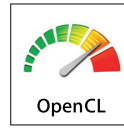


OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance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



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

OpenCL API Reference

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` **clGetPlatformIDsKHR** (`cl_uint num_entries,`
`cl_platform_id *platforms,` `cl_uint *num_platforms`)

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

param_name: CL_PLATFORM_{PROFILE, VERSION},
 CL_PLATFORM_{NAME, VENDOR, EXTENSIONS},
 CL_PLATFORM_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_PREFERRED_{PLATFORM, LOCAL,
 GLOBAL}_ATOMIC_ALIGNMENT,
 CL_DEVICE_GLOBAL_VARIABLE_SHARING,
 CL_DEVICE_HOST_UNIFIED_MEMORY,
 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_READ_IMAGE_ARGS,
 CL_DEVICE_MAX_WRITE_IMAGE_ARGS,
 CL_DEVICE_MAX_{CLOCK_FREQUENCY, PIPE_ARGS},
 CL_DEVICE_MAX_{COMPUTE_UNITS, SAMPLERS},
 CL_DEVICE_MAX_CONSTANT_{ARGS, BUFFER_SIZE},
 CL_DEVICE_MAX_{MEM_ALLOC, PARAMETER}_SIZE,
 CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE,
 CL_DEVICE_MAX_ON_DEVICE_{QUEUES, EVENTS},
 CL_DEVICE_MAX_WORK_GROUP_SIZE,
 CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
 CL_DEVICE_MEM_BASE_ADDR_ALIGN,
 CL_DEVICE_NAME,
 CL_DEVICE_NATIVE_VECTOR_WIDTH_{CHAR, INT},
 CL_DEVICE_NATIVE_VECTOR_WIDTH_{LONG, SHORT},
 CL_DEVICE_NATIVE_VECTOR_WIDTH_{DOUBLE, HALF},
 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_VECTOR_WIDTH_{CHAR, INT},
 CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE,
 CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF,
 CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG,
 CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT,
 CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
 CL_DEVICE_PREFERRED_INTEROP_USER_SYNC,
 CL_DEVICE_PROFILE,
 CL_DEVICE_PROFILING_TIMER_RESOLUTION,
 CL_DEVICE_SPIR_VERSIONS,
 CL_DEVICE_QUEUE_ON_DEVICE_PROPERTIES,
 CL_DEVICE_QUEUE_ON_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, DRIVER}_VERSION

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: 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`)

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_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, CL_QUEUE_SIZE,
 CL_QUEUE_REFERENCE_COUNT,
 CL_QUEUE_PROPERTIES

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`)

Buffer Objects

Elements 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 Buffer Objects [5.2.2]

`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`)

(Continued on next page >)

Buffer Objects (continued)

```
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, CL_MAP_WRITE, CL_MAP_WRITE_INVALIDATE_REGION

Conversions and Type Casting Examples [6.2]

```
T a = (T)b; // Scalar to scalar,
           // or scalar to vector
           _rte to nearest even
           _rtz toward zero
T a = convert_T(b);
           _rtp toward + infinity
T a = convert_T_R(b);
           _rtn toward - infinity
T a = as_T(b);
T a = convert_T_sat_R(b);
R: one of the following rounding modes:
```

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

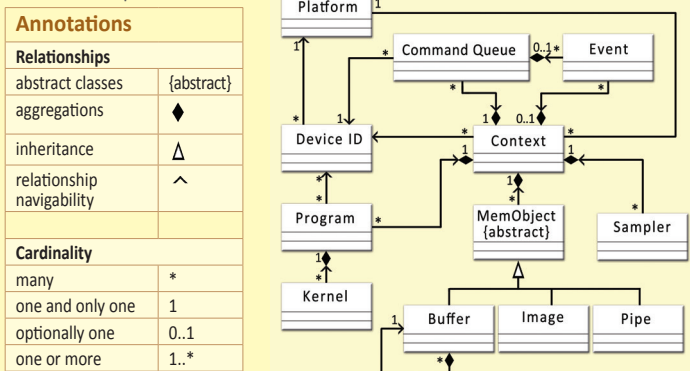
Query Memory Object [5.5.5]

```
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, CL_MEM_FLAGS, CL_MEM_SIZE, CL_MEM_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_RESOURCE_KHR, CL_MEM_D3D11_RESOURCE_KHR, CL_MEM_DX9_MEDIA_ADAPTER_TYPE, CL_MEM_SURFACE_INFO_KHR [Table 5.12]

OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language¹ (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.



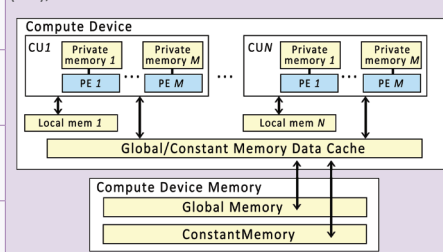
¹ Unified Modeling Language (<http://www.uml.org/>) is a trademark of Object Management Group (OMG).

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

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



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

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.

SVM Sharing Granularity [5.6.1]

```
void * clSVMAlloc (cl_context context, cl_svm_mem_flags flags, size_t size,
    unsigned int alignment)
```

flags: [Table 5.13] 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 *svm_pointers[],
    void (CL_CALLBACK *pfn_free_func)
    (cl_command_queue command_queue,
    cl_uint num_svm_pointers,
    void *svm_pointers[], void *user_data),
    void *user_data, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

(Continued on next page >)

Shared Virtual Memory (continued)

```

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)

```

Kernel Objects

A kernel is a function declared in a program, identified by the `__kernel` qualifier. 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, 5.9.3]

```

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_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 5.19]
             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_{LOCAL, PRIVATE}_MEM_SIZE,
             CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_
             MULTIPLE [Table 5.20]

cl_int clGetKernelArgInfo (cl_kernel kernel,
    cl_uint arg_idx, cl_kernel_arg_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
param_name:
             CL_KERNEL_ARG_{ACCESS, ADDRESS}_QUALIFIER,
             CL_KERNEL_ARG_NAME,
             CL_KERNEL_ARG_TYPE_{NAME, QUALIFIER} [Table 5.21]

```

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 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_int clGetKernelSubGroupInfoKHR

```

(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:
             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)

```

Flush and Finish [5.15]

```

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

```

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_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 5.16]
             CL_PROGRAM_REFERENCE_COUNT,
             CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
             CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES},
             CL_PROGRAM_{NUM_KERNELS, KERNEL_NAMES}

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 5.17]
             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
```

```

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 5.22]

```

```

cl_int clRetainEvent (cl_event event)

```

(Continued on next page >)

Event Objects (continued)

```
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 5.23]
CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_
COMMAND_COMPLETE,
CL_PROFILING_COMMAND_{SUBMIT, START, END}
```

OpenCL C Language Reference

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]

Table with 3 columns: OpenCL Type, API Type, Description. Lists types like bool, char, short, int, float, double, half, size_t, ptrdiff_t, intptr_t, uintptr_t, void.

Built-in Vector Data Types [6.1.2]

Table with 3 columns: OpenCL Type, API Type, Description. Lists vector types like charn, uchar, shortn, ushortn, intn, uintn, longn, ulongn, floatn, doublen, halfn.

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.

Table with 3 columns: OpenCL Type, API Type, Description. Lists image and array types like image2d, image3d, image2d_array, image1d, image1d_buffer.

Table with 3 columns: OpenCL Type, API Type, Description. Lists sampler and queue types like image1d_array, image2d, image2d_array, sampler_t, queue_t, ndranger_t, clk_event_t, reserve_id_t, event_t, cl_mem_fence_flags.

Reserved Data Types [6.1.4]

Table with 2 columns: OpenCL Type, Description. Lists reserved vector types like booln, halfn, quad, complex half, complex float, complex double, complex quad, floatnxm, doublenxm.

Vector Component Addressing [6.1.7]

Vector Components

Table showing vector components for float2, float3, float4, float8, float16 across indices 0-15.

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

Table mapping vector types to their component addressing notations (e.g., float2 to v.lo, v.s0).

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

Preprocessor Directives & Macros [6.10]

Table listing preprocessor directives and macros like #pragma OPENCL FP_CONTRACT, __FILE__, __func__, __LINE__, __OPENCL_VERSION__, __CL_VERSION__, __CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE__, etc.

Operators and Qualifiers

Operators [6.3]

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

Table of arithmetic and logical operators: +, -, *, %, /, --, ++, ==, !=, &, ~, ^, >, <, >=, <=, |, !, &&, ||, ?:, >>, <<, =, , 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)))
```

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.

```
int (^myBlock)(int) =
^(int num) {return num * multiplier;};
```

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.

Table of work-item built-in functions: uint get_work_dim(), size_t get_global_size(), size_t get_global_id(), size_t get_local_size().

(Continued on next page >)

Work-Item Functions (continued)

Table with 2 columns: Function Name and Description. Includes size_t_get_enqueued_local_size, size_t_get_local_id, size_t_get_num_groups, size_t_get_group_id, size_t_get_global_offset.

Table with 2 columns: Function Name and Description. Includes size_t_get_global_linear_id, size_t_get_local_linear_id, uint_get_sub_group_size, uint_get_max_sub_group_size, uint_get_num_sub_groups, uint_get_enqueued_num_sub_groups, uint_get_sub_group_id, uint_get_sub_group_local_id.

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))

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.

HN indicates that half and native variants are available using only the float or floatn 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 floatn types.

Table with 2 columns: Function Name and Description. Includes T acos, T acosh, T acospi, T asin, T asinh, T asinpi, T atan, T atan2, T atanh, T atanpi, T atan2pi, T cbrt, T ceil, T copysign, T cos, T cosh, T cospi, T half_divide, T native_divide, T erfc, T erf, T exp, T exp2, T exp10.

Table with 2 columns: Function Name and Description. Includes T expm1, T fabs, T fdim, T floor, T fma, T fmax, T fmax, T fmin, T fmin, T fmod, T fract, Ts frexp, Tn frexp, T hypot, int[n] ilogb, Ts ldexp, Tn ldexp, T lgamma, Ts lgamma_r, Tn lgamma_r, T log, T log2, T log10, T log1p, T logb, T mad, T maxmag, T minmag, T modf, float[n] nan, half[n] nan, double[n] nan.

Table with 2 columns: Function Name and Description. Includes T nextafter, T pow, Ts pow, Tn pow, T power, T half_recip, T native_recip, T remainder, Ts remquo, Tn remquo, T rint, Ts rootn, Tn rootn, T round, T rsqrt, T sin, T sincos, T sinh, T sinpi, T sqrt, T tan, T tanh, T tanpi, T tgamma, T trunc.

Math Constants [6.13.2] [9.4.2]

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

Table with 2 columns: Constant Name and Description. Includes MAXFLOAT, HUGE_VALF, HUGE_VAL, INFINITY, 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.

Table with 2 columns: Macro Name and Value. Includes M_E_F, M_LOG2E_F, M_LOG10E_F, M_LN2_F, M_LN10_F, M_PI_F, M_PI_2_F, M_PI_4_F, M_1_PI_F, M_2_PI_F, M_2_SQRTPI_F, M_SQRT2_F, M_SQRT1_2_F.

Integer Built-in Functions [6.13.3]

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

Table with 2 columns: Function Name and Description. Includes Tu abs, Tu abs_diff, T add_sat, T hadd, T rhadd, T clamp, T clz, T ctz, T mad_hi, T mad_sat, T max, T min, T mul_hi, T rotate, T sub_sat, T popcount.

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

Table with 2 columns: Function Name and Description. Includes short[n] upsample, ushort[n] upsample.

Table with 2 columns: Function Name and Description. Includes int[n] upsample, uint[n] upsample, long[n] upsample, ulong[n] upsample.

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

Table with 2 columns: Function Name and Description. Includes T mad24, T mul24.

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.

Table with 2 columns: Function Name and Description. Includes T clamp, Tn clamp, T degrees, T max, Tn max, T min, Tn min.

(Continued on next page >)

Common Functions (continued)

<i>T mix</i> (<i>T x</i> , <i>T y</i> , <i>T a</i>)	Linear blend of <i>x</i> and <i>y</i>
<i>Tn mix</i> (<i>Tn x</i> , <i>Tn y</i> , <i>Ts a</i>)	
<i>T radians</i> (<i>T degrees</i>)	<i>degrees</i> to radians
<i>T step</i> (<i>T edge</i> , <i>T x</i>)	0.0 if <i>x</i> < <i>edge</i> , else 1.0
<i>Tn step</i> (<i>Ts edge</i> , <i>Tn x</i>)	
<i>T smoothstep</i> (<i>T edge0</i> , <i>T edge1</i> , <i>T x</i>)	Step and interpolate
<i>T smoothstep</i> (<i>Ts edge0</i> , <i>Ts edge1</i> , <i>T x</i>)	
<i>T sign</i> (<i>T x</i>)	Sign of <i>x</i>

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] <i>cross</i> (float[3,4] <i>p0</i> , float[3,4] <i>p1</i>)	Cross product
double[3,4] <i>cross</i> (double[3,4] <i>p0</i> , double[3,4] <i>p1</i>)	
half[3,4] <i>cross</i> (half[3,4] <i>p0</i> , half[3,4] <i>p1</i>)	
<i>Ts distance</i> (<i>T p0</i> , <i>T p1</i>)	Vector distance
<i>Ts dot</i> (<i>T p0</i> , <i>T p1</i>)	Dot product

<i>Ts length</i> (<i>T p</i>)	Vector length
<i>T normalize</i> (<i>T p</i>)	Normal vector length 1
float <i>fast_distance</i> (float <i>p0</i> , float <i>p1</i>)	Vector distance
float <i>fast_distance</i> (floatn <i>p0</i> , floatn <i>p1</i>)	
float <i>fast_length</i> (float <i>p</i>)	Vector length
float <i>fast_length</i> (floatn <i>p</i>)	
float <i>fast_normalize</i> (float <i>p</i>)	Normal vector length 1
floatn <i>fast_normalize</i> (floatn <i>p</i>)	

Relational Built-in Functions [6.13.6]

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

int <i>isequal</i> (float <i>x</i> , float <i>y</i>)	Compare of <i>x</i> == <i>y</i>
intrn <i>isequal</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isequal</i> (double <i>x</i> , double <i>y</i>)	
longn <i>isequal</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isequal</i> (half <i>x</i> , half <i>y</i>)	Compare of <i>x</i> != <i>y</i>
shortn <i>isequal</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>isnotequal</i> (float <i>x</i> , float <i>y</i>)	
intrn <i>isnotequal</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isnotequal</i> (double <i>x</i> , double <i>y</i>)	Compare of <i>x</i> > <i>y</i>
longn <i>isnotequal</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isnotequal</i> (half <i>x</i> , half <i>y</i>)	
shortn <i>isnotequal</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>isgreater</i> (float <i>x</i> , float <i>y</i>)	Compare of <i>x</i> >= <i>y</i>
intrn <i>isgreater</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isgreater</i> (double <i>x</i> , double <i>y</i>)	
longn <i>isgreater</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isgreater</i> (half <i>x</i> , half <i>y</i>)	Compare of <i>x</i> >= <i>y</i>
shortn <i>isgreater</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>isgreaterequal</i> (float <i>x</i> , float <i>y</i>)	
intrn <i>isgreaterequal</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isgreaterequal</i> (double <i>x</i> , double <i>y</i>)	Compare of <i>x</i> >= <i>y</i>
longn <i>isgreaterequal</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isgreaterequal</i> (half <i>x</i> , half <i>y</i>)	
shortn <i>isgreaterequal</i> (halfn <i>x</i> , halfn <i>y</i>)	

int <i>isless</i> (float <i>x</i> , float <i>y</i>)	Compare of <i>x</i> < <i>y</i>
intrn <i>isless</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isless</i> (double <i>x</i> , double <i>y</i>)	
longn <i>isless</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isless</i> (half <i>x</i> , half <i>y</i>)	Compare of <i>x</i> < <i>y</i>
shortn <i>isless</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>islessequal</i> (float <i>x</i> , float <i>y</i>)	
intrn <i>islessequal</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>islessequal</i> (double <i>x</i> , double <i>y</i>)	Compare of <i>x</i> <= <i>y</i>
longn <i>islessequal</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>islessequal</i> (half <i>x</i> , half <i>y</i>)	
shortn <i>islessequal</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>islessgreater</i> (float <i>x</i> , float <i>y</i>)	Compare of (<i>x</i> < <i>y</i>) (<i>x</i> > <i>y</i>)
intrn <i>islessgreater</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>islessgreater</i> (double <i>x</i> , double <i>y</i>)	
longn <i>islessgreater</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>islessgreater</i> (half <i>x</i> , half <i>y</i>)	Test for finite value
shortn <i>islessgreater</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>isfinite</i> (float)	
intrn <i>isfinite</i> (floatn)	
int <i>isfinite</i> (double)	Test for finite value
longn <i>isfinite</i> (doublen)	
int <i>isfinite</i> (half)	
shortn <i>isfinite</i> (halfn)	
int <i>isinf</i> (float)	Test for + or - infinity
intrn <i>isinf</i> (floatn)	
int <i>isinf</i> (double)	
longn <i>isinf</i> (doublen)	
int <i>isinf</i> (half)	Test for a NaN
shortn <i>isinf</i> (halfn)	
int <i>isnan</i> (float)	
intrn <i>isnan</i> (floatn)	

int <i>isnan</i> (double)	Test for a NaN
longn <i>isnan</i> (doublen)	
int <i>isnan</i> (half)	
shortn <i>isnan</i> (halfn)	
int <i>isnormal</i> (float)	Test for a normal value
intrn <i>isnormal</i> (floatn)	
int <i>isnormal</i> (double)	
longn <i>isnormal</i> (doublen)	
int <i>isnormal</i> (half)	Test for a normal value
shortn <i>isnormal</i> (halfn)	
int <i>isordered</i> (float <i>x</i> , float <i>y</i>)	
intrn <i>isordered</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isordered</i> (double <i>x</i> , double <i>y</i>)	Test if arguments are ordered
longn <i>isordered</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isordered</i> (half <i>x</i> , half <i>y</i>)	
shortn <i>isordered</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>isunordered</i> (float <i>x</i> , float <i>y</i>)	Test if arguments are unordered
intrn <i>isunordered</i> (floatn <i>x</i> , floatn <i>y</i>)	
int <i>isunordered</i> (double <i>x</i> , double <i>y</i>)	
longn <i>isunordered</i> (doublen <i>x</i> , doublen <i>y</i>)	
int <i>isunordered</i> (half <i>x</i> , half <i>y</i>)	Test for sign bit
shortn <i>isunordered</i> (halfn <i>x</i> , halfn <i>y</i>)	
int <i>signbit</i> (float)	
intrn <i>signbit</i> (floatn)	
int <i>signbit</i> (double)	Test for sign bit
longn <i>signbit</i> (doublen)	
int <i>signbit</i> (half)	
shortn <i>signbit</i> (halfn)	
int <i>any</i> (<i>Ti x</i>)	1 if MSB in component of <i>x</i> is set; else 0
int <i>all</i> (<i>Ti x</i>)	1 if MSB in all components of <i>x</i> are set; else 0

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.

<i>Tn vload</i> (<i>size_t offset</i> , const [constant] <i>T *p</i>)	Read vector data from address (<i>p</i> + (<i>offset</i> * <i>n</i>))
void <i>vstoren</i> (<i>Tn data</i> , <i>size_t offset</i> , <i>T *p</i>)	Write vector data to address (<i>p</i> + (<i>offset</i> * <i>n</i>))
float <i>vload_half</i> (<i>size_t offset</i> , const [constant] half * <i>p</i>)	Read a half from address (<i>p</i> + <i>offset</i>)
floatn <i>vload_halfn</i> (<i>size_t offset</i> , const [constant] half * <i>p</i>)	Read a halfn from address (<i>p</i> + (<i>offset</i> * <i>n</i>))
void <i>vstore_half</i> (float <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	Write a half to address (<i>p</i> + <i>offset</i>)
void <i>vstore_half_R</i> (float <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstore_half</i> (double <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstore_half_R</i> (double <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	

void <i>vstore_half_R</i> (double <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	Write a half to address (<i>p</i> + <i>offset</i>)
void <i>vstore_halfn</i> (floatn <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	Write a half vector to address (<i>p</i> + (<i>offset</i> * <i>n</i>))
void <i>vstore_halfn_R</i> (floatn <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstore_halfn</i> (doublen <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstore_halfn_R</i> (doublen <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
floatn <i>vloada_halfn</i> (<i>size_t offset</i> , const [constant] half * <i>p</i>)	Read half vector data from (<i>p</i> + (<i>offset</i> * <i>n</i>)). For half3, read from (<i>p</i> + (<i>offset</i> * 4)).
void <i>vstorea_halfn</i> (floatn <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	Write half vector data to (<i>p</i> + (<i>offset</i> * <i>n</i>)). For half3, write to (<i>p</i> + (<i>offset</i> * 4)).
void <i>vstorea_halfn_R</i> (floatn <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstorea_halfn</i> (doublen <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	
void <i>vstorea_halfn_R</i> (doublen <i>data</i> , <i>size_t offset</i> , half * <i>p</i>)	

<i>T bitsselect</i> (<i>T a</i> , <i>T b</i> , <i>T c</i>)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
half <i>bitsselect</i> (half <i>a</i> , half <i>b</i> , half <i>c</i>)	
halfn <i>bitsselect</i> (halfn <i>a</i> , halfn <i>b</i> , halfn <i>c</i>)	
<i>T select</i> (<i>T a</i> , <i>T b</i> , <i>Ti c</i>)	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
<i>T select</i> (<i>T a</i> , <i>T b</i> , <i>Tu c</i>)	
halfn <i>select</i> (halfn <i>a</i> , halfn <i>b</i> , shortn <i>c</i>)	
half <i>select</i> (half <i>a</i> , half <i>b</i> , short <i>c</i>)	
halfn <i>select</i> (halfn <i>a</i> , halfn <i>b</i> , ushortn <i>c</i>)	
half <i>select</i> (half <i>a</i> , half <i>b</i> , ushort <i>c</i>)	

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 <i>work_group_barrier</i> (cl_mem_fence_flags <i>flags</i> [, memory_scope <i>scope</i>])	Work-items in a work-group must execute this before any can continue
void <i>atomic_work_item_fence</i> (cl_mem_fence_flags <i>flags</i> [, memory_scope <i>scope</i>])	Orders loads and stores of a work-item executing a kernel
void <i>sub_group_barrier</i> (cl_mem_fence_flags <i>flags</i> [, memory_scope <i>scope</i>])	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, charn, uchar, uchar, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, ulongn, float, floatn, optionally double or doublen, or half or halfn if the cl_khr_fp16 extension is enabled.

event_t <i>async_work_group_copy</i> (__local <i>T *dst</i> , const __global <i>T *src</i> , <i>size_t num_gentypes</i> , event_t <i>event</i>)	Copies <i>num_gentypes</i> <i>T</i> elements from <i>src</i> to <i>dst</i>
event_t <i>async_work_group_copy</i> (__global <i>T *dst</i> , const __local <i>T *src</i> , <i>size_t num_gentypes</i> , event_t <i>event</i>)	
event_t <i>async_work_group_strided_copy</i> (__local <i>T *dst</i> , const __global <i>T *src</i> , <i>size_t num_gentypes</i> , <i>size_t src_stride</i> , event_t <i>event</i>)	Copies <i>num_gentypes</i> <i>T</i> elements from <i>src</i> to <i>dst</i>
event_t <i>async_work_group_strided_copy</i> (__global <i>T *dst</i> , const __local <i>T *src</i> , <i>size_t num_gentypes</i> , <i>size_t dst_stride</i> , event_t <i>event</i>)	
void <i>wait_group_events</i> (int <i>num_events</i> , event_t * <i>event_list</i>)	Wait for <i>async_work_group_copy</i> to complete
void <i>prefetch</i> (const __global <i>T *p</i> , <i>size_t num_gentypes</i>)	Prefetch <i>num_gentypes</i> * sizeof(<i>T</i>) 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.

Atomic Functions

In the following definitions, **A** refers to one of the atomic_* types. **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.

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

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

Atomic Types and Enum Constants

Parameter Type	Values	Description
memory_order	<code>memory_order_relaxed</code> <code>memory_order_acquire</code> <code>memory_order_release</code> <code>memory_order_acq_rel</code> <code>memory_order_seq_cst</code>	Enum which identifies memory ordering constraints.
memory_scope	<code>memory_scope_work_item</code> <code>memory_scope_work_group</code> <code>memory_scope_sub_group</code> memory_scope_sub_group requires the <code>cl_khr_subgroups</code> extension. <code>memory_scope_device</code> (default for functions that do not take a <code>memory_scope</code> argument) <code>memory_scope_all_svm_devices</code>	Enum which identifies scope of memory ordering constraints. memory_scope_sub_group requires the <code>cl_khr_subgroups</code> extension.
atomic_flag	32-bit int representing a lock-free, primitive atomic flag; and several atomic analogs of integer types.	

Atomic integer and floating-point types

<code>atomic_int</code>	<code>atomic_long</code>	<code>atomic_float</code>	<code>atomic_intptr_t</code>	<code>atomic_size_t</code>
<code>atomic_uint</code>	<code>atomic_ulong</code>	<code>atomic_double</code>	<code>atomic_uintptr_t</code>	<code>atomic_ptrdiff_t</code>

Atomic Macros

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

64-bit Atomics [9.3]

The `cl_khr_int64_base_atomics` extension enables 64-bit versions of the following functions: `atom_add`, `atom_sub`, `atom_inc`, `atom_dec`, `atom_xchg`, `atom_cmpxchg`

The `cl_khr_int64_extended_atomics` extension enables 64-bit versions of the following functions: `atom_min`, `atom_max`, `atom_and`, `atom_or`, `atom_xor`

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.

<code>global T * to_global(T *ptr)</code> <code>const global T * to_global(const T *ptr)</code>	global address space
<code>local T * to_local(T *ptr)</code> <code>const local T * to_local(const T *ptr)</code>	local address space
<code>private T * to_private(T *ptr)</code> <code>const private T * to_private(const T *ptr)</code>	private address space
<code>cl_mem_fence_flags get_fence(T *ptr)</code> <code>const cl_mem_fence_flags get_fence(const T *ptr)</code>	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)

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)
```

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, uint sub_group_local_id)
```

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)
```

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.

<code>int read_pipe (pipe T p, T *ptr)</code>	Read packet from <i>p</i> into <i>ptr</i> .	<code>reserve_id_t reserve_read_pipe (pipe T p, uint num_packets)</code>	Reserve <i>num_packets</i> entries for reading from or writing to <i>p</i> .
<code>int read_pipe (pipe T p, reserve_id_t reserve_id, uint index, T *ptr)</code>	Read packet from reserved area of the pipe <i>reserve_id</i> and <i>index</i> into <i>ptr</i> .	<code>reserve_id_t reserve_write_pipe (pipe T p, uint num_packets)</code>	
<code>int write_pipe (pipe T p, const T *ptr)</code>	Write packet specified by <i>ptr</i> to <i>p</i> .	<code>void commit_read_pipe (pipe T p, reserve_id_t reserve_id)</code>	Indicates that all reads and writes to <i>num_packets</i> associated with reservation <i>reserve_id</i> are completed.
<code>int write_pipe (pipe T p, reserve_id_t reserve_id, uint index, const T *ptr)</code>	Write packet specified by <i>ptr</i> to reserved area <i>reserve_id</i> and <i>index</i> .	<code>void commit_write_pipe (pipe T p, reserve_id_t reserve_id)</code>	
<code>bool is_valid_reserve_id (reserve_id_t reserve_id)</code>	Return true if <i>reserve_id</i> is a valid reservation ID and false otherwise.	<code>uint get_pipe_max_packets (pipe T p)</code>	Returns maximum number of packets specified when <i>p</i> was created.
		<code>uint get_pipe_num_packets (pipe T p)</code>	Returns the number of available entries in <i>p</i> .

<code>void work_group_commit_read_pipe (pipe T p, reserve_id_t reserve_id)</code> <code>void work_group_commit_write_pipe (pipe T p, reserve_id_t reserve_id)</code> <code>void sub_group_commit_read_pipe (pipe T p, reserve_id_t reserve_id)</code> <code>void sub_group_commit_write_pipe (pipe T p, reserve_id_t reserve_id)</code>	Indicates that all reads and writes to <i>num_packets</i> associated with reservation <i>reserve_id</i> are completed.
<code>reserve_id_t work_group_reserve_read_pipe (pipe T p, uint num_packets)</code> <code>reserve_id_t work_group_reserve_write_pipe (pipe T p, uint num_packets)</code> <code>reserve_id_t sub_group_reserve_read_pipe (pipe T p, uint num_packets)</code> <code>reserve_id_t sub_group_reserve_write_pipe (pipe T p, uint num_packets)</code>	Reserve <i>num_packets</i> entries for reading from or writing to <i>p</i> . 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.

<code>int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags, const ndrange_t ndrange, void (^block)(void))</code>	Allows a work-item to enqueue a block for execution to <i>queue</i> .
<code>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))</code>	Work-items can enqueue multiple blocks to a device queue(s). <i>flags</i> may be one of <code>CLK_ENQUEUE_FLAGS_{NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP}</code>
<code>int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags, const ndrange_t ndrange, void (^block)(local void *, ...), uint size0, ...)</code>	
<code>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, ...)</code>	
<code>uint get_kernel_work_group_size (void (^block)(void))</code> <code>uint get_kernel_work_group_size (void (^block)(local void *, ...))</code>	Query the maximum work-group size that can be used to execute a block.
<code>uint get_kernel_preferred_work_group_size_multiple (void (^block)(void))</code> <code>uint get_kernel_preferred_work_group_size_multiple (void (^block)(local void *, ...))</code>	Returns the preferred multiple of work-group size for launch.
<code>int enqueue_marker (queue_t queue, uint num_events_in_wait_list, const clk_event_t *event_wait_list, clk_event_t *event_ret)</code>	Enqueue a marker command to <i>queue</i> .
<code>uint get_kernel_sub_group_count_for_ndrange (const ndrange_t ndrange, void (^block)(void))</code> <code>uint get_kernel_sub_group_count_for_ndrange (const ndrange_t ndrange, void (^block)(local void *, ...))</code>	Returns number of subgroups in each workgroup of the dispatch.
<code>uint get_kernel_max_sub_group_size_for_ndrange (const ndrange_t ndrange, void (^block)(void))</code> <code>uint get_kernel_max_sub_group_size_for_ndrange (const ndrange_t ndrange, void (^block)(local void *, ...))</code>	Returns the maximum sub-group size for a block.

Miscellaneous Vector Functions [6.13.12]

Tm and *Tn* are type charn, uchar, shortn, ushortn, intrn, uintn, longn, ulongn, floatn, optionally doublen, or halfn 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, ushortn, uintn, or ulongn.

<code>int vec_step (Tn a)</code> <code>int vec_step (typename)</code>	Takes a 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.
<code>Tn shuffle (Tm x, TUn mask)</code> <code>Tn shuffle2 (Tm x, Tm y, TUn mask)</code>	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.

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.

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

Helper Built-in Functions [6.13.17.9]

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

OpenCL Image Processing Reference

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

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 [5.3.4]

`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)`

(Continued on next page >)

Image Objects (continued)

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, CL_MAP_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 5.9] CL_IMAGE_FORMAT, CL_IMAGE_BUFFER,
            CL_IMAGE_ARRAY_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
```

Also see [clGetMemObjectInfo](#) [5.4.5]

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 + write): CL_HALF_FLOAT, CL_FLOAT,
                    CL_UNORM_INT8, CL_UNORM_INT16,
                    CL_SNORM_INT8, CL_SNORM_INT16,
                    CL_SIGNED_INT8, CL_SIGNED_INT16,
                    CL_UNSIGNED_INT8, CL_UNSIGNED_INT16
```

```
CL_DEPTH (read + 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 + write): CL_HALF_FLOAT, CL_FLOAT,
                    CL_UNORM_INT8, CL_UNORM_INT16,
                    CL_SNORM_INT8, CL_SNORM_INT16,
                    CL_SIGNED_INT8, CL_SIGNED_INT16,
                    CL_UNSIGNED_INT8, CL_UNSIGNED_INT16
```

```
CL_RGBA (read + write): CL_HALF_FLOAT, CL_FLOAT,
                    CL_UNORM_INT8, CL_UNORM_INT16,
                    CL_SNORM_INT8, CL_SNORM_INT16,
                    CL_SIGNED_INT8, CL_SIGNED_INT16,
                    CL_UNSIGNED_INT8, CL_UNSIGNED_INT16
```

```
CL_BGRA (read + 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: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT8, CL_UNORM_INT16,
            CL_SIGNED_INT8, CL_SIGNED_INT16, CL_UNSIGNED_INT8, CL_UNSIGNED_INT16,
            CL_SNORM_INT8, CL_SNORM_INT16
```

```
CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT,
              CL_UNORM_INT8, CL_UNORM_INT16, CL_SNORM_INT8, CL_SNORM_INT16
```

```
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_INT8, CL_UNORM_INT16, CL_HALF_FLOAT,
              CL_FLOAT, CL_SNORM_INT8, CL_SNORM_INT16
```

```
CL_RG, CL_RA: CL_HALF_FLOAT, CL_FLOAT,
              CL_UNORM_INT8, CL_UNORM_INT16, CL_SIGNED_INT8, CL_SIGNED_INT16,
              CL_UNSIGNED_INT8, CL_UNSIGNED_INT16, CL_SNORM_INT8, CL_SNORM_INT16
```

```
CL_RGB: CL_UNORM_SHORT_555, CL_UNORM_SHORT_565,
        CL_UNORM_INT101010
```

```
CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8,
          CL_UNSIGNED_INT8, CL_SNORM_INT8
```

```
CL_BGRA: CL_{SIGNED, UNSIGNED}_INT8, CL_SNORM_INT8
```

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. Writing to sRGB images from a kernel requires 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, and image 3D writes require the extension `cl_khr_3d_image_writes`.

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 (image1d_t image, sampler_t sampler,
    {int, float} coord)
```

```
float4 read_imagef (image1d_t image, int coord)
```

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

```
float4 read_imagef (image1d_array_t image, int2 coord)
```

```
float4 read_imagef (image1d_buffer_t image, int coord)
```

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

```
int4 read_imagei (image1d_t image, int coord)
```

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

```
int4 read_imagei (image1d_array_t image, int2 coord)
```

```
int4 read_imagei (image1d_buffer_t image, int coord)
```

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

```
uint4 read_imageui (image1d_t image, int coord)
```

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

```
uint4 read_imageui (image1d_array_t image, int2 coord)
```

```
uint4 read_imageui (image1d_buffer_t image, int coord)
```

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

```
half4 read_imageh (image1d_t image, int coord)
```

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

```
half4 read_imageh (image1d_array_t image, int2 coord)
```

```
half4 read_imageh (image1d_buffer_t image, int coord)
```

```
void write_imagef (image1d_t image, int coord, float4 color)
```

```
void write_imagef (image1d_array_t image, int2 coord,
    float4 color)
```

```
void write_imagef (image1d_buffer_t image, int coord,
    float4 color)
```

```
void write_imagei (image1d_t image, int coord, int4 color)
```

```
void write_imagei (image1d_array_t image, int2 coord,
    int4 color)
```

```
void write_imagei (image1d_buffer_t image, int coord,
    int4 color)
```

```
void write_imageh (image1d_t image, int coord, half4 color)
```

```
void write_imageh (image1d_array_t image, int2 coord,
    half4 color)
```

```
void write_imageh (image1d_buffer_t image, int coord,
    half4 color)
```

```
void write_imageui (image1d_t image, int coord, uint4 color)
```

```
void write_imageui (image1d_array_t image, int2 coord,
    uint4 color)
```

```
void write_imageui (image1d_buffer_t image, int coord,
    uint4 color)
```

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 (image2d_t image, sampler_t sampler,
    {int2, float2} coord)
```

```
float4 read_imagef (image2d_t image, int2 coord)
```

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

```
float4 read_imagef (image2d_array_t image, int4 coord)
```

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

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

```
float read_imagef (image2d_depth_t image, int2 coord)
```

```
float read_imagef (image2d_array_depth_t image, int4 coord)
```

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

```
int4 read_imagei (image2d_t image, int2 coord)
```

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

```
int4 read_imagei (image2d_array_t image, int4 coord)
```

```
int4 read_imagei (image2d_depth_t image, sampler_t sampler,
    {int4, float4} coord)
```

```
int4 read_imagei (image2d_array_depth_t image, int4 coord)
```

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

```
uint4 read_imageui (image2d_t image, int2 coord)
```

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

```
uint4 read_imageui (image2d_array_t image, int4 coord)
```

Read and write functions for 2D images (continued)

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

```
half4 read_imageh (image2d_t image, int2 coord)
```

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

```
half4 read_imageh (image2d_array_t image, int4 coord)
```

```
void write_imagef (image2d_t image, int2 coord, float4 color)
```

```
void write_imagef (image2d_array_t image, int4 coord,
    float4 color)
```

```
void write_imagef (image2d_depth_t image, int2 coord, int lod,
    float depth)
```

```
void write_imagef (image2d_array_depth_t image, int4 coord,
    int lod, float depth)
```

```
void write_imagei (image2d_t image, int2 coord, int4 color)
```

```
void write_imagei (image2d_array_t image, int4 coord,
    int4 color)
```

```
void write_imageui (image2d_t image, int2 coord, uint4 color)
```

```
void write_imageui (image2d_array_t image, int4 coord,
    uint4 color)
```

```
void write_imageh (image2d_t image, int2 coord, half4 color)
```

```
void write_imageh (image2d_array_t image, int4 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.

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

```
float4 read_imagef (image3d_t image, int4 coord)
```

```
int4 read_imagei (image3d_t image, sampler_t sampler,
    {int4, float4} coord)
```

```
int4 read_imagei (image3d_t image, int4 coord)
```

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

```
uint4 read_imageui (image3d_t image, int4 coord)
```

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

```
half4 read_imageh (image3d_t image, int4 coord)
```

```
void write_imagef (image3d_t image, int4 coord, float4 color)
```

```
void write_imagei (image3d_t image, int4 coord, int4 color)
```

```
void write_imageui (image3d_t image, int4 coord, uint4 color)
```

```
void write_imageh (image3d_t image, int4 coord, half4 color)
```

(Continued on next page >)

Image Read and Write (continued)

Extended mipmap read and write functions [9.18.2.1]
These functions require the `cl_khr_mipmap_image` and `cl_khr_mipmap_image_writes` extensions.

```
float read_imagef (image2d_[depth_]t image,
                  sampler_t sampler, float2 coord, float lod)
int4 read_imagei (image2d_t image, sampler_t sampler,
                 float2 coord, float lod)
uint4 read_imageui (image2d_t image, sampler_t sampler,
                  float2 coord, float lod)

float read_imagef (image2d_[depth_]t image,
                  sampler_t sampler, float2 coord, float2 gradient_x,
                  float2 gradient_y)
int4 read_imagei (image2d_t image, sampler_t sampler,
                 float2 coord, float2 gradient_x, float2 gradient_y)
uint4 read_imageui (image2d_t image, sampler_t sampler,
                  float2 coord, float2 gradient_x, float2 gradient_y)

float4 read_imagef (image1d_t image, sampler_t sampler,
                  float coord, float lod)
int4 read_imagei (image1d_t image, sampler_t sampler,
                 float coord, float lod)
uint4 read_imageui (image1d_t image, sampler_t sampler,
                  float coord, float lod)

float4 read_imagef (image1d_t image, sampler_t sampler,
                  float coord, float gradient_x, float gradient_y)
int4 read_imagei (image1d_t image, sampler_t sampler,
                 float coord, float gradient_x, float gradient_y)
uint4 read_imageui (image1d_t image, sampler_t sampler,
                  float coord, float gradient_x, float gradient_y)

float4 read_imagef (image3d_t image, sampler_t sampler,
                  float4 coord, float lod)
int4 read_imagei (image3d_t image, sampler_t sampler,
                 float4 coord, float lod)
uint4 read_imageui (image3d_t image, sampler_t sampler,
                  float4 coord, float lod)

float4 read_imagef (image3d_t image, sampler_t sampler,
                  float4 coord, float4 gradient_x, float4 gradient_y)
int4 read_imagei (image3d_t image, sampler_t sampler,
                 float4 coord, float4 gradient_x, float4 gradient_y)
uint4 read_imageui (image3d_t image, sampler_t sampler,
                  float4 coord, float4 gradient_x, float4 gradient_y)

float4 read_imagef (image1d_array_t image, sampler_t sampler,
                  float2 coord, float lod)
int4 read_imagei (image1d_array_t image, sampler_t sampler,
                 float2 coord, float lod)
uint4 read_imageui (image1d_array_t image, sampler_t sampler,
                  float2 coord, float lod)

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

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 5.14]
CL_SAMPLER_NORMALIZED_COORDS,
CL_SAMPLER_ADDRESSING_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, CL_SAMPLER_FILTER_MODE,
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_NORMALIZED_COORDS [Table 5.15]
```

```
int4 read_imagei (image1d_array_t image, sampler_t sampler,
                 float2 coord, float gradient_x, float gradient_y)
uint4 read_imageui (image1d_array_t image, sampler_t sampler,
                  float2 coord, float gradient_x, float gradient_y)

float read_imagef (image2d_array_[depth_]t image,
                  sampler_t sampler, float4 coord, float lod)
int4 read_imagei (image2d_array_t image, sampler_t sampler,
                 float4 coord, float lod)
uint4 read_imageui (image2d_array_t image,
                  sampler_t sampler, float4 coord, float lod)

float read_imagef (image2d_array_[depth_]t image,
                  sampler_t sampler, float4 coord, float2 gradient_x,
                  float2 gradient_y)
int4 read_imagei (image2d_array_t image, sampler_t sampler,
                 float4 coord, float2 gradient_x, float2 gradient_y)
uint4 read_imageui (image2d_array_t image, sampler_t
                  sampler, float4 coord, float2 gradient_x, float2 gradient_y)

void write_imagef (image2d_[depth_]t image, int2 coord,
                  int lod, float4 color)
void write_imagei (image2d_t image, int2 coord, int lod,
                  int4 color)
void write_imageui (image2d_t image, int2 coord, int lod,
                  uint4 color)

void write_imagef (image1d_t image, int coord, int lod, float4 color)
void write_imagei (image1d_t image, int coord, int lod, int4 color)
void write_imageui (image1d_t image, int coord, int lod, uint4 color)

void write_imagef (image1d_array_t image, int2 coord, int lod,
                  float4 color)
void write_imagei (image1d_array_t image, int2 coord, int lod,
                  int4 color)
void write_imageui (image1d_array_t image, int2 coord, int lod,
                  uint4 color)

void write_imagef (image2d_array_[depth_]t image, int4 coord,
                  int lod, float4 color)
void write_imagei (image2d_array_t image, int4 coord, int lod,
                  int4 color)
void write_imageui (image2d_array_t image, int4 coord, int lod,
                  uint4 color)

void write_imagef (image3d_t image, int4 coord, int lod,
                  float4 color)
void write_imagei (image3d_t image, int4 coord, int lod,
                  int4 color)
void write_imageui (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 (image2d_msaa_depth_t image,
                  int2 coord, int sample)
float read_imagef (image2d_array_depth_msaa_t image,
                  int4 coord, int sample)

float4 read_imagef{f, i, ui} (image2d_msaa_t image,
                              int2 coord, int sample)
float4 read_imagef{f, i, ui} (image2d_array_msaa_t image,
                              int4 coord, int sample)
```

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_{REPEAT, CLAMP, NONE},
CLK_ADDRESS_{CLAMP_TO_EDGE},
CLK_ADDRESS_{MIRRORED_REPEAT}

filter-mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```

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 (image{1,2,3}d_t image)
int get_image_width (image1d_buffer_t image)
int get_image_width (image{1,2}d_array_t image)
int get_image_width (image2d_[array_]depth_t image)
int get_image_width (image2d_[array_]msaa_t image)
int get_image_width (image2d_[array_]msaa_depth_t image)

int get_image_height (image{2,3}d_t image)
int get_image_height (image2d_array_t image)
int get_image_height (image2d_[array_]depth_t image)
int get_image_height (image2d_[array_]msaa_t image)
int get_image_height (image2d_[array_]msaa_depth_t image)

int get_image_depth (image3d_t image)
```

Query image array size

```
size_t get_image_array_size (image1d_array_t image)
size_t get_image_array_size (image2d_array_t image)
size_t get_image_array_size (image2d_array_depth_t image)
size_t get_image_array_size (
  image2d_array_msaa_depth_t image)
```

Query image dimensions

```
int2 get_image_dim (image2d_t image)
int2 get_image_dim (image2d_array_t image)
int4 get_image_dim (image3d_t image)
int2 get_image_dim (image2d_[array_]depth_t image)
int2 get_image_dim (image2d_[array_]msaa_t image)
int2 get_image_dim (image2d_[array_]msaa_depth_t image)
```

Query image Channel data type and order

```
int get_image_channel_data_type (image{1,2,3}d_t image)
int get_image_channel_data_type (image1d_buffer_t image)
int get_image_channel_data_type (image{1,2}d_array_t image)
int get_image_channel_data_type (
  image2d_[array_]depth_t image)
int get_image_channel_data_type (
  image2d_[array_]msaa_t image)
int get_image_channel_data_type (
  image2d_[array_]msaa_depth_t image)

int get_image_channel_order (image{1,2,3}d_t image)
int get_image_channel_order (image1d_buffer_t image)
int get_image_channel_order (image{1,2}d_array_t image)
int get_image_channel_order (
  image2d_[array_]depth_t image)
int get_image_channel_order (
  image2d_[array_]msaa_t image)
int get_image_channel_order (
  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 (image1d_t image)
int get_image_num_mip_levels (image2d_[depth_]t image)
int get_image_num_mip_levels (image3d_t image)
int get_image_num_mip_levels (image1d_array_t image)
int get_image_num_mip_levels (
  image2d_array_[depth_]t image)

int get_image_num_samples (
  image2d_[array_]msaa_t image)
int get_image_num_samples (
  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
```

A C++ wrapper is available for developing OpenCL applications in C++.

See www.khronos.org/registry/cl/

OpenCL Extensions Reference

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: `clGetPlatformInfo()` or `clGetDeviceInfo()`

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

<code>cl_apple_gl_sharing</code> (see <code>cl_khr_gl_sharing</code>)
<code>cl_khr_3d_image_writes</code>
<code>cl_khr_byte_addressable_store</code>

<code>cl_khr_context_abort</code>
<code>cl_khr_d3d10_sharing</code>
<code>cl_khr_d3d11_sharing</code>
<code>cl_khr_depth_images</code>
<code>cl_khr_dx9_media_sharing</code>
<code>cl_khr_egl_event</code>
<code>cl_khr_egl_image</code>
<code>cl_khr_fp16</code>
<code>cl_khr_fp64</code>
<code>cl_khr_gl_depth_images</code>
<code>cl_khr_gl_event</code>
<code>cl_khr_gl_msaa_sharing</code>
<code>cl_khr_gl_sharing</code>
<code>cl_khr_global_int32_base_atomics - atomic_*</code>

<code>cl_khr_global_int32_extended_atomics - atomic_*</code>
<code>cl_khr_icd</code>
<code>cl_khr_image2d_from_buffer</code>
<code>cl_khr_initialize_memory</code>
<code>cl_khr_int64_base_atomics - atom_*</code>
<code>cl_khr_int64_extended_atomics - atom_*</code>
<code>cl_khr_local_int32_base_atomics - atomic_*</code>
<code>cl_khr_local_int32_extended_atomics - atomic_*</code>
<code>cl_khr_mipmap_image</code>
<code>cl_khr_mipmap_image_writes</code>
<code>cl_khr_srgb_image_writes</code>
<code>cl_khr_spir</code>
<code>cl_khr_subgroups</code>
<code>cl_khr_terminate_context</code>

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`)

DX9 Media Surface Sharing [9.9]

These functions require the extension `cl_khr_dx9_media_sharing`. The associated header file is `cl_dx9_media_sharing.h`.

```
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)
```

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_egl_event` extension.

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`

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)
```

```
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 Interoperability [9.19, 9.20]

Create CL Image Objects from EGL [9.19]

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 [9.20]

This function requires the extension `cl_khr_egl_event`.

```
cl_event clCreateEventFromEGLSyncKHR (
    cl_context context, CLEGLSyncKHR sync,
    CLEGLDisplayKHR display, cl_int *errcode_ret)
```


OpenCL Reference Card Index

The following index shows each item included on this card along with the page on which it is described. The color of the row in the table below is the color of the box to which you should refer.

A					
Access Qualifiers	10	clEnqueueNDRangeKernel	3	clRetainProgram	3
Address Space Qualifier Functions	7	clEnqueueReadBuffer	1	clRetainSampler	10
Aligned attribute qualifiers	5	clEnqueueReadBufferRect	1	clSetEventCallback	4
Async Copies and Prefetch	6	clEnqueueReadImage	8	clSetKernelArg	3
Atomic Functions	7	clEnqueueReleaseD3D10ObjectsKHR	11	clSetKernelArgSVMPointer	3
Attribute Qualifiers	5	clEnqueueReleaseD3D11ObjectsKHR	11	clSetKernelExecInfo	3
		clEnqueueReleaseDX9MediaSurfacesKHR	11	clSetMemObjectDestructorCallback	2
		clEnqueueReleaseEGLObjectsKHR	11	clSetUserEventStatus	3
B		clEnqueueReleaseGLObjects	11	clSVMAlloc	2
Barriers	4	clEnqueueSVM[Un]Map	3	clSVMFree	2
Blocks	4	clEnqueueSVMFree	2	clTerminateContextKHR	1
Buffer Objects	1-2	clEnqueueSVMMem{cpy, Fill}	3	clUnloadPlatformCompiler	3
		clEnqueueUnmapMemObject	2	clWaitForEvents	3
C		clEnqueueWriteBuffer	2	Command Queues	1
cl_KHR	11	clEnqueueWriteBufferRect	2	Common Built-in Functions	5-6
clBuildProgram	3	clEnqueueWriteImage	8	Compiler Options	3
clCompileProgram	3	clFinish	3	const sampler_t	10
clCreateBuffer	1	clFlush	3	Contexts	1
clCreateCommandQueueWithProperties	1	clGetCommandQueueInfo	1	Conversions and Type Casting	2
clCreateContext	1	clGetContextInfo	1	Copy Between Image, Buffer Objects	8
clCreateContextFromType	1	clGetDeviceIDs	1		
clCreateEventFromEGLsyncKHR	11	clGetDeviceIDsFromD3D10KHR	11	D	
clCreateEventFromGLsyncKHR	11	clGetDeviceIDsFromD3D11KHR	11	Data Types	4
clCreateFromD3D10BufferKHR	11	clGetDeviceIDsFromDX9MediaAdapterKHR	11	Debugging options	3
clCreateFromD3D10Texture2DKHR	11	clGetDeviceInfo	1	Device Architecture Diagram	2
clCreateFromD3D10Texture3DKHR	11	clGetEventInfo	3	Direct3D 10 Sharing	11
clCreateFromD3D11BufferKHR	11	clGetEventProfilingInfo	4	Direct3D 11 Sharing	11
clCreateFromD3D11Texture2DKHR	11	clGetExtensionFunctionAddressForPlatform	1	DX9 Media Surface Sharing	11
clCreateFromD3D11Texture3DKHR	11	clGetGLContextInfoKHR	11	E - F	
clCreateFromDX9MediaSurfaceKHR	11	clGetGLObjectInfo	11	EGL Interoperability	11
clCreateFromEGLImageKHR	11	clGetGLTextureInfo	11	Enqueing & Kernel Query Built-in Functions	8
clCreateFromGLBuffer	11	clGetKernelArgInfo	3	Event Built-in Functions	8
clCreateFromGLRenderbuffer	11	clGetKernelInfo	3	Event Objects	3
clCreateFromGLTexture	11	clGetKernelSubGroupInfoKHR	3	Execute Kernels	3
clCreatelImage	8	clGetKernelWorkGroupInfo	3	Extension Function Pointers	1
clCreateKernel	3	clGetMemObjectInfo	2	Extensions	11
clCreateKernelsInProgram	3	clGetMemObjectInfo	9	Fence Functions	6
clCreatePipe	2	clGetPipeInfo	2		
clCreateProgramWithBinary	3	clGetPlatformIDs	1	G - H	
clCreateProgramWithBuiltInKernels	3	clGetPlatformInfo	1	Geometric Built-in Functions	6
clCreateProgramWithSource	3	clGetProgramBuildInfo	3	Helper Built-in Functions	8
clCreateSamplerWithProperties	10	clGetProgramInfo	3		
clCreateSubBuffer	1	clGetSamplerInfo	10	I	
clCreateSubDevices	1	clGetSupportedImageFormats	8	Image Formats	9
clCreateUserEvent	3	clGetPlatformIDsKHR	1	Image Objects	8-9
clEnqueueAcquireD3D10ObjectsKHR	11	clLinkProgram	3	Image Query Functions	10
clEnqueueAcquireD3D11ObjectsKHR	11	clReleaseCommandQueue	1	Image Read and Write Functions	9-10
clEnqueueAcquireDX9MediaSurfacesKHR	11	clReleaseContext	1	Integer Built-in Functions	5
clEnqueueAcquireEGLObjectsKHR	11	clReleaseDevice	1		
clEnqueueAcquireGLObjects	11	clReleaseEvent	4	K	
clEnqueueBarrierWithWaitList	4	clReleaseKernel	3	Kernel Arguments and Queries	3
clEnqueueCopyBuffer	2	clReleaseMemObject	2	Kernel Objects	3
clEnqueueCopyBufferToImage	8	clReleaseProgram	3	Kernel Query Built-in Functions	8
clEnqueueCopyImage	8	clReleaseSampler	10		
clEnqueueCopyImageToBuffer	8	clRetainCommandQueue	1	L	
clEnqueueFillBuffer	2	clRetainContext	1	Library linking options	3
clEnqueueFillImage	8	clRetainDevice	1	Linker Options	3
clEnqueueMapBuffer	2	clRetainEvent	3		
clEnqueueMapImage	9	clRetainKernel	3	M	
clEnqueueMarkerWithWaitList	4	clRetainMemObject	2	Map and Unmap Image Objects	9
clEnqueueMigrateMemObjects	2			Map Buffer Objects	2
clEnqueueNativeKernel	3			Markers, Barriers, Waiting for Events	4
				Math Built-in Functions	5
				Math Constants	5
				Memory Fence Functions	6
				Memory Objects	2
				Migrate Memory Objects	2
				O	
				OpenCL Class Diagram	2
				OpenCL Extensions	11
				OpenGL Sharing	11
				Operators	4
				Optimization options	3
				P	
				Partitioning a Device	1
				Pipe Built-in Functions	8
				Pipes	2
				Prefetch	6
				Preprocessor	3
				Preprocessor Directives & Macros	4
				printf Function	7
				Profiling Operations	4
				Program linking options	3
				Program Objects	3
				Q	
				Qualifiers	4
				Query Image Objects	9
				Query Image Functions	10
				Query List of Supported Image Formats	8
				Query Memory Object	2
				Query Program Objects	3
				Querying Platform Info & Devices	1
				R	
				Read, Write, Copy Buffer Objects	1-2
				Read, Write, Copy, Fill Image Objects	8
				Read and Write Image Objects	9-10
				Relational Built-in Functions	6
				S - T	
				Sampler Objects, Declaration Fields	10
				Scalar Data Types	4
				Separate Compilation and Linking	3
				Share Objects	11
				Shared Virtual Memory	2-3
				SPIR compiler options	3
				Supported Data Types	4
				SVM Sharing Granularity	2
				Synchronization & Memory Fence Functions	6
				Type Casting Examples	2
				Types	4
				U - V	
				Unload the OpenCL Compiler	3
				Unroll attribute qualifiers	5
				Vector Component Addressing	4
				Vector Data Load/Store	6
				Vector Functions	8
				Vector Data Types	4
				Version	3
				W	
				Waiting for Events	3
				Warning request/suppress	3
				Workgroup Functions	7
				Work-Item Built-in Functions	4-5



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.

OpenCL is a trademark of Apple Inc. and is used under license by Khronos.