexcept;
[const_]iterator **end**() [const] noexcept;
[const_]reverse_iterator **rbegin**() [const] noexcept;
[const_]reverse_iterator **rend**() [const] noexcept;
const_iterator **cbegin**() const noexcept;
const_iterator **cend**() const noexcept;
const_reverse_iterator **crbegin**() const noexcept;
const_reverse_iterator **crend**() const noexcept;

Capacities from struct array

constexpr size_type **size**() const noexcept;
constexpr size_type **max_size**() const noexcept;
constexpr bool **empty**() const noexcept;

Relational Built-in Functions [3.21]

Header <opencl_relational>

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result. *T* is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, or optionally double or double*n* (if *cl_khr_fp64* is enabled) or half or half*n* (if *cl_khr_fp16* is enabled). *n* is 2, 3, 4, 8, or 16.

bool*n* **isequal**(float*n* *x*, float*n* *y*); bool*n* **isequal**(half*n* *x*, half*n* *y*); bool*n* **isequal**(double*n* *x*, double*n* *y*);	Compare of *x* == *y*
bool*n* **isnotequal**(float*n* *x*, float*n* *y*);	Compare of *x* != *y*
bool*n* **isgreater**(float*n* *x*, float*n* *y*);	Compare of *x* > *y*
bool*n* **isgreaterequal**(float*n* *x*, float*n* *y*);	Compare of *x* >= *y*
bool*n* **isless**(float*n* *x*, float*n* *y*);	Compare of *x* < *y*
bool*n* **islessequal**(float*n* *x*, float*n* *y*);	Compare of *x* <= *y*
bool*n* **islessgreater**(float*n* *x*, float*n* *y*);	Compare of (*x* < *y*) || (*x* > *y*)
bool*n* **isordered**(float*n* *x*, float*n* *y*);	Test if arguments are ordered
bool*n* **isunordered**(float*n* *x*, float*n* *y*);	Test if arguments are unordered
bool*n* **isfinite**(float*n* *x*, float*n* *y*);	Test for finite value
bool*n* **isinf**(float*n* *x*, float*n* *y*);	Test for + or – infinity
bool*n* **isnan**(float*n* *x*, float*n* *y*);	Test for a NaN
bool*n* **isnormal**(float*n* *x*, float*n* *y*);	Test for a normal value
bool*n* **signbit**(float*n* *x*, float*n* *y*);	Test for sign bit
bool **any**(bool*n* *t*);	1 if MSB in component of *x* is set; else 0
bool **all**(bool*n* *t*);	1 if MSB in all components of *x* are set; else 0
T **bitselect**(*T a*, *T b*, *T c*);	Each bit of result is corresponding bit of *a* if corresponding bit of *c* is 0
T **select**(*T a*, *T b*, bool*n* *c*);	For each component of a vector type, result[i] = if MSB of *c*[i] is set ? *b*[i] : *a*[i] For scalar type, result = *c* ? *b* : *a*

printf Function [3.23]

Header <*opencl_printf*>

Writes output to an implementation-defined stream.

int **printf** (constant char * restrict *format*, ...)

printf output synchronization

When the event associated with a particular kernel invocation completes, the output of applicable **printf** calls is flushed to the implementation-defined output stream.

printf format string

%[*flags*][*width*][.*precision*][*vector*][*length*] *conversion*

Examples:

The following examples show the use of the vector specifier in the **printf** format string.

```
float4  f = float4(1.0f, 2.0f, 3.0f, 4.0f);
uchar4 uc = uchar4(0xFA, 0xFB, 0xFC, 0xFD);
printf("f4 = %2.2v4hlf\n", f);
printf("uc = %#v4hhx\n", uc);
```

The above two printf calls print the following:

```
f4 = 1.00,2.00,3.00,4.00
uc = 0xfa,0xfb,0xfc,0xfd
```

Limits [3.26]

Header <*opencl_limits*>

Half is available if *cl_khr_fp16* is enabled, and double is available if *cl_khr_fp64* is enabled.

Floating point limits

OpenCL C++ Macros (*x* is HALF, FLT, DBL)	HALF	FLT	DBL	Application Macro (*x* is HALF, FLT, DBL)
*x*_DIG	3	6	15	CL_*x*_DIG
*x*_MANT_DIG	11	24	53	CL_*x*_MANT_DIG
*x*_MAX_10_EXP +4	+4	+38	+308	CL_*x*_MAX_10_EXP
*x*_MAX_EXP	+16	+128	+1024	CL_*x*_MAX_EXP
*x*_MIN_10_EX	-4	-37	-307	CL_*x*_MIN_10_EXP
*x*_MIN_EXP	-13	-125	-1021	CL_*x*_MIN_EXP
*x*_RADIX	2	2	2	CL_*x*_RADIX
*x*_MAX	0x1.ffcp15h	0x1.fffffep127f	0x1.fffffffffffffp1023	CL_*x*_MAX
*x*_MIN	0x1.0p-14h	0x1.0p-126f	0x1.0p-1022	CL_*x*_MIN
*x*_EPSILON	0x1.0p-10h	0x1.0p-23f	0x1.0p-52	CL_*x*_EPSILON

enum float_round_style
round_indeterminate, round_toward_zero, round_to_nearest, round_toward_infinity, round_toward_neg_infinity

enum float_denorm_style
denorm_indeterminate, denorm_absent, denorm_present

Integer limits

```
#define CHAR_BIT 8
#define CHAR_MAX SCHAR_MAX
#define CHAR_MIN SCHAR_MIN
#define INT_MAX 2147483647
#define INT_MIN (-2147483647 – 1)
#define LONG_MAX 0x7fffffffffffffffL
#define LONG_MIN (-0x7fffffffffffffffL – 1)
#define SCHAR_MAX 127
#define SCHAR_MIN (-127 – 1)
#define SHRT_MAX 32767
#define SHRT_MIN (-32767 – 1)
#define UCHAR_MAX 255
#define USHRT_MAX 65535
#define UINT_MAX 0xffffffff
#define ULONG_MAX 0xffffffffffffffffUL
```

(Continued on next page >)

Limits (continued)

Class numeric limits [3.26.2]

template<class T> class numeric_limits;

All the members below are declared as static constexpr.

bool is_specialized = false;
T **min**() noexcept { return T(); }
T **max**() noexcept { return T(); }
T **lowest**() noexcept { return T(); }
int digits = 0;
int digits10 = 0;
int max_digits10 = 0;
bool is_signed = false;
bool is_integer = false;
bool is_exact = false;
int radix = 0;
T **epsilon**() noexcept {return T()};
T **round_error**() noexcept { return T(); }
int min_exponent = 0;
int min_exponent10 = 0;
int max_exponent = 0;
int max_exponent10 = 0;
bool has_infinity = false;
bool has_quiet_NaN = false;
bool has_signaling_NaN = false;
float_denorm_style has_denorm = denorm_absent;
bool has_denorm_loss = false;
T **infinity**() noexcept { return T(); }
T **quiet_NaN**() noexcept { return T(); }
T **signaling_NaN**() noexcept { return T(); }
T **denorm_min**() noexcept { return T(); }
bool is_iec559 = false;
bool is_bounded = false;
bool is_modulo = false;
bool traps = false;
bool tinyness_before = false;
float_round_style round_style = round_toward_zero;
bool is_scalar = false;
bool is_vector = false;

Non-members

template<class T> class numeric_limits<const T>;
template<class T> class numeric_limits<volatile T>;
template<class T> class numeric_limits< const volatile T>;

Math Constants [3.27]

Header <*opencl_math_constants*>

The values of the following symbolic constants are single-precision float.

MAXFLOAT	Value of maximum non-infinite single-precision floating-point number
HUGE_VALF	Positive float expression, evaluates to +infinity
HUGE_VAL	Positive double expression, evals. to +infinity
INFINITY	Constant float expression, positive or unsigned infinity
NAN	Constant float expression, quiet NaN

template<class T> class math_constants;
template<> class math_constants<half*n*>;
template<> class math_constants<float*n*>;
template<> class math_constants<double*n*>;

Tuple Library [3.28]

Header <*opencl_tuple*>

template <class... Types> class tuple;

Tuple creation functions	make_tuple() tie()	forward_as_tuple() tuple_cat()	
Tuple helper classes	class tuple_size	class tuple_element	
Element access	get()		
Relational operators	operator==() operator>()	operator<() operator<=()	operator!=() operator>=()
Specialized algorithms	swap()		

Constants, functions, and macros

The preprocessor macros in the table below are shown for double and are available if *cl_khr_fp64* is enabled. Append _F for float, or append _H for half if *cl_khr_fp16* is enabled.

Name of constant	FunctName	Preprocessor macros
e	e()	M_E
log2e	log2e()	M_LOG2E
log10e	log10e()	M_LOG10E
ln2	ln2()	M_LN2
ln10	ln10()	M_LN10
pi	pi()	M_PI
pi_2	pi_2()	M_PI_2
pi_4	pi_4()	M_PI_4
one_pi	one_pi()	M_1_PI
two_pi	two_pi()	M_2_PI
two_sqrtpi	two_sqrtpi()	M_2_SQRTPI
sqrt2	sqrt2()	M_SQRT2
sqrt1_2	sqrt1_2()	M_SQRT1_2

Replace the placeholders in the templates below with values from the indicated column in the table above.

template<class T> constexpr T ***Constant***_v = math_constants<T>::***FunctName***;

template<class T> class math_constants;
static constexpr T ***FunctName*** noexcept { return T(); }

Examples:

template<class T> constexpr T **pi**_v = math_constants<T>::**pi()**;

template<class T> class math_constants;
static constexpr T **pi()** noexcept { return T(); }

Type Traits Library [3.29]

Header <*opencl_type_traits*>

template <class... Types> class tuple;

Primary type categories

is_void, is_null_pointer, is_integral, is_floating_point, is_array, is_pointer, is_enum, is_union, is_class, is_function, is_lvalue_reference, is_rvalue_reference, is_member_object_pointer, is_member_function_pointer

Composite type categories

is_reference, is_arithmetic, is_object, is_ fundamental, is_scalar, is_compound, is_member_pointer

Type property queries

alignment_of, rank, extent

Type relations

is_same, is_base_of, is_convertible

Const-volatile modifications

remove_const, add_const, remove_volatile, add_volatile, remove_cv, add_cv

As modifications

remove_as, remove_attrs, add_constant, remove_constant, add_local, remove_local, add_global, remove_global, add_private, remove_private, add_generic, remove_generic

Reference modifications

remove_reference, add_lvalue_reference, add_rvalue_reference

Sign modifications

make_signed, make_unsigned

Array modifications

remove_extent, remove_all_extents

Pointer modifications

add_pointer, remove_pointer

Built-in vector queries

vector_size, is_vector_type

Built-in vector modifications

vector_element, make_vector

Other transformations

aligned_storage, aligned_union, decay, enable_if, common_type, underlying_type, conditional, result_of

Type properties

is_const, is_volatile, is_private, is_local, is_global, is_constant, is_generic, is_vector, is_trivial, is_trivially_copyable, is_pod, is_literal_type, is_empty, is_polymorphic, is_abstract, is_final, is_signed, is_unsigned, is_standard_layout, is_[trivially_]constructible, is_[trivially_]default_constructible, is_[[trivially_]copy_]constructible, is_[[trivially_]move_]constructible, is_[trivially_]assignable, is_[[trivially_]copy_]assignable, is_[[trivially_]move_]assignable, is_[trivially_, nothrow_]destructible, is_nothrow_[default_]constructible, is_nothrow_[copy_, move_]constructible, is_nothrow_[copy_, move_]assignable, has_virtual_destructor

Iterator Library [3.30]

Header <*opencl_iterator*>

template<class Category, class T,
class Distance = ptrdiff_t,
class Pointer = T*,
class Reference = T&> struct iterator;

Iterator operations

advance() **distance**()
next() **prev**()

Tags

input_iterator_tag
output_iterator_tag
forward_iterator_tag
bidirectional_iterator_tag
random_access_iterator_tag

Range access

begin() **cbegin**() **rbegin**() **crbegin**()
end() **cend**() **rend**() **crend**()

Predefined iterators

inserter() **front_inserter**() **back_inserter**()
make_reverse_iterator()
make_move_iterator()
operator*OP*() where *OP* may be ==, !=, <, > <=, >=, +, -

Vector Wrapper Library [3.7]

Header <*opencl_vec*>

template<class T, size_t Size> struct vec;

struct vec members

vec( ) = default;
vec(const vec &) = default;
vec(vec &&) = default;
vec(const vector_type &*r*) noexcept;
vec(vector_type &&*r*) noexcept;
template <class... Params>
vec(Params... params) noexcept;

operator **vector_type**() const noexcept;
operator*OP*() where *OP* may be =, ++, --, +=, -=, *=, /=, %=

swizzle()

Simple swizzles

If preprocessor macro SIMPLE_SWIZZLES is defined, then:

auto ***func***() noexcept; where *func* may be x through zzzz

Non-member operators

operator*OP*() where *OP* may be ==, !=, <, > <=, >=, +, -, *, /

Vector Utilities [3.9]

Header <*opencl_vector_utility*>

template <size_t Channel, class Vec>
constexpr remove_attrs_t <vector_element_t<Vec>>
get(Vec & *vector*) noexcept;

template <size_t Channel, class Vec>
constexpr void **set**(Vec & *vector*, remove_attrs_t<vector_element_t<Vec>> *value*) noexcept;

struct channel_ref members

operator*OP*() where *OP* may be =, ++, --, +=, -=, *=, /=, %=

OpenCL C Language Reference

Section and table references are to the OpenCL 2.0 C Language specification.

Supported Data Types

Half vector and scalar types require *cl_khr_fp16*. Double types require that CL_DEVICE_DOUBLE_FP_CONFIG is not zero.

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
double	cl_double	64-bit IEEE 754
half	cl_half	16-bit float (storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	32- or 64-bit signed integer
uintptr_t	--	32- or 64-bit unsigned integer
void	void	void

Built-in Vector Data Types [6.1.2]

n is 2, 3, 4, 8, or 16.

OpenCL Type	API Type	Description
[u]char*n*	cl_[u]char*n*	8-bit [un]signed
[u]short*n*	cl_[u]short*n*	16-bit [un]signed
[u]int*n*	cl_[u]int*n*	32-bit [un]signed
[u]long*n*	cl_[u]long*n*	64-bit [un]signed
float*n*	cl_float*n*	32-bit float
double*n*	cl_double*n*	64-bit float

Other Built-in Data Types [6.1.3]

The OPTIONAL types shown below are only defined if CL_DEVICE_IMAGE_SUPPORT is CL_TRUE. API type for application shown in italics where applicable. Items in blue require the *cl_khr_gl_msaa_sharing* extension.

OpenCL Type		Description
image2d_[msaa_]t	OPTIONAL	2D image handle
image3d_t	OPTIONAL	3D image handle
image2d_array_ [msaa_]t	OPTIONAL	2D image array
image1d_t	OPTIONAL	1D image handle
image1d_buffer_t	OPTIONAL	1D image buffer
image1d_array_t	OPTIONAL	1D image array
image2d_ [msaa_]depth_t	OPTIONAL	2D depth image
image2d_array_ [msaa_]depth_t	OPTIONAL	2D depth image array
sampler_t	OPTIONAL	sampler handle
queue_t		
ndrange_t		
clk_event_t		
reserve_id_t		
event_t		event handle
cl_mem_fence_flags		

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool*n*	boolean vector
half*n*	16-bit, vector
quad, quad*n*	128-bit float, vector
complex half, complex half*n* imaginary half, imaginary half*n*	16-bit complex, vector
complex float, complex float*n* imaginary float, imaginary float*n*	32-bit complex, vector
complex double, complex double*n* imaginary double, imaginary double*n*	64-bit complex, vector
complex quad, complex quad*n* imaginary quad, imaginary quad*n*	128-bit complex, vector
float*n*x*m*	*n***m* matrix of 32-bit floats
double*n*x*m*	*n***m* matrix of 64-bit floats

Vector Component Addressing [6.1.7]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalences

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3 *	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz

	v.lo	v.hi	v.odd	v.even
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Operators and Qualifiers

Operators [6.3]

These operators behave similarly as in C99 except operands may include vector types when possible:

+	-	*	%	/	--
++	==	!=	&	~	^
>	<	>=	<=	|	!
&&	||	?:	>>	<<	=
,	*op=*	*sizeof*			

Address Space Qualifiers [6.5]

__global, global __local, local
__constant, constant __private, private

Function Qualifiers [6.7]

__kernel, kernel
__attribute__((vec_type_hint(*type*)))
//type defaults to int
__attribute__((work_group_size_hint(*X*, *Y*, *Z*)))
__attribute__((reqd_work_group_size(*X*, *Y*, *Z*)))

Preprocessor Directives & Macros [6.10]

#pragma OPENCL FP_CONTRACT *on-off-switch*
on-off-switch: ON, OFF, DEFAULT

__FILE__	Current source file
__func__	Current function name
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number, e.g: 200
__CL_VERSION_1_0	Substitutes integer 100 for 1.0
__CL_VERSION_1_1	Substitutes integer 110 for 1.1
__CL_VERSION_1_2	Substitutes integer 120 for 1.2
__CL_VERSION_2_0	Substitutes integer 200 for 2.0
__OPENCL_C_VERSION__	Sub. integer for OpenCL C version
__ENDIAN_LITTLE__	1 if device is little endian
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if –cl-fast-relaxed-math optimization option is specified
FP_FAST_FMA	Defined if double **fma** is fast
FP_FAST_FMAF	Defined if float **fma** is fast
FP_FAST_FMA_HALF	Defined if half **fma** is fast

__kernel_exec (X, *typen*) Same as:
__kernel __attribute__((work_group_size_hint(*X*, 1, 1)))
__attribute__((vec_type_hint(*typen*)))

Conversions, Type Casting Examples [6.2]

T a = (*T*)*b*; *// Scalar to scalar, or scalar to vector*
T a = convert_*T*(*b*);
T a = convert_*T_R*(*b*);
T a = as_*T*(*b*);
T a = convert_*T*_sat_*R*(*b*);

R: one of the rounding modes
_rte to nearest even
_rtz toward zero
_rtp toward + infinity
_rtn toward - infinity

Attribute Qualifiers [6.11]

Use to specify special attributes of enum, struct, and union types.

__attribute__((aligned(*n*))) __attribute__((endian(host)))
__attribute__((aligned)) __attribute__((endian(device)))
__attribute__((packed)) __attribute__((endian))

Use to specify special attributes of variables or structure fields.

__attribute__((aligned(*alignment*)))
__attribute__((nosvm))

Use to specify basic blocks and control-flow-statements.

__attribute__((attr1)) {...}

Use to specify that a loop (for, while, and do loops) can be unrolled. (Must appear immediately before the loop to be affected.)

__attribute__((opencl_unroll_hint(*n*)))
__attribute__((opencl_unroll_hint))

Access Qualifiers [6.6]

Apply to 2D and 3D image types to declare if the image memory object is being read or written by a kernel.

__read_only, read_only __write_only, write_only
__read_write, read_write

Blocks [6.12]

A result value type with a list of parameter types: for example:

1. The ^ declares variable "myBlock" is a Block.
2. The return type for the Block "myBlock"is int.
3. myBlock takes a single argument of type int.
4. The argument is named "num."
5. Multiplier captured from block's environment.

```
②   ①        ③
int (^myBlock)(int) =
 ^(int num) {return num * multiplier; };
                ④        ⑤
```

Work-Item Built-in Functions [6.13.1]

Query the number of dimensions, global, and local work size specified to **clEnqueueNDRangeKernel**, and global and local identifier of each work-item when this kernel is executed on a device.

uint **get_work_dim** ()	Number of dimensions in use
size_t **get_global_size** (uint *dimindx*)	Number of global work-items
size_t **get_global_id** (uint *dimindx*)	Global work-item ID value
size_t **get_local_size** (uint *dimindx*)	Number of local work-items if kernel executed with uniform work-group size
size_t **get_enqueued_local_size** (uint *dimindx*)	Number of local work-items
size_t **get_local_id** (uint *dimindx*)	Local work-item ID
size_t **get_num_groups** (uint *dimindx*)	Number of work-groups
size_t **get_group_id** (uint *dimindx*)	Work-group ID
size_t **get_global_offset** (uint *dimindx*)	Global offset
size_t **get_global_linear_id** ()	Work-items 1-dimensional global ID
size_t **get_local_linear_id** ()	Work-items 1-dimensional local ID
uint **get_sub_group_size** ()	Number of work-items in the subgroup
uint **get_max_sub_group_size** ()	Maximum size of a subgroup
uint **get_num_sub_groups** ()	Number of subgroups
uint **get_enqueued_num_sub_groups** ()	
uint **get_sub_group_id** ()	Sub-group ID
uint **get_sub_group_local_id** ()	Unique work-item ID

Math Built-in Functions [6.13.2]

Ts is type float, optionally double (if *cl_khr_fp64* is enabled), or half (if *cl_khr_fp16* is enabled). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*. All angles are in radians.

HN indicates that half and native variants are available using only the float or float*n* types by prepending "half_" or "native_" to the function name. Prototypes shown in brown text are available in half_ and native_ forms only using the float or float*n* types.

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*y*, *x*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) **HN**	Cosine
T **cosh** (*T*)	Hyperbolic cosine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may only be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Calculates error function of *T*
T **exp** (*T x*) **HN**	Exponential base e
T **exp2** (*T*) **HN**	Exponential base 2
T **exp10** (*T*) **HN**	Exponential base 10
T **expm1** (*T x*)	e^x -1.0
T **fabs** (*T*)	Absolute value
T **fdim** (*T x*, *T y*)	Positive difference between *x* and *y*
T **floor** (*T*)	Round to integer toward infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) *Tn* **fmax** (*Tn x*, *Ts y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) *Tn* **fmin** (*Tn x*, *Ts y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *T* **iptr*)	Fractional value in *x*
Ts **frexp** (*T x*, int **exp*) *Tn* **frexp** (*T x*, int*n* **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of $x^2 + y^2$
int[*n*] **ilogb** (*T x*)	Return exponent as an integer value
Ts **ldexp** (*T x*, int *n*) *Tn* **ldexp** (*T x*, int*n n*)	*x* * 2^n
T **lgamma** (*T x*) *Ts* **lgamma_r** (*Ts x*, int **signp*) *Tn* **lgamma_r** (*Tn x*, int*n* **signp*)	Log gamma function
T **log** (*T*) **HN**	Natural logarithm
T **log2** (*T*) **HN**	Base 2 logarithm
T **log10** (*T*) **HN**	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*
T **minmag** (*T x*, *T y*)	Minimum magnitude of *x* and *y*
T **modf** (*T x*, *T* **iptr*)	Decompose floating-point number
float[*n*] **nan** (uint[*n*] *nancode*)	Quiet NaN (Return is scalar when *nancode* is scalar)
half[*n*] **nan** (ushort[*n*] *nancode*) double[*n*] **nan** (ulong[*n*] *nancode*)	Quiet NaN (Return is scalar when *nancode* is scalar)
T **nextafter** (*T x*, *T y*)	Next representable floating-point value after *x* in the direction of *y*
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y*
Ts **pown** (*T x*, int *y*) *Tn* **pown** (*T x*, int*n y*)	Compute x^y, where *y* is an integer
T **powr** (*T x*, *T y*) **HN**	Compute x^y, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x* (*T* may only be float or float*n*)
T **remainder** (*T x*, *T y*)	Floating point remainder
Ts **remquo** (*Ts x*, *Ts y*, int **quo*) *Tn* **remquo** (*Tn x*, *Tn y*, int*n* **quo*)	Remainder and quotient
T **rint** (*T*)	Round to nearest even integer
Ts **rootn** (*T x*, int *y*) *Tn* **rootn** (*T x*, int*n y*)	Compute *x* to the power of 1/*y*
T **round** (*T x*)	Integral value nearest to *x* rounding
T **rsqrt** (*T*) **HN**	Inverse square root
T **sin** (*T*) **HN**	Sine
T **sincos** (*T x*, *T* **cosval*)	Sine and cosine of *x*
T **sinh** (*T*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **sqrt** (*T*) **HN**	Square root
T **tan** (*T*) **HN**	Tangent
T **tanh** (*T*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)
T **tgamma** (*T*)	Gamma function
T **trunc** (*T*)	Round to integer toward zero

Math Constants [6.13.2]

The values of the following symbolic constants are single-precision float.

MAXFLOAT	Value of maximum non-infinite single-precision floating-point number
HUGE_VALF	Positive float expression, evaluates to +infinity
HUGE_VAL	Positive double expression, evals. to +infinity OPTIONAL
INFINITY	Constant float expression, positive or unsigned infinity
NAN	Constant float expression, quiet NaN

When double precision is supported (if *cl_khr_fp64* is enabled) macros ending in _F are available in type double by removing _F from the macro name, and in type half (if *cl_khr_fp16* is enabled) by replacing _F with _H.

M_E_F	Value of e
M_LOG2E_F	Value of $\log_2 e$
M_LOG10E_F	Value of $\log_{10} e$
M_LN2_F	Value of $\log_e 2$
M_LN10_F	Value of $\log_e 10$
M_PI_F	Value of π
M_PI_2_F	Value of π / 2
M_PI_4_F	Value of π / 4
M_1_PI_F	Value of 1 / π
M_2_PI_F	Value of 2 / π
M_2_SQRTPI_F	Value of 2 / √π
M_SQRT2_F	Value of √2
M_SQRT1_2_F	Value of 1 / √2

Image Read and Write Functions [6.13.14]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage**. *sampler* specifies the addressing and filtering mode to use. *aQual* refers to one of the access qualifiers. For samplerless read functions this may be read_only or read_write.

- Writes to images with sRGB channel orders requires device support of the *cl_khr_srgb_image_writes* extension.
- **read_imageh** and **write_imageh** require the *cl_khr_fp16* extension.
- MSAA images require the *cl_khr_gl_msaa_sharing* extension.
- Image 3D writes require the extension *cl_khr_3d_image_writes*. [9.4.8]

Read and write functions for 2D images

Read an element from a 2D image, or write a color value to a location in a 2D image.

float4 **read_imagef** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

int4 **read_imagei** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

uint4 **read_imageui** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float4 **read_imagef** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

int4 **read_imagei** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

uint4 **read_imageui** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float **read_imagef** (read_only image2d_depth_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float **read_imagef** (read_only image2d_array_depth_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float4 **read_imagef** (*aQual* image2d_t *image*, int2 *coord*)

int4 **read_imagei** (*aQual* image2d_t *image*, int2 *coord*)

uint4 **read_imageui** (*aQual* image2d_t *image*, int2 *coord*)

float4 **read_imagef** (*aQual* image2d_array_t *image*, int4 *coord*)

int4 **read_imagei** (*aQual* image2d_array_t *image*, int4 *coord*)

uint4 **read_imageui** (*aQual* image2d_array_t *image*, int4 *coord*)

float **read_imagef** (*aQual* image2d_depth_t *image*, int2 *coord*)

float **read_imagef** (*aQual* image2d_array_depth_t *image*, int4 *coord*)

half4 **read_imageh** (read_only image2d_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

half4 **read_imageh** (*aQual* image2d_t *image*, int2 *coord*)

half4 **read_imageh** (read_only image2d_array_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

half4 **read_imageh** (*aQual* image2d_array_t *image*, int4 *coord*)

void **write_imagef** (*aQual* image2d_t *image*, int2 *coord*, float4 *color*)

void **write_imagei** (*aQual* image2d_t *image*, int2 *coord*, int4 *color*)

void **write_imageui** (*aQual* image2d_t *image*, int2 *coord*, uint4 *color*)

void **write_imageh** (*aQual* image2d_t *image*, int2 *coord*, half4 *color*)

void **write_imagef** (*aQual* image2d_array_t *image*, int4 *coord*, float4 *color*)

void **write_imagei** (*aQual* image2d_array_t *image*, int4 *coord*, int4 *color*)

void **write_imageui** (*aQual* image2d_array_t *image*, int4 *coord*, uint4 *color*)

void **write_imagef** (*aQual* image2d_depth_t *image*, int2 *coord*, float *depth*)

void **write_imagef** (*aQual* image2d_array_depth_t *image*, int4 *coord*, float *depth*)

void **write_imageh** (*aQual* image2d_array_t *image*, int4 *coord*, half4 *color*)

Read and write functions for 1D images

Read an element from a 1D image, or write a color value to a location in a 1D image.

float4 **read_imagef** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

int4 **read_imagei** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

uint4 **read_imageui** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

float4 **read_imagef** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float4} *coord*)

int4 **read_imagei** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

uint4 **read_imageui** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float2} *coord*)

float4 **read_imagef** (*aQual* image1d_t *image*, int *coord*)

float4 **read_imagef** (*aQual* image1d_buffer_t *image*, int *coord*)

int4 **read_imagei** (*aQual* image1d_t *image*, int *coord*)

uint4 **read_imageui** (*aQual* image1d_t *image*, int *coord*)

int4 **read_imagei** (*aQual* image1d_buffer_t *image*, int *coord*)

uint4 **read_imageui** (*aQual* image1d_buffer_t *image*, int *coord*)

float4 **read_imagef** (*aQual* image1d_array_t *image*, int2 *coord*)

int4 **read_imagei** (*aQual* image1d_array_t *image*, int2 *coord*)

uint4 **read_imageui** (*aQual* image1d_array_t *image*, int2 *coord*)

half4 **read_imageh** (read_only image1d_t *image*, sampler_t *sampler*, {int, float} *coord*)

half4 **read_imageh** (*aQual* image1d_t *image*, int *coord*)

half4 **read_imageh** (read_only image1d_array_t *image*, sampler_t *sampler*, {int2, float4} *coord*)

half4 **read_imageh** (*aQual* image1d_array_t *image*, int2 *coord*)

half4 **read_imageh** (*aQual* image1d_buffer_t *image*, int *coord*)

void **write_imagef** (*aQual* image1d_t *image*, int *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_t *image*, int *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_t *image*, int *coord*, uint4 *color*)

void **write_imageh** (*aQual* image1d_t *image*, int *coord*, half4 *color*)

void **write_imagef** (*aQual* image1d_buffer_t *image*, int *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_buffer_t *image*, int *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_buffer_t *image*, int *coord*, uint4 *color*)

void **write_imageh** (*aQual* image1d_buffer_t *image*, int *coord*, half4 *color*)

void **write_imagef** (*aQual* image1d_array_t *image*, int2 *coord*, float4 *color*)

void **write_imagei** (*aQual* image1d_array_t *image*, int2 *coord*, int4 *color*)

void **write_imageui** (*aQual* image1d_array_t *image*, int2 *coord*, uint4 *color*)

void **write_imageh** (*aQual* image1d_array_t *image*, int2 *coord*, half4 *color*)

Read and write functions for 3D images

Read an element from a 3D image, or write a color value to a location in a 3D image. Writing to 3D images requires the *cl_khr_3d_image_writes* extension [9.4.8].

float4 **read_imagef** (read_only image3d_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

int4 **read_imagei** (read_only image3d_t *image*, sampler_t *sampler*, int4 *coord*)

int4 **read_imagei** (read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*)

uint4 **read_imageui** (read_only image3d_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

float4 **read_imagef** (*aQual* image3d_t *image*, int4 *coord*)

int4 **read_imagei** (*aQual* image3d_t *image*, int4 *coord*)

uint4 **read_imageui** (*aQual* image3d_t *image*, int4 *coord*)

half4 **read_imageh** (read_only image3d_t *image*, sampler_t *sampler*, {int4, float4} *coord*)

half4 **read_imageh** (*aQual* image3d_t *image*, int4 *coord*)

void **write_imagef** (*aQual* image3d_t *image*, int4 *coord*, float4 *color*)

void **write_imagei** (*aQual* image3d_t *image*, int4 *coord*, int4 *color*)

void **write_imageui** (*aQual* image3d_t *image*, int4 *coord*, uint4 *color*)

void **write_imageh** (*aQual* image3d_t *image*, int4 *coord*, half4 *color*)

Extended mipmap read and write functions

These functions require the *cl_khr_mipmap_image* and *cl_khr_mipmap_image_writes* extensions.

float **read_imagef** (read_only image2d_[depth_]t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

int4 **read_imagei** (read_only image2d_t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

uint4 **read_imageui** (read_only image2d_t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

float **read_imagef** (read_only image2d_ [depth_]t *image*, sampler_t *sampler*, float2 *coord*, float2 *gradient_x*, float2 *gradient_y*)

int4 **read_imagei** (read_only image2d_t *image*, sampler_t *sampler*, float2 *coord*, float2 *gradient_x*, float2 *gradient_y*)

uint4 **read_imageui** (read_only image2d_t *image*, sampler_t *sampler*, float2 *coord*, float2 *gradient_x*, float2 *gradient_y*)

float4 **read_imagef** (read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *lod*)

int4 **read_imagei** (read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *lod*)

uint4 **read_imageui**(read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *lod*)

float4 **read_imagef** (read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *gradient_x*, float *gradient_y*)

int4 **read_imagei** (read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *gradient_x*, float *gradient_y*)

uint4 **read_imageui**(read_only image1d_t *image*, sampler_t *sampler*, float *coord*, float *gradient_x*, float *gradient_y*)

float4 **read_imagef** (read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

int4 **read_imagei**(read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

uint4 **read_imageui**(read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

float4 **read_imagef** (read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float4 *gradient_x*, float4 *gradient_y*)

(Continued on next page >)

Image Read and Write (continued)

Extended mipmap read and write functions (cont'd)

int4 **read_imagei**(read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float4 *gradient_x*, float4 *gradient_y*)

uint4 **read_imageui**(read_only image3d_t *image*, sampler_t *sampler*, float4 *coord*, float4 *gradient_x*, float4 *gradient_y*)

float4 **read_imagef** (read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

int4 **read_imagei** (read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

uint4 **read_imageui**(read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *lod*)

float4 **read_imagef** (read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *gradient_x*, float *gradient_y*)

int4 **read_imagei** (read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *gradient_x*, float *gradient_y*)

uint4 **read_imageui**(read_only image1d_array_t *image*, sampler_t *sampler*, float2 *coord*, float *gradient_x*, float *gradient_y*)

float **read_imagef** (read_only image2d_array_ [depth_]t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

int4 **read_imagei** (read_only image2d_array_t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

uint4 **read_imageui** (read_only image2d_array_t *image*, sampler_t *sampler*, float4 *coord*, float *lod*)

float **read_imagef** (read_only image2d_array_ [depth_]t *image*, sampler_t *sampler*, float4 *coord*, float2 *gradient_x*, float2 *gradient_y*)

int4 **read_imagei** (read_only image2d_array_t *image*, sampler_t *sampler*, float4 *coord*, float2 *gradient_x*, float2 *gradient_y*)

uint4 **read_imageui** (read_only image2d_array_t *image*, sampler_t *sampler*, float4 *coord*, float2 *gradient_x*, float2 *gradient_y*)

void **write_imagef** (*aQual* image2d_ [depth_]t *image*, int2 *coord*, int *lod*, float4 *color*)

void **write_imagei** (*aQual* image2d_t *image*, int2 *coord*, int *lod*, int4 *color*)

void **write_imageui** (*aQual* image2d_t *image*, int2 *coord*, int *lod*, uint4 *color*)

void **write_imagef** (*aQual* image1d_t *image*, int *coord*, int *lod*, float4 *color*)

void **write_imagei** (*aQual* image1d_t *image*, int *coord*, int *lod*, int4 *color*)

void **write_imageui** (*aQual* image1d_t *image*, int *coord*, int *lod*, uint4 *color*)

void **write_imagef** (*aQual* image1d_array_t *image*, int2 *coord*, int *lod*, float4 *color*)

void **write_imagei** (*aQual* image1d_array_t *image*, int2 *coord*, int *lod*, int4 *color*)

void **write_imageui** (*aQual* image1d_array_t *image*, int2 *coord*, int *lod*, uint4 *color*)

void **write_imagef** (*aQual* image2d_array_ [depth_]t *image*, int4 *coord*, int *lod*, float4 *color*)

void **write_imagei** (*aQual* image2d_array_t *image*, int4 *coord*, int *lod*, int4 *color*)

void **write_imageui** (*aQual* image2d_array_t *image*, int4 *coord*, int *lod*, uint4 *color*)

void **write_imagef** (*aQual* image3d_t *image*, int4 *coord*, int *lod*, float4 *coord*)

void **write_imagei** (*aQual* image3d_t *image*, int4 *coord*, int *lod*, int4 *color*)

void **write_imageui** (*aQual* image3d_t *image*, int4 *coord*, int *lod*, uint4 *color*)

Extended multi-sample image read functions [9.10.3]

The extension *cl_khr_gl_msaa_sharing* adds the following built-in functions.

float **read_imagef** (*aQual* image2d_msaa_depth_t *image*, int2 *coord*, int *sample*)

float **read_imagef** (*aQual* image2d_array_depth_msaa_t *image*, int4 *coord*, int *sample*)

float4 **read_image{f, i, ui}** (image2d_msaa_t *image*, int2 *coord*, int *sample*)

float4 **read_image{f, i, ui}** (image2d_array_msaa_t *image*, int4 *coord*, int *sample*)

Image Query Functions [6.13.14.5] [9.10.3]

The MSAA forms require the extension *cl_khr_gl_msaa_sharing*. Mipmap requires the extension *cl_khr_mipmap_image*.

Query image width, height, and depth in pixels

int **get_image_width** (*aQual* image{1,2,3}d_t *image*)

int **get_image_width** (*aQual* image1d_buffer_t *image*)

int **get_image_width** (*aQual* image{1,2}d_array_t *image*)

int **get_image_width** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_width** (*aQual* image2d_[array_]msaa_t *image*)

int **get_image_width** (*aQual* image2d_ [array_]msaa_depth_t *image*)

int **get_image_height** (*aQual* image{2,3}d_t *image*)

int **get_image_height** (*aQual* image2d_array_t *image*)

int **get_image_height** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_height** (*aQual* image2d_[array_]msaa_t *image*)

int **get_image_height** (*aQual* image2d_[array_]msaa_depth_t *image*)

int **get_image_depth** (image3d_t *image*)

Query image array size

size_t **get_image_array_size** (*aQual* image1d_array_t *image*)

size_t **get_image_array_size** (*aQual* image2d_array_t *image*)

size_t **get_image_array_size** (*aQual* image2d_array_depth_t *image*)

size_t **get_image_array_size** (*aQual* image2d_array_msaa_depth_t *image*)

Query image dimensions

int2 **get_image_dim** (*aQual* image2d_t *image*)

int2 **get_image_dim** (*aQual* image2d_array_t *image*)

int4 **get_image_dim** (*aQual* image3d_t *image*)

int2 **get_image_dim** (*aQual* image2d_[array_]depth_t *image*)

int2 **get_image_dim** (*aQual* image2d_[array_]msaa_t *image*)

int2 **get_image_dim** (*aQual* image2d_ [array_]msaa_depth_t *image*)

Query image channel data type and order

int **get_image_channel_data_type** (*aQual* image{1,2,3}d_t *image*)

int **get_image_channel_data_type** (*aQual* image1d_buffer_t *image*)

int **get_image_channel_data_type** (*aQual* image{1,2}d_array_t *image*)

int **get_image_channel_data_type** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_channel_data_type** (*aQual* image2d_[array_]msaa_t *image*)

int **get_image_channel_data_type** (*aQual* image2d_[array_]msaa_depth_t *image*)

int **get_image_channel_order** (*aQual* image{1,2,3}d_t *image*)

int **get_image_channel_order** (*aQual* image1d_buffer_t *image*)

int **get_image_channel_order** (*aQual* image{1,2}d_array_t *image*)

int **get_image_channel_order** (*aQual* image2d_[array_]depth_t *image*)

int **get_image_channel_order** (*aQual* image2d_[array_]msaa_t *image*)

int **get_image_channel_order**(*aQual* image2d_[array_]msaa_depth_t *image*)

Extended query functions [9.10.3]

These functions require the cl_khr_mipmap_image extension.

int **get_image_num_mip_levels** (*aQual* image1d_t *image*)

int **get_image_num_mip_levels** (*aQual* image2d_ [depth_]t *image*)

int **get_image_num_mip_levels** (*aQual* image3d_t *image*)

int **get_image_num_mip_levels** (*aQual* image1d_array_t *image*)

int **get_image_num_mip_levels** (*aQual* image2d_array_[depth_]t *image*)

int **get_image_num_samples** (*aQual* image2d_[array_]msaa_t *image*)

int **get_image_num_samples** (*aQual* image2d_ [array_]msaa_depth_t *image*)

Notes

Integer Built-in Functions [6.13.3]

T is type char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, or ulongn, where *n* is 2, 3, 4, 8, or 16. *Tu* is the unsigned version of *T*. *Tsc* is the scalar version of *T*.

Tu **abs** (*T x*)	| *x* |
Tu **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *Tsc min*, *Tsc max*)	min(max(*x*, *minval*), *maxval*)
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **ctz** (*T x*)	Number of trailing 0-bits in *x*
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *Tsc y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*) *T* **min** (*T x*, *Tsc y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	High half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result
T **popcount** (*T x*)	Number of non-zero bits in *x*

*For **upsample**, return type is scalar when the parameters are scalar.*

short[*n*] **upsample** (char[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort[*n*] **upsample** (uchar[*n*] *hi*, uchar[*n*] *lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int[*n*] **upsample** (short[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint[*n*] **upsample** (ushort[*n*] *hi*, ushort[*n*] *lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long[*n*] **upsample** (int[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong[*n*] **upsample** (uint[*n*] *hi*, uint[*n*] *lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

The following fast integer functions optimize the performance of kernels. In these functions, *T* is type int, uint, intn, or uintn, where *n is 2, 3, 4, 8,* or *16*.

T **mad24** (*T x*, *T y*, *T z*)	Multiply 24-bit integer values *x*, *y*, add 32-bit int. result to 32-bit integer *z*
T **mul24** (*T x*, *T y*)	Multiply 24-bit integer values *x* and *y*

Common Built-in Functions [6.13.4]

These functions operate component-wise and use round to nearest even rounding mode. *Ts* is type float, optionally double (if *cl_khr_fp64* is enabled) or half and halfn (if *cl_khr_fp16* is enabled). *Tn* is the vector form of *Ts*, where *n* is 2, 3, 4, 8, or 16. *T* is *Ts* and *Tn*.

T **clamp** (*T x*, *T min*, *T max*) *Tn* **clamp** (*Tn x*, *Ts min*, *Ts max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) *Tn* **max** (*Tn x*, *Ts y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) *Tn* **min** (*Tn x*, *Ts y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) *Tn* **mix** (*Tn x*, *Tn y*, *Ts a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*) *Tn* **step** (*Ts edge*, *Tn x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*) *T* **smoothstep** (*Ts edge0*, *Ts edge1*, *T x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

Relational Built-in Functions [6.13.6]

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result. *T* is type float, floatn, char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, ulongn, or optionally double or doublen (if *cl_khr_fp64* is enabled) or half and halfn (if *cl_khr_fp16* is enabled). *Ti* is type char, charn, short, shortn, int, intn, long, or longn. *Tu* is type uchar, ucharn, ushort, ushortn, uint, uintn, ulong, or ulongn. *n* is 2, 3, 4, 8, or 16.

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*)	Compare of *x* >= *y*
long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*)	Compare of *x* < *y*
long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*) int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Compare of *x* <= *y*
int **islessgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*) int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*) int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value
int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	Test for + or – infinity
int **isnan** (float) int*n* **isnan** (float*n*)	Test for a NaN
int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double)	Test for a normal value
long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a normal value
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*) int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	Test if arguments are ordered
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*) int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*)	Test if arguments are unordered
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*) int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign bit
int **any** (*Ti x*)	1 if MSB in component of x is set; else 0
int **all** (*Ti x*)	1 if MSB in all components of x are set; else 0
T **bitselect** (*T a*, *T b*, *T c*) half **bitselect** (half *a*, half *b*, half *c*) half*n* **bitselect** (half*n* *a*, half*n* *b*, half*n* *c*)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
T **select** (*T a*, *T b*, *Ti c*) *T* **select** (*T a*, *T b*, *Tu c*) half*n* **select** (half*n* *a*, half*n* *b*, short*n* *c*) half **select** (half *a*, half *b*, short *c*) half*n* **select** (half*n* *a*, half*n* *b*, ushort*n* *c*) half **select** (half *a*, half *b*, ushort *c*)	For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i] For scalar type, result = c ? b : a

Geometric Built-in Functions [6.13.5]

Ts is scalar type float, optionally double (if *cl_khr_fp64* is enabled), or half (if *cl_khr_fp16* is enabled). ***T*** is ***Ts*** and the 2-, 3-, or 4-component vector forms of ***Ts***.

float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product
Ts **distance** (*T p0*, *T p1*)	Vector distance
Ts **dot** (*T p0*, *T p1*)	Dot product
Ts **length** (*T p*)	Vector length
T **normalize** (*T p*)	Normal vector length 1
float **fast_distance** (float *p0*, float *p1*) float **fast_distance** (float*n* *p0*, float*n* *p1*)	Vector distance
float **fast_length** (float *p*) float **fast_length** (float*n* *p*)	Vector length
float **fast_normalize** (float *p*) float*n* **fast_normalize** (float*n* *p*)	Normal vector length 1

Vector Data Load/Store [6.13.7]

T is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double (if *cl_khr_fp64* is enabled), or half (if *cl_khr_fp16* is enabled). *Tn* refers to the vector form of type *T*, where *n* is 2, 3, 4, 8, or 16. *R* defaults to current rounding mode, or is one of the rounding modes listed in 6.2.3.2.

Tn **vloadn** (size_t *offset*, const [constant] *T* *p)	Read vector data from address (*p* + (*offset* * *n*))
void **vstoren** (*Tn data*, size_t *offset*, *T* *p)	Write vector data to address (*p* + (*offset* * *n*)
float **vload_half** (size_t *offset*, const [constant] half *p)	Read a half from address (*p* + *offset*)
float*n* **vload_half*n*** (size_t *offset*, const [constant] half *p)	Read a half*n* from address (*p* + (*offset* * *n*))

void **vstore_half** (float *data*, size_t *offset*, half *p) void **vstore_half_*R*** (float *data*, size_t *offset*, half *p) void **vstore_half** (double *data*, size_t *offset*, half *p)	Write a half to address (*p* + *offset*)
void **vstore_half_*R*** (double *data*, size_t *offset*, half *p)	Write a half to address (*p* + *offset*)
void **vstore_half*n*** (float*n* *data*, size_t *offset*, half *p) void **vstore_half*n*_*R*** (float*n* *data*, size_t *offset*, half *p) void **vstore_half*n*** (double*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))

void **vstore_half*n*_*R*** (double*n* *data*, size_t *offset*, half *p)	Write a half vector to address (*p* + (*offset* * *n*))
float*n* **vloada_half*n*** (size_t *offset*, const [constant] half *p)	Read half vector data from (*p* + (*offset* * *n*)). For half3, read from (*p* + (*offset* * 4)).
void **vstorea_half*n*** (float*n* *data*, size_t *offset*, half *p) void **vstorea_half*n*_*R*** (float*n* *data*, size_t *offset*, half *p) void **vstorea_half*n*** (double*n* *data*, size_t *offset*, half *p) void **vstorea_half*n*_*R*** (double*n* *data*, size_t *offset*, half *p)	Write half vector data to (*p* + (*offset* * *n*)). For half3, write to (*p* + (*offset* * 4)).

Synchronization & Memory Fence Functions [6.13.8]

flags argument is the memory address space, set to a 0 or an OR'd combination of CLK_*X*_MEM_FENCE where *X* may be LOCAL, GLOBAL, or IMAGE. Memory fence functions provide ordering between memory operations of a work-item.

void **work_group_barrier** (cl_mem_fence_flags *flags* [, memory_scope *scope*])	Work-items in a work-group must execute this before any can continue
void **atomic_work_item_fence** (cl_mem_fence_flags *flags* [, memory_scope *scope*])	Orders loads and stores of a work-item executing a kernel
void **sub_group_barrier** (cl_mem_fence_flags *flags* [, memory_scope *scope*])	Work-items in a sub-group must execute this before any can continue

Async Copies and Prefetch [6.13.10]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, optionally double or double*n* (if *cl_khr_fp64* is enabled), or half or half*n* (if *cl_khr_fp16* is enabled).

event_t **async_work_group_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global *T* *dst, const __local *T* *src, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_strided_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global *T* *dst, const __local *T* *src, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	Copies *num_gentypes* *T* elements from *src* to *dst*
void **wait_group_events** (int *num_events*, event_t *event_list*)	Wait for completion of **async_work_group_copy**
void **prefetch** (const __global *T* *p, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(*T*) bytes into global cache

Atomic Functions [6.13.11]

OpenCL C implements a subset of the C11 atomics (see section 7.17 of the C11 specification) and synchronization operations.

In the following tables, **A** refers to an atomic_* type (not including atomic_flag). **C** refers to its corresponding non-atomic type. **M** refers to the type of the other argument for arithmetic operations. For atomic integer types, **M** is **C**. For atomic pointer types, **M** is ptrdiff_t.

The type atomic_* is a 32-bit integer. atomic_long and atomic_ulong require extension *cl_khr_int64_base_atomics* or *cl_khr_int64_extended_atomics*. The atomic_double type is available if *cl_khr_fp64* is enabled. The default scope is work_group for local atomics and all_svm_devices for global atomics. The extensions *cl_khr_int64_base_atomics* and *cl_khr_int64_extended_atomics* implement atomic operations on 64-bit signed and unsigned integers to locations in __global and __local memory.

See the table under Atomic Types and Enum Constants for information about parameter types memory_order, memory_scope, and memory_flag.

void **atomic_init**(volatile A *obj, C *value*)	Initializes the atomic object pointed to by *obj* to the value *value*.
void **atomic_work_item_fence**(cl_mem_fence_flags *flags*, memory_order *order*, memory_scope *scope*)	Effects based on value of *order*. *flags* must be CLK_{GLOBAL, LOCAL, IMAGE}_MEM_FENCE or a combination of these.
void **atomic_store**(volatile A *object, C *desired*) void **atomic_store_explicit**(volatile A *object, C *desired*, memory_order *order* [, memory_scope *scope*])	Atomically replace the value pointed to by *object* with the value of *desired*. Memory is affected according to the value of *order*.
C **atomic_load**(volatile A *object) C **atomic_load_explicit**(volatile A *object, memory_order *order*[, memory_scope *scope*])	Atomically returns the value pointed to by *object*. Memory is affected according to the value of *order*.
C **atomic_exchange**(volatile A *object, C *desired*) C **atomic_exchange_explicit**(volatile A *object, C *desired*, memory_order *order* [, memory_scope *scope*])	Atomically replace the value pointed to by *object* with *desired*. Memory is affected according to the value of *order*.
bool **atomic_compare_exchange_strong**(volatile A *object, C *expected*, C *desired*) bool **atomic_compare_exchange_strong_explicit**(volatile A *object, C *expected*, C *desired*, memory_order *success*, memory_order *failure*[, memory_scope *scope*]) bool **atomic_compare_exchange_weak**(volatile A *object, C *expected*, C *desired*) bool **atomic_compare_exchange_weak_explicit**(volatile A *object, C *expected*, C *desired*, memory_order *success*, memory_order *failure*[, memory_scope *scope*])	Atomically compares the value pointed to by *object* for equality with that in *expected*, and if true, replaces the value pointed to by *object* with *desired*, and if false, updates the value in *expected* with the value pointed to by *object*. These operations are atomic read-modify-write operations.
C **atomic_fetch_<*key*>**(volatile A *object, M *operand*) C **atomic_fetch_<*key*>_explicit**(volatile A *object, M *operand*, memory_order *order* [, memory_scope *scope*])	Atomically replaces the value pointed to by *object* with the result of the computation applied to the value pointed to by *object* and the given *operand*.
bool **atomic_flag_test_and_set**(volatile atomic_flag *object) bool **atomic_flag_test_and_set_explicit**(volatile atomic_flag *object, memory_order *order*[, memory_scope *scope*])	Atomically sets the value pointed to by *object* to true. Memory is affected according to the value of *order*. Returns atomically, the value of the object immediately before the effects.
void **atomic_flag_clear**(volatile atomic_flag *object) void **atomic_flag_clear_explicit**(volatile atomic_flag *object, memory_order *order*[, memory_scope *scope*])	Atomically sets the value pointed to by *object* to false. The order argument shall not be *memory_order_acquire* nor *memory_order_acq_rel*. Memory is affected according to the value of *order*.

Values for *key* for atomic_fetch and modify functions

key	op	computation
add	+	addition
sub	-	subtraction
or	|	bitwise inclusive or
xor	^	bitwise exclusive or

key	op	computation
and	&	bitwise and
min	min	compute min
max	max	compute max

Atomic Types and Enum Constants

Parameter Type	Values
memory_order	memory_order_relaxed memory_order_acquire memory_order_release memory_order_ acq_rel memory_order_seq_cst
memory_scope	memory_scope_work_item memory_scope_work_group memory_scope_sub_group memory_scope_all_svm_devices memory_scope_device (default for functions that do not take a memory_scope argument)

Atomic integer and floating-point types

† indicates types supported by a limited subset of atomic operations

‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture.

§ indicates types supported only if extensions *cl_khr_int64_base_atomics* and *cl_khr_int64_extended_atomics* are enabled.

atomic_int	atomic_long §	atomic_float †	atomic_intptr_t ‡§	atomic_size_t ‡§
atomic_uint	atomic_ulong §	atomic_double †§	atomic_uintptr_t ‡§	atomic_ptrdiff_t ‡§
atomic_flag				

Atomic macros

#define ATOMIC_VAR_INIT(C value)	Expands to a token sequence to initialize an atomic object of a type that is initialization-compatible with value.
#define ATOMIC_FLAG_INIT	Initialize an atomic_flag to the clear state.

Address Space Qualifier Functions [6.13.9]

T refers to any of the built-in data types supported by OpenCL C or a user-defined type.

[const] global *T* * **to_global** ([const] *T* *ptr)	global address space
[const] local *T* * **to_local** ([const] *T* *ptr)	local address space
[const] private *T* * **to_private** ([const] *T* *ptr)	private address space
[const] cl_mem_fence_flags **get_fence(** [const] *T* *ptr)	Memory fence value: CLK_GLOBAL_MEM_FENCE, CLK_LOCAL_MEM_FENCE

printf Function [6.13.13]

Writes output to an implementation-defined stream.

int **printf** (constant char * restrict *format*, ...)

printf output synchronization

When the event associated with a particular kernel invocation completes, the output of applicable **printf** calls is flushed to the implementation-defined output stream.

printf format string

The format string follows C99 conventions and supports an optional vector specifier:

%[flags][width][.precision][vector][length] conversion

Examples:

The following examples show the use of the vector specifier in the **printf** format string.

```
float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);
uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);
printf("f4 = %2.2v4hlf\n", f);
printf("uc = %#v4hhx\n", uc);
```

The above two printf calls print the following:

```
f4 = 1.00,2.00,3.00,4.00
uc = 0xfa,0xfb,0xfc,0xfd
```

Miscellaneous Vector Functions [6.13.12]

Tm and *Tn* are type char*n*, uchar*n*, short*n*, ushort*n*, int*n*, uint*n*, long*n*, ulong*n*, float*n*, optionally double*n* (if *cl_khr_fp64* is enabled) or half*n* (if *cl_khr_fp16* is enabled), where *n* is 2,4,8, or 16 except in **vec_step** it may also be 3. *TUn* is uchar*n*, ushort*n*, uint*n*, or ulong*n*.

int **vec_step** (*Tn a*) int **vec_step** (*typename*)	Takes built-in scalar or vector data type argument. Returns 1 for scalar, 4 for 3-component vector, else number of elements in the specified type.
Tn **shuffle** (*Tm x, TUn mask*) *Tn* **shuffle2** (*Tm x, Tm y, TUn mask*)	Construct permutation of elements from one or two input vectors, return a vector with same element type as input and length that is the same as the shuffle mask.

Workgroup Functions [6.13.15]

T is type int, uint, long, ulong, or float, optionally double (if *cl_khr_fp64* is enabled) or half (if *cl_khr_fp64* is enabled).

Returns a non-zero value if *predicate* evaluates to non-zero for all or any workitems in the work-group.

int **work_group_all** (int *predicate*)
int **work_group_any** (int *predicate*)
int **sub_group_all** (int *predicate*)
int **sub_group_any** (int *predicate*)

Return result of reduction operation specified by <*op*> for all values of *x* specified by workitems in work-group. <*op*> may be min, max, or add.

T **work_group_reduce_<*op*>** (*T x*)
T **sub_group_reduce_<op>** (*T x*)

Broadcast the value of *a* to all work-items in the work-group. *local_id* must be the same value for all workitems in the work-group.

T **work_group_broadcast** (*T* a, size_t *local_id*)
T **work_group_broadcast** (*T* a, size_t *local_id_x*, size_t *local_id_y*)
T **work_group_broadcast** (*T* a, size_t *local_id_x*, size_t *local_id_y*, size_t *local_id_z*)
T **sub_group_broadcast** (*T* x, size_t *local_id*)

Do an exclusive or inclusive scan operation specified by <*op*> of all values specified by work-items in the work-group. The scan results are returned for each work-item. <*op*> may be min, max, or add.

T **work_group_scan_exclusive_<op>** (*T x*)
T **work_group_scan_inclusive_<op>** (*T x*)
T **sub_group_scan_exclusive_<op>** (*T x*)
T **sub_group_scan_inclusive_<op>** (*T x*)

Pipe Built-in Functions [6.13.16.2-4]

T represents the built-in OpenCL C scalar or vector integer or floating-point data types or any user defined type built from these scalar and vector data types. Half scalar and vector types require the *cl_khr_fp16* extension. Double or vector double types require the *cl_khr_fp64* extension. The macro CLK_NULL_RESERVE_ID refers to an invalid reservation ID.

int **read_pipe** (__read_only pipe *T p*, *T* *ptr)	Read packet from *p* into *ptr*.
int **read_pipe** (__read_only pipe *T p*, reserve_id_t *reserve_id*, uint *index*, *T* *ptr)	Read packet from reserved area of the pipe *reserve_id* and *index* into *ptr*.
int **write_pipe** (__write_only pipe *T p*, const *T* *ptr)	Write packet specified by *ptr* to *p*.
int **write_pipe** (__write_only pipe *T p*, reserve_id_t *reserve_id*, uint *index*, const *T* *ptr)	Write packet specified by *ptr* to reserved area *reserve_id* and *index*.
bool **is_valid_reserve_id** (reserve_id_t *reserve_id*)	Return true if *reserve_id* is a valid reservation ID and false otherwise.

reserve_id_t **reserve_read_pipe** (__read_only pipe *T p*, uint *num_packets*) reserve_id_t **reserve_write_pipe** (__write_only pipe *T p*, uint *num_packets*)	Reserve *num_packets* entries for reading from or writing to *p*.
void **commit_read_pipe** (__read_only pipe *T p*, reserve_id_t *reserve_id*) void **commit_write_pipe** (__write_only pipe *T p*, reserve_id_t *reserve_id*)	Indicates that all reads and writes to *num_packets* associated with reservation *reserve_id* are completed.
uint **get_pipe_max_packets** (pipe *T p*)	Returns maximum number of packets specified when *p* was created.
uint **get_pipe_num_packets** (pipe *T p*)	Returns the number of available entries in *p*.

void **work_group_commit_read_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **work_group_commit_write_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **sub_group_commit_read_pipe** (pipe *T p*, reserve_id_t *reserve_id*) void **sub_group_commit_write_pipe** (pipe *T p*, reserve_id_t *reserve_id*)	Indicates that all reads and writes to *num_packets* associated with reservation *reserve_id* are completed.
reserve_id_t **work_group_reserve_read_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **work_group_reserve_write_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **sub_group_reserve_read_pipe** (pipe *T p*, uint *num_packets*) reserve_id_t **sub_group_reserve_write_pipe** (pipe *T p*, uint *num_packets*)	Reserve *num_packets* entries for reading from or writing to *p*. Returns a valid reservation ID if the reservation is successful.

Enqueuing and Kernel Query Built-in Functions [6.13.17]

A kernel may enqueue code represented by Block syntax, and control execution order with event dependencies including user events and markers. There are several advantages to using the Block syntax: it is more compact; it does not require a cl_kernel object; and enqueuing can be done as a single semantic step. The macro CLK_NULL_EVENT refers to an invalid device event. The macro CLK_NULL_QUEUE refers to an invalid device queue.

int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, void (^*block*)(void)) int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, uint *num_events_in_wait_list*, const clk_event_t *event_wait_list, clk_event_t *event_ret, void (^*block*)(void)) int **enqueue_kernel** (queue_t *queue*, kernel_enqueue_flags_t *flags*, const ndrange_t *ndrange*, void (^*block*)(local void *, ...), uint *size0*, ...) int **enqueue_kernel** (queue_t *queue*, kernel_ enqueue_flags_t *flags*, const ndrange_t *ndrange*, uint *num_events_in_wait_list*, const clk_event_t *event_wait_list, clk_event_t *event_ret, void (^*block*)(local void *, ...), uint *size0*, ...)	Allows a work-item to enqueue a block for execution to *queue*. Work-items can enqueue multiple blocks to a device queue(s). *flags* may be one of CLK_ENQUEUE_FLAGS_{NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP}

uint **get_kernel_work_group_size** (void (^*block*)(void)) uint **get_kernel_work_group_size** (void (^*block*)(local void *, ...))	Query the maximum work-group size that can be used to execute a block.
uint **get_kernel_preferred_work_group_size_multiple** (void (^*block*)(void)) uint **get_kernel_preferred_work_group_size_multiple** (void (^*block*)(local void *, ...))	Returns the preferred multiple of work-group size for launch.
int **enqueue_marker** (queue_t *queue*, uint *num_events_in_wait_list*, const clk_event_t *event_wait_list, clk_event_t *event_ret)	Enqueue a marker command to *queue*.
uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(void)) uint **get_kernel_sub_group_count_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(local void *, ...))	Returns number of subgroups in each workgroup of the dispatch.
uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(void)) uint **get_kernel_max_sub_group_size_for_ndrange** (const ndrange_t *ndrange*, void (^*block*)(local void *, ...))	Returns the maximum sub-group size for a block.

Event Built-in Functions [6.13.17.8]

T is type int, uint, long, ulong, or float, optionally double (if *cl_khr_fp64* is enabled), or half (if *cl_khr_fp16* is enabled).

void **retain_event** (clk_event_t *event*)	Increments event reference count.
void **release_event** (clk_event_t *event*)	Decrements event reference count.
clk_event_t **create_user_event** ()	Create a user event.
bool **is_valid_event** (clk_event_t *event*)	True for valid event.
void **set_user_event_status** (clk_event_t *event*, int *status*)	Sets the execution status of a user event. *status*: CL_COMPLETE or a negative error value.
void **capture_event_profiling_info** (clk_event_t *event*, clk_profiling_info *name*, global void **value*)	Captures profiling information for command associated with *event* in value.

Helper Built-in Functions [6.13.17.9]

queue_t **get_default_queue** (void)	Default queue or CLK_NULL_QUEUE
ndrange_t **ndrange_1D** (size_t *global_work_size*) ndrange_t **ndrange_1D** (size_t *global_work_size*, size_t *local_work_size*) ndrange_t **ndrange_1D** (size_t *global_work_offset*, size_t *global_work_size*, size_t *local_work_size*)	Builds a 1D ND-range descriptor.
ndrange_t **ndrange_*n*D** (const size_t *global_work_size*[*n*]) ndrange_t **ndrange_*n*D** (size_t *global_work_size*, const size_t *local_work_size*[*n*]) ndrange_t **ndrange_*n*D** (const size_t *global_work_offset*, const size_t *global_work_size*, const size_t *local_work_size*[*n*])	Builds a 2D or 3D ND-range descriptor. *n* may be 2 or 3.

Notes

OpenCL Extensions Reference

Section and table references are to the OpenCL Extensions 2.1 specification.

Using OpenCL Extensions [9]

In this section, extensions shown in *italics* provide core features.

#pragma OPENCL EXTENSION *extension_name* : {enable |disable}

To test if an extension is supported, use **clGetPlatformInfo()** or **clGetDeviceInfo()**

To get the address of the extension function: **clGetExtensionFunctionAddressForPlatform()**

cl_apple_gl_sharing (see cl_khr_gl_sharing)
cl_khr_3d_image_writes
cl_khr_byte_addressable_store
cl_khr_context_abort
cl_khr_d3d10_sharing
cl_khr_d3d11_sharing
cl_khr_depth_images
cl_khr_device_enqueue_local_arg_types
cl_khr_dx9_media_sharing
cl_khr_egl_event
cl_khr_egl_image
cl_khr_fp16
cl_khr_fp64
cl_khr_gl_depth_images
cl_khr_gl_event
cl_khr_gl_msaa_sharing
cl_khr_gl_sharing
cl_khr_global_int32_base_atomics - atomic_*()
cl_khr_global_int32_extended_atomics - atomic_*()
cl_khr_icd
cl_khr_image2d_from_buffer
cl_khr_initialize_memory
cl_khr_int64_base_atomics - atom_*()
cl_khr_int64_extended_atomics - atom_*()
cl_khr_local_int32_base_atomics - atomic_*()
cl_khr_local_int32_extended_atomics - atomic_*()
cl_khr_mipmap_image
cl_khr_mipmap_image_writes
cl_khr_priority_hints
cl_khr_srgb_image_writes
cl_khr_spir
cl_khr_subgroup_named_barrier
cl_khr_terminate_context
cl_khr_throttle_hints

OpenGL, OpenGL ES Sharing [9.3 - 9.5]

These functions require the *cl_khr_gl_sharing* or *cl_apple_gl_sharing* extension.

CL Context > GL Context, Sharegroup

cl_int **clGetGLContextInfoKHR** (
const cl_context_properties *properties,
cl_gl_context_info *param_name*,
size_t *param_value_size*, void *param_value,
size_t *param_value_size_ret*)

param_name: CL_DEVICES_FOR_GL_CONTEXT_KHR, CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR

CL Buffer Objects > GL Buffer Objects

cl_mem **clCreateFromGLBuffer** (cl_context *context*, cl_mem_flags *flags*, GLuint *bufobj*, cl_int *errcode_ret*)

flags: CL_MEM_{READ_ONLY, WRITE_ONLY, READ_WRITE}

CL Image Objects > GL Textures

cl_mem **clCreateFromGLTexture** (cl_context *context*, cl_mem_flags *flags*, GLenum *texture_target*, GLint *miplevel*, GLuint *texture*, cl_int *errcode_ret*)

flags: See **clCreateFromGLBuffer**

texture_target: GL_TEXTURE_{1D, 2D}[_ARRAY], GL_TEXTURE_{3D, BUFFER, RECTANGLE}, GL_TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, GL_TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}, GL_TEXTURE_2D_MULTISAMPLE[_ARRAY] (Requires extension *cl_khr_gl_msaa_sharing*)

CL Image Objects > GL Renderbuffers

cl_mem **clCreateFromGLRenderbuffer** (
cl_context *context*, cl_mem_flags *flags*,
GLuint *renderbuffer*, cl_int *errcode_ret*)

flags: See **clCreateFromGLBuffer**

Query Information

cl_int **clGetGLObjectInfo** (cl_mem *memobj*,
cl_gl_object_type *gl_object_type*,
GLuint *gl_object_name*)

gl_object_type returns:
CL_GL_OBJECT_TEXTURE_BUFFER,
CL_GL_OBJECT_TEXTURE{1D, 2D, 3D},
CL_GL_OBJECT_TEXTURE{1D, 2D}_ARRAY,
CL_GL_OBJECT_{BUFFER, RENDERBUFFER}

cl_int **clGetGLTextureInfo** (cl_mem *memobj*,
cl_gl_texture_info *param_name*,
size_t *param_value_size*, void *param_value,
size_t *param_value_size_ret*)

param_name: CL_GL_{TEXTURE_TARGET, MIPMAP_LEVEL}, CL_GL_NUM_SAMPLES (Requires extension *cl_khr_gl_msaa_sharing*)

Share Objects

cl_int **clEnqueue{Acquire, Release}GLObjects** (
cl_command_queue *command_queue*,
cl_uint *num_objects*, const cl_mem *mem_objects,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

CL Event Objects > GL Sync Objects

cl_event **clCreateEventFromGLsyncKHR** (
cl_context *context*, GLsync *sync*, cl_int *errcode_ret*)

Requires the *cl_khr_gl_event* extension.

Direct3D 10 Sharing [9.6.7]

These functions require the *cl_khr_d3d10_sharing* extension. The associated header file is <*cl_d3d10.h*>.

cl_int **clGetDeviceIDsFromD3D10KHR** (
cl_platform_id *platform*,
cl_d3d10_device_source_khr *d3d_device_source*,
void *d3d_object*,
cl_d3d10_device_set_khr *d3d_device_set*,
cl_uint *num_entries*, cl_device_id *devices*,
cl_uint *num_devices*)

d3d_device_source:
CL_D3D10_{DEVICE, DXGI_ADAPTER}_KHR

d3d_device_set:
CL_{ALL, PREFERRED}_DEVICES_FOR_D3D10_KHR

cl_mem **clCreateFromD3D10BufferKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D10Buffer *resource*, cl_int *errcode_ret*)

flags: See **clCreateFromGLBuffer**

cl_mem **clCreateFromD3D10Texture2DKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D10Texture2D *resource*, UINT *subresource*,
cl_int *errcode_ret*)

flags: See **clCreateFromD3D10BufferKHR**

cl_mem **clCreateFromD3D10Texture3DKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D10Texture3D *resource*, UINT *subresource*,
cl_int *errcode_ret*)

flags: See **clCreateFromGLBuffer**

cl_int **clEnqueue{Acquire, Release}D3D10ObjectsKHR** (
cl_ command_queue *command_queue*,
cl_uint *num_objects*, const cl_mem *mem_objects,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

DX9 Media Surface Sharing [9.7]

Header <*cl_dx9_media_sharing.h*>
Enable the extension *cl_khr_dx9_media_sharing*.

cl_int **clGetDeviceIDsFromDX9MediaAdapterKHR** (
cl_platform_id *platform*,
cl_uint *num_media_adapters*,
cl_dx9_media_adapter_type_khr*media_adapters_type*,
void *media_adapters*,
cl_dx9_media_adapter_set_khr *media_adapter_set*,
cl_uint *num_entries*, cl_device_id *devices*,
cl_int *num_devices*)

media_adapter_type:
CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

media_adapter_set: CL_{ALL, PREFERRED}_DEVICES_-FOR_DX9_MEDIA_ADAPTER_KHR

cl_mem **clCreateFromDX9MediaSurfaceKHR** (
cl_context *context*, cl_mem_flags *flags*,
cl_dx9_media_adapter_type_khr *adapter_type*,
void *surface_info*, cl_uint *plane*, cl_int *errcode_ret*)

flags: See **clCreateFromGLBuffer**

adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

cl_int
clEnqueue{Acquire, Release}DX9MediaSurfacesKHR(
cl_command_queue *command_queue*,
cl_uint *num_objects*, const cl_mem *mem_objects,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

Direct3D 11 Sharing [9.8.7]

Header <*cl_d3d11.h*> These functions require the *cl_khr_d3d11_sharing* extension. For values of *flags*, see **clCreateFromGLBuffer.**

cl_int **clGetDeviceIDsFromD3D11KHR** (
cl_platform_id *platform*,
cl_d3d11_device_source_khr *d3d_device_source*,
void *d3d_object*,
cl_d3d11_device_set_khr *d3d_device_set*,
cl_uint *num_entries*, cl_device_id *devices*,
cl_uint *num_devices*)

d3d_device_source: CL_D3D11_DEVICE_KHR, CL_D3D11_DXGI_ADAPTER_KHR

d3d_device_set: CL_ALL_DEVICES_FOR_D3D11_KHR, CL_PREFERRED_DEVICES_FOR_D3D11_KHR

cl_mem **clCreateFromD3D11BufferKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D11Buffer *resource*, cl_int *errcode_ret*)

cl_mem **clCreateFromD3D11Texture3DKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D11Texture3D *resource*, UINT *subresource*,
cl_int *errcode_ret*)

cl_mem **clCreateFromD3D11Texture2DKHR** (
cl_context *context*, cl_mem_flags *flags*,
ID3D11Texture2D *resource*,
UINT *subresource*, cl_int *errcode_ret*)

cl_int **clEnqueue{Acquire, Release}D3D11ObjectsKHR** (
cl_command_queue *command_queue*,
cl_uint *num_objects*, const cl_mem *mem_objects,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

EGL Interoperabililty [9.16, 9.17]

Create CL Event Objects from EGL

This function requires the extension *cl_khr_egl_event*.

cl_event **clCreateEventFromEGLsyncKHR** (
cl_context *context*, CLeglSyncKHR *sync*,
CLeglDisplayKHR *display*, cl_int *errcode_ret*)

Create CL Image Objects from EGL

These functions require the extension *cl_khr_egl_image*.

cl_mem **clCreateFromEGLImageKHR** (
cl_context *context*, CLeglDisplayKHR *display*,
CLeglImageKHR *image*, cl_mem_flags *flags*,
const cl_egl_image_properties_khr *properties*,
cl_int *errcode_ret*)

cl_int **clEnqueue{Acquire, Release}EGLObjectsKHR** (
cl_command_queue *command_queue*,
cl_uint *num_objects*, const cl_mem *mem_objects,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*,
cl_event *event*)

OpenCL Extensions

Example of Enqueuing Kernels

Arguments that are a pointer type to local address space [6.13.17.2]

A block passed to enqueue_kernel can have arguments declared to be a pointer to local memory. The enqueue_kernel built-in function variants allow blocks to be enqueued with a variable number of arguments. Each argument must be declared to be a void pointer to local memory. These enqueue_kernel built-in function variants also have a corresponding number of arguments each of type uint that follow the block argument. These arguments specify the size of each local memory pointer argument of the enqueued block.

```
kernel void
my_func_A_local_arg1(global int *a, local int *lptr, ...)
{
   ...
}

kernel void
my_func_A_local_arg2(global int *a,
   local int *lptr1, local float4 *lptr2, ...)
{
   ...
}

kernel void
my_func_B(global int *a, ...)
{
   ...
   ndrange_t ndrange = ndrange_1d(...);
   uint local_mem_size = compute_local_mem_size();
   enqueue_kernel(get_default_queue(),
       CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
       ndrange,
       ^(local void *p){
          my_func_A_local_arg1(a, (local int *)p, ...);},
       local_mem_size);
}

kernel void
my_func_C(global int *a, ...)
{
   ...
   ndrange_t ndrange = ndrange_1d(...);
   void (^my_blk_A)(local void *, local void *) =
       ^(local void *lptr1, local void *lptr2){
          my_func_A_local_arg2(
             a,
             (local int *)lptr1,
             (local float4 *)lptr2, ...);};
   // calculate local memory size for lptr
   // argument in local address space for my_blk_A
   uint local_mem_size = compute_local_mem_size();
   enqueue_kernel(get_default_queue(),
       CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
       ndrange,
       my_blk_A,
       local_mem_size, local_mem_size * 4);
}
```

A Complete Example [6.13.17.3]

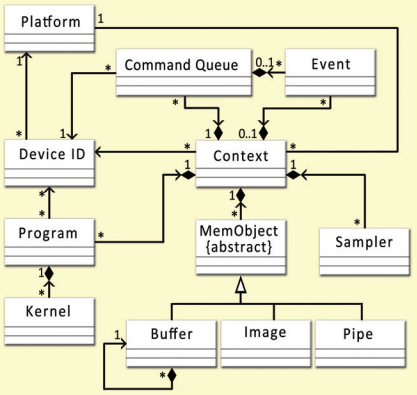
The example below shows how to implement an iterative algorithm where the host enqueues the first instance of the nd-range kernel (dp_func_A). The kernel dp_func_A will launch a kernel (evaluate_dp_work_A) that will determine if new nd-range work needs to be performed. If new nd-range work does need to be performed, then evaluate_dp_work_A will enqueue a new instance of dp_func_A . This process is repeated until all the work is completed.

```
kernel void
dp_func_A(queue_t q, ...)
{
   ...
   // queue a single instance of evaluate_dp_work_A to
   // device queue q. queued kernel begins execution after
   // kernel dp_func_A finishes

   if (get_global_id(0) == 0)
   {
       enqueue_kernel(q,
                      CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                      ndrange_1d(1),
                      ^{evaluate_dp_work_A(q, ...);});
   }
}
kernel void
evaluate_dp_work_A(queue_t q,...)
{
   // check if more work needs to be performed
   bool more_work = check_new_work(...);
   if (more_work)
   {
       size_t global_work_size = compute_global_size(...);
       void (^dp_func_A_blk)(void) =
          ^{dp_func_A(q, ...});

       // get local WG-size for kernel dp_func_A
       size_t local_work_size =
          get_kernel_work_group_size(dp_func_A_blk);

       // build nd-range descriptor
       ndrange_t ndrange = ndrange_1D(global_work_size,
                                      local_work_size);

       // enqueue dp_func_A
       enqueue_kernel(q,
                      CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                      ndrange,
                      dp_func_A_blk);
   }
   ...
}
```

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language[1] (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.

Annotations	
Relationships	
abstract classes	{abstract}
aggregations	♦
inheritance	△
relationship navigability	^
Cardinality	
many	*
one and only one	1
optionally one	0..1
one or more	1..*



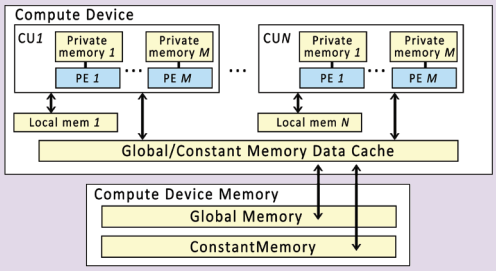
[1] Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

OpenCL Device Architecture Diagram

The table below shows memory regions with allocation and memory access capabilities. R=Read, W=Write

	Global	Constant	Local	Private
Host	Dynamic allocation R/W access	Dynamic allocation R/W access	Dynamic allocation No access	No allocation No access
Kernel	No allocation R/W access	Static allocation R-only access	Static allocation R/W access	Static allocation R/W access

The conceptual OpenCL device architecture diagram shows processing elements (PE), compute units (CU), and devices. The host is not shown.



Notes

OpenCL Reference Card Index

The following index shows the page number for each item included in this guide. The color of the row in the table below is the color of the box to which you should refer.