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.

[n.n.n] and purple text: sections and text in the OpenCL API 2.1 Spec.
[n.n.n] and green text: sections and text in the OpenCL C 2.0 Spec.
[n.n.n] and blue text: sections and text in the OpenCL Extension 2.1 Spec.

OpenCL API Reference

Section and table references are to the OpenCL API 2.1 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 when the appropriate extension is supported.

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 * *platfoms*, 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_*X*
(where *X* may be 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_SUBGROUP_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_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_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, NUM_DEVICES, PROPERTIES}, CL_CONTEXT_{D3D10, 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: [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

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]

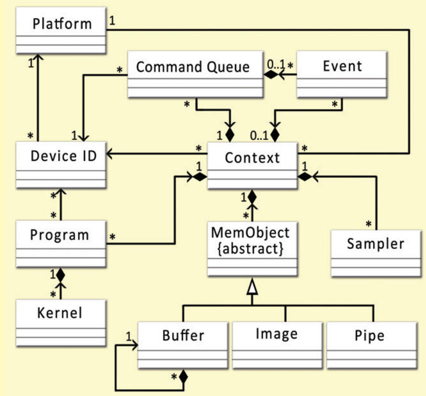
void * **clEnqueueMapBuffer** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_map*, cl_map_flags *map_flags*, size_t *offset*, size_t *size*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*, cl_int **errcode_ret*)

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

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	Δ
relationship navigability	^
Cardinality	
many	*
one and only one	1
optionally one	0..1
one or more	1..*



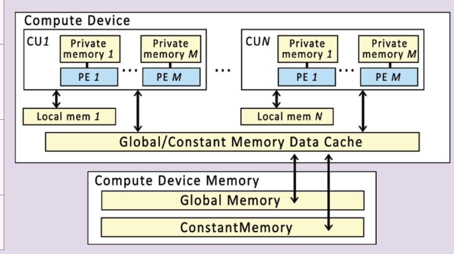
[1] Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).

OpenCL Device Architecture Diagram

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

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



Conversions and 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 following rounding modes:

- _rte to nearest even
- _rtz toward zero
- _rtp toward + infinity
- _rtn toward - infinity

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, CL_MEM_{READ, WRITE}_ONLY, CL_MEM_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

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 supported.

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: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_OFFSET, CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_ASSOCIATED_MEMOBJECT, CL_MEM_CONTEXT, CL_MEM_USES_SVM_POINTER, CL_MEM_{D3D10, D3D11}_RESOURCE_KHR, CL_MEM_DX9_MEDIA_{ADAPTER_TYPE, SURFACE_INFO}_KHR [Table 5.13]

Shared Virtual Memory

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.
See more on SVM on page 4 of this reference guide.

SVM Sharing Granularity [5.6.1]

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 [5.6.2]

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*)

OpenCL API

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*)

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

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

Building Program Executables [5.8.2]

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 [5.8.3]

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 [5.8.6]

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

Query Program Objects [5.8.7]

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_REFERENCE_COUNT,
  CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
  CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES},
  CL_PROGRAM_{NUM_KERNELS, KERNEL_NAMES},
  CL_PROGRAM_IL

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

Compiler Options [5.8.4]

SPIR options require the cl_khr_spir extension.

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 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

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:
  -x spir   // indicate that binary is in SPIR format
  -spir-std=*x*   //*x* is SPIR spec version, e.g.: 1.2

Linker Options [5.8.5]

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

Flush and Finish [5.15]

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

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

Kernel Objects

A kernel object encapsulates the specific __kernel function and the argument values to be used when executing it. Items in blue apply when the appropriate extension is supported.

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,
  CL_KERNEL_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
  [Table 5.21]

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_{ACCESS, ADDRESS}_QUALIFIER,
  CL_KERNEL_ARG_NAME,
  CL_KERNEL_ARG_TYPE_{NAME, QUALIFIER}

(Continued on next page >)

OpenCL API

Kernel Objects (continued)

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

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: Shared Virtual Memory [3.3.3]

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

- **Coarse-Grained buffer SVM**: Sharing occurs at the granularity of regions of OpenCL buffer memory objects. Consistency is enforced at synchronization points and with map/unmap commands to drive updates between the host and the device. This form of SVM is similar to the use of cl_mem buffers, with two differences. First, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Second, concurrent access by multiple kernels on the same device is valid as long as the set of concurrently executing kernels is bounded by synchronization points. Concurrent access by multiple kernels on the same device is valid as long as the set of kernels is bounded by synchronization points. This form of SVM is similar to non-SVM use of memory; however, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Program scope global variables are treated as per-device coarse-grained SVM for addressing and sharing purposes.
- **Fine-Grained buffer SVM**: Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects. Loads and stores may be cached. This means consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.
- **Fine-Grained system SVM**: Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory. Loads and stores may be cached so consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.

Coarse-Grained buffer SVM is required in the core OpenCL specification. The two finer grained approaches are optional features in OpenCL. The various SVM mechanisms to access host memory from the work-items associated with a kernel instance are summarized in table 3-2 below.

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

Notes

OpenCL C Language Reference

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

Supported Data Types

The optional double scalar and vector types are supported if 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 OPTIONAL	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]

OpenCL Type	API Type	Description
char*n*	cl_char*n*	8-bit signed
uchar*n*	cl_uchar*n*	8-bit unsigned
short*n*	cl_short*n*	16-bit signed
ushort*n*	cl_ushort*n*	16-bit unsigned
int*n*	cl_int*n*	32-bit signed
uint*n*	cl_uint*n*	32-bit unsigned
long*n*	cl_long*n*	64-bit signed
ulong*n*	cl_ulong*n*	64-bit unsigned
float*n*	cl_float*n*	32-bit float
double*n* OPTIONAL	cl_double*n*	64-bit float
half*n*	Requires the cl_khr_fp16 extension	

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

OpenCL C Language

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.

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*)))

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*)))

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))

Blocks [6.12]

A result value type with a list of parameter types, similar to a function type. In this 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. Sub-groups require the cl_khr_subgroups extension.

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] [9.4.2]

Ts is type float, optionally double, or half if the cl_khr_fp16 extension 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] [9.4.2]

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, macros ending in _F are available in type double by removing _F from the macro name, and in type half when the cl_khr_fp16 extension 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

Integer Built-in Functions [6.13.3]

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*. *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, int*n* or int*n*,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] [9.4.3]

These functions operate component-wise and use round to nearest even rounding mode. *Ts* is type float, optionally double, 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*) *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*

OpenCL C Language

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, 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*. *Ti* is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. *Tu* is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. *n* is 2, 3, 4, 8, or 16. half and half*n* types require the cl_khr_fp16 extension [9.4.5].

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] [9.4.4]

Ts is scalar type float, optionally double, or half if the half extension 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] [9.4.6]

T is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension 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_halfn** (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_halfn** (float*n* *data*, size_t *offset*, half **p*) void **vstore_halfn_*R*** (float*n* *data*, size_t *offset*, half **p*) void **vstore_halfn** (double*n* *data*, size_t *offset*, half **p*)	Write a half vector to address (*p* + (*offset* * *n*))

void **vstore_halfn_*R*** (double*n* *data*, size_t *offset*, half **p*)	Write a half vector to address (*p* + (*offset* * *n*))
float*n* **vloada_halfn** (size_t *offset*, const [constant] half **p*)	Read half vector data from (*p* + (*offset* * *n*)). For half3, read from (*p* + (*offset* * 4)).
void **vstorea_halfn** (float*n* *data*, size_t *offset*, half **p*) void **vstorea_halfn_*R*** (float*n* *data*, size_t *offset*, half **p*) void **vstorea_halfn** (double*n* *data*, size_t *offset*, half **p*) void **vstorea_halfn_*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. Sub-groups require the cl_khr_subgroups extension.

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] [9.4.7]

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*, or half or half*n* if the cl_khr_fp16 extension 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 requires double precision support. 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*. IThese 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

memory_scope_sub_group requires the cl_khr_subgroups extension.

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 both 64-bit extensions are supported.

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

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);
printf("f4 = %2.2v4f\n", f);
    Output:  f4 = 1.00,2.00,3.00,4.00

uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);
printf("uc = %#v4x\n", uc);
    Output:  uc = 0xfa,0xfb,0xfc,0xfd

uint2 ui = (uint2)(0x12345678, 0x87654321);
printf("unsigned short value = (%#v2hx)\n", ui);
    Output:  unsigned short value = (0x5678,0x4321)
```

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*, or half*n* if the cl_khr_fp16 extension is supported, 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] [9.17.3.4]

T is type int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension is supported. Sub-groups require the cl_khr_subgroups extension. Double and vector types require double precision support.

Returns a non-zero value if *predicate* evaluates to non-zero for all or any workitems in the work-group or sub-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 or sub_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 or sub_group. *local_id* must be the same value for all workitems in the work-group. *n* may be 2 or 3.

- *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 or sub-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. Sub-groups require the cl_khr_subgroups extension. Double or vector double types require double precision support. 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] [9.17.3.6]

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. Sub-groups require the cl_khr_subgroups extension. 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.

OpenCL C Language

Event Built-in Functions [6.13.17.8]

T is type int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension 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_nD** (const size_t *global_work_size*[*n*]) ndrange_t **ndrange_nD** (size_t *global_work_size*, const size_t *local_work_size*[*n*]) ndrange_t **ndrange_nD** (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.

OpenCL Image Processing Reference

A subset of the OpenCL API 2.1 and C Language 2.0 specifications pertaining to image processing and graphics.

OpenCL C Language

OpenCL Image Processing

Image Objects

Items in blue apply when the appropriate extension is supported.

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

Image Formats [5.3.1.1]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support [Table 5.8]

CL_R (read or write): CL_HALF_FLOAT, CL_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_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_FLOAT, CL_UNORM_INT{8,16}, 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)

Optional support [Table 5.6]

CL_R, CL_A (read and write): CL_HALF_FLOAT, CL_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_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_FLOAT, CL_SNORM_INT{8,16}
CL_RG, CL_RA: CL_HALF_FLOAT, CL_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

Notes

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 [9.17.2.1]

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 >)

OpenCL Image Processing

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 [9.12.3]

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] [9.12]

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 [9.18.2.1]

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*)

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

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 can be 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

OpenCL Image Processing

OpenCL Extensions Reference

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

Using OpenCL Extensions [9]

The following extensions extend the OpenCL API. Extensions shown in *italics* provide core features.

To control an extension: #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_subgroups
- cl_khr_terminate_context
- cl_khr_throttle_hints

OpenGL Sharing [9.5 - 9.7]

These functions require the cl_khr_gl_sharing or cl_apple_ gl_sharing extension.

CL Context > GL Context, Sharegroup [9.5.5]

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 [9.6.2]

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 [9.6.3]

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 [9.6.4]

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

flags: See **clCreateFromGLBuffer**

Query Information [9.6.5]

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 [9.6.6]

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 [9.7.4]

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

Requires the cl_khr_gl_event extension.

Direct3D 10 Sharing [9.8.7]

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 [9.9]

The header file is 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 [9.10.7.3 - 9.10.7.6]

These functions require the cl_khr_d3d11_sharing extension. Associated header file is cl_d3d11.h.

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*)

flags: See **clCreateFromGLBuffer**

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

flags: See **clCreateFromGLBuffer**

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

flags: See **clCreateFromGLBuffer**

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 [9.18, 9.19]

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*)

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*)

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);
   }
   ...
}
```

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.

OpenCL

KHRONOS GROUP™

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn more about the Khronos Group.