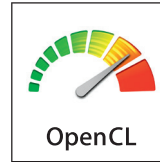


OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance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

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



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

OpenCL API Reference

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

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. **Items in blue apply when the appropriate extension is supported.**

Querying Platform Info & Devices [4.1-2] [9.16.9]

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

`cl_int 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_HOST_TIMER_RESOLUTION, CL_PLATFORM_ICD_SUFFIX_KHR [Table 4.1]

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

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

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

param_name: [Table 4.3]
 CL_DEVICE_ADDRESS_BITS, CL_DEVICE_AVAILABLE, CL_DEVICE_BUILT_IN_KERNELS, CL_DEVICE_COMPILER_AVAILABLE, CL_DEVICE_{DOUBLE, HALF, SINGLE} FP CONFIG, CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_EXTENSIONS, CL_DEVICE_ERROR_CORRECTION_SUPPORT, CL_DEVICE_EXECUTION_CAPABILITIES, CL_DEVICE_GLOBAL_MEM_CACHE_{SIZE, TYPE}, CL_DEVICE_GLOBAL_MEM_{CACHELINE_SIZE, SIZE}, CL_DEVICE_GLOBAL_VARIABLE_PREFERRED_TOTAL_SIZE, CL_DEVICE_IL_VERSION, CL_DEVICE_IMAGE_MAX_{ARRAY, BUFFER} SIZE, CL_DEVICE_IMAGE_SUPPORT, CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT}, CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}, CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT, CL_DEVICE_IMAGE_PITCH_ALIGNMENT, CL_DEVICE_LINKER_AVAILABLE, CL_DEVICE_LOCAL_MEM_{TYPE, SIZE}, CL_DEVICE_MAX_{CLOCK_FREQUENCY, PIPE_ARGS}, CL_DEVICE_MAX_{COMPUTE_UNITS, SAMPLERS}, CL_DEVICE_MAX_CONSTANT_{ARGS, BUFFER_SIZE}, CL_DEVICE_MAX_GLOBAL_VARIABLE_SIZE, CL_DEVICE_MAX_{MEM_ALLOC, PARAMETER} SIZE, CL_DEVICE_MAX_NUM_SUB_GROUPS, CL_DEVICE_MAX_ON_DEVICE_{QUEUES, EVENTS}, CL_DEVICE_MAX_{READ, WRITE} IMAGE_ARGS, CL_DEVICE_MAX_READ_WRITE_IMAGE_ARGS, CL_DEVICE_MAX_SUB_GROUPS, CL_DEVICE_MAX_WORK_GROUP_SIZE, CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES}, CL_DEVICE_MEM_BASE_ADDR_ALIGN, CL_DEVICE_NAME, CL_DEVICE_NATIVE_VECTOR_WIDTH_X (where X may be CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT), CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT,

CL_DEVICE_{OPENCL_C_VERSION, PARENT_DEVICE}, CL_DEVICE_PARTITION_AFFINITY_DOMAIN, CL_DEVICE_PARTITION_MAX_SUB_DEVICES, CL_DEVICE_PARTITION_{PROPERTIES, TYPE}, CL_DEVICE_PIPE_MAX_ACTIVE_RESERVATIONS, CL_DEVICE_PIPE_MAX_PACKET_SIZE, CL_DEVICE_{PLATFORM, PRINTF_BUFFER_SIZE}, CL_DEVICE_PREFERRED_Y_ATOMIC_ALIGNMENT (where Y may be LOCAL, GLOBAL, PLATFORM), CL_DEVICE_PREFERRED_VECTOR_WIDTH_Z (where Z may be CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT), CL_DEVICE_PREFERRED_INTEROP_USER_SYNC, CL_DEVICE_PROFILE, CL_DEVICE_PROFILING_TIMER_RESOLUTION, CL_DEVICE_SPIR_VERSIONS, CL_DEVICE_SUBGROUP_INDEPENDENT_FORWARD_PROGRESS, CL_DEVICE_QUEUE_ON_{DEVICE, HOST} PROPERTIES, CL_DEVICE_QUEUE_ON_DEVICE_MAX_SIZE, CL_DEVICE_QUEUE_ON_DEVICE_PREFERRED_SIZE, CL_DEVICE_{REFERENCE_COUNT, VENDOR_ID}, CL_DEVICE_SVM_CAPABILITIES, CL_DEVICE_TERMINATE_CAPABILITY_KHR, CL_DEVICE_{TYPE, VENDOR}, CL_DEVICE_VENDOR_ID, CL_{DEVICE, DRIVER}_VERSION

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

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

Partitioning a Device [4.3]

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

properties: [Table 4.4] CL_DEVICE_PARTITION_EQUALLY, CL_DEVICE_PARTITION_BY_COUNTS, CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN

`cl_int clRetainDevice` (cl_device_id device)
`cl_int clReleaseDevice` (cl_device_id device)

Contexts [4.4]

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

properties: [Table 4.5]

NULL or CL_CONTEXT_PLATFORM, CL_CONTEXT_INTEROP_USER_SYNC, CL_CONTEXT_{D3D10, D3D11} DEVICE_KHR, CL_CONTEXT_ADAPTER_{D3D9, D3D9EX}_KHR, CL_CONTEXT_ADAPTER_DXVA_KHR, CL_CONTEXT_MEMORY_INITIALIZE_KHR, CL_CONTEXT_TERMINATE_KHR, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

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

properties: See `clCreateContext`
 device_type: See `clGetDeviceIDs`

`cl_int clRetainContext` (cl_context context)
`cl_int clReleaseContext` (cl_context context)

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

param_name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, NUM_DEVICES, PROPERTIES}, CL_CONTEXT_{D3D10, D3D11}_PREFER_SHARED_RESOURCES_KHR [Table 4.6]

`cl_int clTerminateContextKHR` (cl_context context)

Get CL Extension Function Pointers [9.2]

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

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

`cl_int clRetainCommandQueue` (cl_command_queue command_queue)

`cl_int clReleaseCommandQueue` (cl_command_queue command_queue)

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

param_name: [Table 5.2]

CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE[DEFAULT], CL_QUEUE_SIZE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

The OpenCL Runtime

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

Command Queues [5.1]

`cl_command_queue`

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

properties: [Table 5.1] CL_QUEUE_SIZE, CL_QUEUE_PROPERTIES (bitfield which may be set to an OR of CL_QUEUE_* where * may be: OUT_OF_ORDER_EXEC_MODE_ENABLE, PROFILING_ENABLE, ON_DEVICE[DEFAULT]), CL_QUEUE_THROTTLE_{HIGH, MED, LOW}_KHR (requires the cl_khr_throttle_hint extension), CL_QUEUE_PRIORITY_KHR (bitfield which may be one of CL_QUEUE_PRIORITY_HIGH_KHR, CL_QUEUE_PRIORITY_MED_KHR, CL_QUEUE_PRIORITY_LOW_KHR (requires the cl_khr_priority_hints extension))

Buffer Objects

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

Create Buffer Objects [5.2.1]

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

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

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

flags: See clCreateBuffer
buffer_create_type: CL_BUFFER_CREATE_TYPE_REGION

Read, Write, Copy, Fill Buffer Objects [5.2.2-3]

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

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

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

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

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

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

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

Map Buffer Objects [5.2.4]

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

map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

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)
```

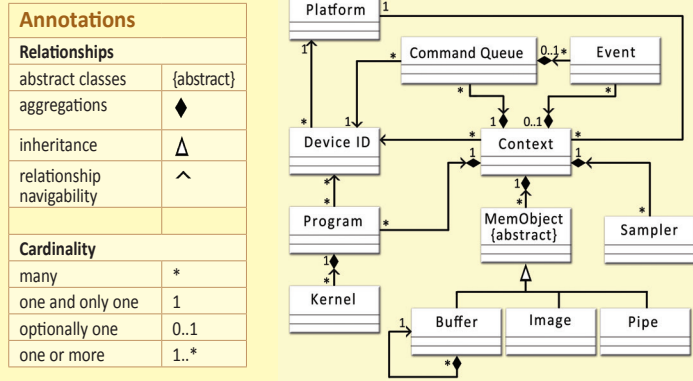
```
cl_int clEnqueueUnmapMemObject (cl_command_queue command_queue, cl_mem memobj, void *mapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
```

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)
```

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.



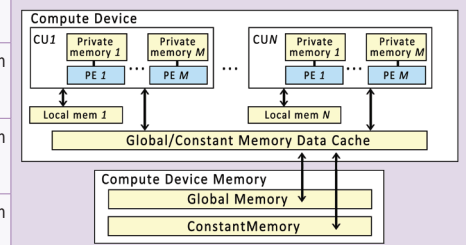
¹ Unified Modeling Language (<http://www.uml.org/>) is a trademark of Object Management Group (OMG).

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

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



Conversions and Type Casting Examples [6.2]

```
T a = (T)b; // Scalar to scalar, // or scalar to vector
T a = convert_T(b);
T a = convert_T_R(b);
T a = as_T(b);
T a = convert_T_sat_R(b);
```

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

Pipes

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

Create Pipe Objects [5.4.1]

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

flags: 0 or CL_MEM_READ_WRITE, CL_MEM_{READ, WRITE}_ONLY, CL_MEM_HOST_NO_ACCESS

Pipe Object Queries [5.4.2]

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

param_name:
CL_PIPE_PACKET_SIZE, CL_PIPE_MAX_PACKETS

flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query Memory Object [5.5.5]

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

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

Shared Virtual Memory

Shared Virtual Memory (SVM) allows the host and kernels executing on devices to directly share complex, pointer-containing data structures such as trees and linked lists. See more on SVM on page 4 of this reference guide.

SVM Sharing Granularity [5.6.1]

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

flags: [Table 5.14]
 CL_MEM_READ_WRITE,
 CL_MEM_WRITE_READ_ONLY,
 CL_MEM_SVM_FINE_GRAIN_BUFFER,
 CL_MEM_SVM_ATOMICS

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

Enqueuing SVM Operations [5.6.2]

`cl_int clEnqueueSVMFree (cl_command_queue command_queue, cl_uint num_svm_pointers, void *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)`

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

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

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

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

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

Program Objects

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

Create Program Objects [5.8.1]

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

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

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

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

`cl_int clRetainProgram (cl_program program)`

`cl_int clReleaseProgram (cl_program program)`

Building Program Executables [5.8.2]

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

Separate Compilation and Linking [5.8.3]

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

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

Unload the OpenCL Compiler [5.8.6]

`cl_int clUnloadPlatformCompiler (cl_platform_id platform)`

Query Program Objects [5.8.7]

`cl_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
 param_name: [Table 5.17]
 CL_PROGRAM_REFERENCE_COUNT,
 CL_PROGRAM_CONTEXT_NUM_DEVICES,
 CL_PROGRAM_CONTEXT_NUM_DEVICES,
 CL_PROGRAM_SOURCE_BINARY_SIZES_BINARIES,
 CL_PROGRAM_NUM_KERNELS_KERNEL_NAMES,
 CL_PROGRAM_IL

`cl_int clGetProgramBuildInfo (cl_program program, cl_device_id device, cl_program_build_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
 param_name: [Table 5.18]
 CL_PROGRAM_BINARY_TYPE,
 CL_PROGRAM_BUILD_GLOBAL_OPTIONS_LOG,
 CL_PROGRAM_BUILD_STATUS_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-zeros
 -cl-finite-math-only -cl-fast-relaxed-math
 -cl-unsafe-math-optimizations

Flush and Finish [5.15]
`cl_int clFlush (cl_command_queue command_queue)`
`cl_int clFinish (cl_command_queue command_queue)`

Kernel Objects

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

Create Kernel Objects [5.9.1]

`cl_kernel clCreateKernel (cl_program program, const char *kernel_name, cl_int *errcode_ret)`
`cl_int clCreateKernelsInProgram (cl_program program, cl_uint num_kernels, cl_kernel *kernels, cl_uint *num_kernels_ret)`

`cl_int clRetainKernel (cl_kernel kernel)`

`cl_int clReleaseKernel (cl_kernel kernel)`

Kernel Arguments and Queries [5.9.2-4]

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

`cl_int clSetKernelArgSVMPointer (cl_kernel kernel, cl_uint arg_index, const void *arg_value)`
`cl_int clSetKernelExecInfo (cl_kernel kernel, cl_kernel_exec_info param_name, size_t param_value_size, const void *param_value)`
 param_name: CL_KERNEL_EXEC_INFO_SVM_PTRS, CL_KERNEL_EXEC_INFO_SVM_FINE_GRAIN_SYSTEM

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

`cl_int clGetKernelInfo (cl_kernel kernel, cl_kernel_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
 param_name: [Table 5.20]
 CL_KERNEL_FUNCTION_NAME,
 CL_KERNEL_NUM_ARGS,
 CL_KERNEL_REFERENCE_COUNT,
 CL_KERNEL_ATTRIBUTES_CONTEXT_PROGRAM

`cl_int clGetKernelWorkGroupInfo (cl_kernel kernel, cl_device_id device, cl_kernel_work_group_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
 param_name: CL_KERNEL_GLOBAL_WORK_SIZE, CL_KERNEL_COMPILE_WORK_GROUP_SIZE, CL_KERNEL_COMPILE_MAX_NUM_SUB_GROUPS, CL_KERNEL_LOCAL_PRIVATE_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE [Table 5.21]

`cl_int clGetKernelArgInfo (cl_kernel kernel, cl_uint arg_index, cl_kernel_arg_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
 param_name: [Table 5.23]
 CL_KERNEL_ARG_ACCESS_ADDRESS_QUALIFIER,
 CL_KERNEL_ARG_NAME,
 CL_KERNEL_ARG_TYPE_NAME_QUALIFIER

(Continued on next page >)

Kernel Objects (continued)

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

param_name: [Table 5.22]
CL_KERNEL_LOCAL_SIZE_FOR_SUB_GROUP_COUNT,
CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_
NDRANGE,
CL_KERNEL_SUB_GROUP_COUNT_FOR_NDRANGE
```

Event Objects

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

Event Objects [5.11]

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

cl_int clSetUserEventStatus (cl_event event,
    cl_int execution_status)

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

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

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

```
cl_int clRetainEvent (cl_event event)
```

```
cl_int clReleaseEvent (cl_event event)
```

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

```
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)
```

Markers, Barriers, Waiting for Events [5.12]

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

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

Profiling Operations [5.14]

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

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

Memory Model: Shared Virtual Memory [3.3.3]

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

- **Coarse-Grained buffer SVM:** Sharing occurs at the granularity of regions of OpenCL buffer memory objects. Consistency is enforced at synchronization points and with map/unmap commands to drive updates between the host and the device. This form of SVM is similar to the use of `cl_mem` buffers, with two differences. First, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Second, concurrent access by multiple kernels on the same device is valid as long as the set of concurrently executing kernels is bounded by synchronization points. Concurrent access by multiple kernels on the same device is valid as long as the set of kernels is bounded by synchronization points. This form of SVM is similar to non-SVM use of memory; however, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Program scope global variables are treated as per-device coarse-grained SVM for addressing and sharing purposes.

- **Fine-Grained buffer SVM:** Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects. Loads and stores may be cached. This means consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.
- **Fine-Grained system SVM:** Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory. Loads and stores may be cached so consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.

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

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

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

Notes

OpenCL C Language Reference

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

Supported Data Types

The optional double scalar and vector types are supported if CL_DEVICE_DOUBLE_FP_CONFIG is not zero.

Built-in Scalar Data Types [6.1.1]

Table with 3 columns: OpenCL Type, API Type, Description. Lists scalar types like bool, char, short, int, long, float, double, half, and integer types like size_t, ptrdiff_t, etc.

Built-in Vector Data Types [6.1.2]

Table with 3 columns: OpenCL Type, API Type, Description. Lists vector types like charn, ushortn, intrn, 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 2 columns: OpenCL Type, Description. Lists image handle and array types like image2d_t, image3d_t, image2d_array_t, image1d_t, image1d_buffer_t.

Table with 3 columns: OpenCL Type, API Type, Description. Lists array types like image1d_array_t, image2d_array_t, sampler_t, queue_t, ndrange_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 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. Columns represent components from 0 to 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.xy, v.xx, v.lo.x

Table showing vector addressing equivalences for float2, float3, float4, float8, float16. Columns: v.lo, v.hi, v.odd, v.even.

Operators and Qualifiers

Operators [6.3]

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

Table of operators: +, ++, >, &&, comma; -, ==, <, ||, op=; *, !=, >=, ?:, sizeof; %, &, <=, >>, size of; /, ~, |, <<, =.

Address Space Qualifiers [6.5]

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

Function Qualifiers [6.7]

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

Attribute Qualifiers [6.11]

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

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

Use to specify special attributes of variables or structure fields.

__attribute__((aligned(alignment))), __attribute__((nosvm))

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

__attribute__((attr1)) {...}

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

__attribute__((opencl_unroll_hint(n))), __attribute__((opencl_unroll_hint))

Preprocessor Directives & Macros [6.10]

Table of preprocessor directives and macros: #pragma OPENCL_FP_CONTRACT, __FILE__, __func__, __LINE__, __OPENCL_VERSION__, CL_VERSION_1_0, CL_VERSION_1_1, CL_VERSION_1_2, CL_VERSION_2_0, __OPENCL_C_VERSION__, __ENDIAN_LITTLE__, __IMAGE_SUPPORT__, __FAST_RELAXED_MATH__, FP_FAST_FMA, FP_FAST_FMAF, FP_FAST_FMA_HALF, __kernel_exec(X, typen), __kernel_attribute__((work_group_size_hint(X, 1, 1))), __attribute__((vec_type_hint(typen)))

Blocks [6.12]

A result value type with a list of parameter types, similar to a function type. In this example:

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

```

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

Work-Item Built-in Functions [6.13.1]

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

<code>uint get_work_dim ()</code>	Number of dimensions in use
<code>size_t get_global_size (uint dimindx)</code>	Number of global work-items
<code>size_t get_global_id (uint dimindx)</code>	Global work-item ID value
<code>size_t get_local_size (uint dimindx)</code>	Number of local work-items if kernel executed with uniform work-group size
<code>size_t get_enqueued_local_size (uint dimindx)</code>	Number of local work-items
<code>size_t get_local_id (uint dimindx)</code>	Local work-item ID
<code>size_t get_num_groups (uint dimindx)</code>	Number of work-groups

<code>size_t get_group_id (uint dimindx)</code>	Work-group ID
<code>size_t get_global_offset (uint dimindx)</code>	Global offset
<code>size_t get_global_linear_id ()</code>	Work-items 1-dimensional global ID
<code>size_t get_local_linear_id ()</code>	Work-items 1-dimensional local ID
<code>uint get_sub_group_size ()</code>	Number of work-items in the subgroup
<code>uint get_max_sub_group_size ()</code>	Maximum size of a subgroup
<code>uint get_num_sub_groups ()</code>	Number of subgroups
<code>uint get_enqueued_num_sub_groups ()</code>	
<code>uint get_sub_group_id ()</code>	Sub-group ID
<code>uint get_sub_group_local_id ()</code>	Unique work-item ID

Math Built-in Functions [6.13.2] [9.4.2]

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

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

<code>T acos (T)</code>	Arc cosine
<code>T acosh (T)</code>	Inverse hyperbolic cosine
<code>T acospi (T x)</code>	$\text{acos}(x) / \pi$
<code>T asin (T)</code>	Arc sine
<code>T asinh (T)</code>	Inverse hyperbolic sine
<code>T asinpi (T x)</code>	$\text{asin}(x) / \pi$
<code>T atan (T y_over_x)</code>	Arc tangent
<code>T atan2 (T y, T x)</code>	Arc tangent of y / x
<code>T atanh (T)</code>	Hyperbolic arc tangent
<code>T atanpi (T x)</code>	$\text{atan}(x) / \pi$
<code>T atan2pi (T x, T y)</code>	$\text{atan2}(y, x) / \pi$
<code>T cbrt (T)</code>	Cube root
<code>T ceil (T)</code>	Round to integer toward + infinity
<code>T copysign (T x, T y)</code>	x with sign changed to sign of y
<code>T cos (T)</code> HN	Cosine
<code>T cosh (T)</code>	Hyperbolic cosine
<code>T cospi (T x)</code>	$\cos(\pi x)$
<code>T half_divide (T x, T y)</code> <code>T native_divide (T x, T y)</code>	x / y (T may only be float or float <i>n</i>)
<code>T erfc (T)</code>	Complementary error function
<code>T erf (T)</code>	Calculates error function of T
<code>T exp (T x)</code> HN	Exponential base e
<code>T exp2 (T)</code> HN	Exponential base 2
<code>T exp10 (T)</code> HN	Exponential base 10
<code>T expm1 (T x)</code>	$e^x - 1.0$
<code>T fabs (T)</code>	Absolute value
<code>T fdim (T x, T y)</code>	Positive difference between x and y
<code>T floor (T)</code>	Round to integer toward infinity
<code>T fma (T a, T b, T c)</code>	Multiply and add, then round
<code>T fmax (T x, T y)</code> <code>Tn fmax (Tn x, Tn y)</code>	Return y if $x < y$, otherwise it returns x

<code>T fmin (T x, T y)</code> <code>Tn fmin (Tn x, Tn y)</code>	Return y if $y < x$, otherwise it returns x
<code>T fmod (T x, T y)</code>	Modulus. Returns $x - y * \text{trunc}(x/y)$
<code>T fract (T x, T *iptr)</code>	Fractional value in x
<code>Ts frexp (T x, int *exp)</code> <code>Tn frexp (T x, intrn *exp)</code>	Extract mantissa and exponent
<code>T hypot (T x, T y)</code>	Square root of $x^2 + y^2$
<code>int[n] ilogb (T x)</code>	Return exponent as an integer value
<code>Ts ldexp (T x, int n)</code> <code>Tn ldexp (T x, intrn n)</code>	$x * 2^n$
<code>T lgamma (T x)</code> <code>Ts lgamma_r (Ts x, int *signp)</code> <code>Tn lgamma_r (Tn x, intrn *signp)</code>	Log gamma function
<code>T log (T)</code> HN	Natural logarithm
<code>T log2 (T)</code> HN	Base 2 logarithm
<code>T log10 (T)</code> HN	Base 10 logarithm
<code>T log1p (T x)</code>	$\ln(1.0 + x)$
<code>T logb (T x)</code>	Exponent of x
<code>T mad (T a, T b, T c)</code>	Approximates $a * b + c$
<code>T maxmag (T x, T y)</code>	Maximum magnitude of x and y
<code>T minmag (T x, T y)</code>	Minimum magnitude of x and y
<code>T modf (T x, T *iptr)</code>	Decompose floating-point number
<code>float[n] nan (uint[n] nancode)</code>	Quiet NaN (Return is scalar when <i>nancode</i> is scalar)
<code>half[n] nan (ushort[n] nancode)</code> <code>double[n] nan (ulong[n] nancode)</code>	Quiet NaN (Return is scalar when <i>nancode</i> is scalar)
<code>T nextafter (T x, T y)</code>	Next representable floating-point value after x in the direction of y
<code>T pow (T x, T y)</code>	Compute x to the power of y
<code>Ts pow_n (T x, int y)</code> <code>Tn pow_n (T x, intrn y)</code>	Compute x^y , where y is an integer
<code>T powr (T x, T y)</code> HN	Compute x^y , where x is ≥ 0
<code>T half_recip (T x)</code> <code>T native_recip (T x)</code>	$1 / x$ (T may only be float or float <i>n</i>)
<code>T remainder (T x, T y)</code>	Floating point remainder
<code>Ts remquo (Ts x, Ts y, int *quo)</code> <code>Tn remquo (Tn x, Tn y, intrn *quo)</code>	Remainder and quotient
<code>T rint (T)</code>	Round to nearest even integer
<code>Ts rootn (T x, int y)</code> <code>Tn rootn (T x, intrn y)</code>	Compute x to the power of $1/y$

<code>T round (T x)</code>	Integral value nearest to x rounding
<code>T rsqrt (T)</code> HN	Inverse square root
<code>T sin (T)</code> HN	Sine
<code>T sincos (T x, T *cosval)</code>	Sine and cosine of x
<code>T sinh (T)</code>	Hyperbolic sine
<code>T sinpi (T x)</code>	$\sin(\pi x)$
<code>T sqrt (T)</code> HN	Square root
<code>T tan (T)</code> HN	Tangent
<code>T tanh (T)</code>	Hyperbolic tangent
<code>T tanpi (T x)</code>	$\tan(\pi x)$
<code>T tgamma (T)</code>	Gamma function
<code>T trunc (T)</code>	Round to integer toward zero

Math Constants [6.13.2] [9.4.2]

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

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

When double precision is supported, macros ending in `_F` are available in type double by removing `_F` from the macro name, and in type half when the `cl_khr_fp16` extension is enabled by replacing `_F` with `_H`.

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

Integer Built-in Functions [6.13.3]

T is type char, charn, uchar, uchar, 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*.

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

<i>T</i> sub_sat (<i>T x</i> , <i>T y</i>)	<i>x</i> - <i>y</i> and saturates the result
<i>T</i> popcount (<i>T x</i>)	Number of non-zero bits in <i>x</i>
For <i>upsample</i> , return type is scalar when the parameters are scalar.	
short[<i>n</i>] upsample (char[<i>n</i>] <i>hi</i> , uchar[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((short)hi[<i>i</i>] << 8) lo[<i>i</i>]
ushort[<i>n</i>] upsample (uchar[<i>n</i>] <i>hi</i> , uchar[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((ushort)hi[<i>i</i>] << 8) lo[<i>i</i>]
int[<i>n</i>] upsample (short[<i>n</i>] <i>hi</i> , ushort[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((int)hi[<i>i</i>] << 16) lo[<i>i</i>]
uint[<i>n</i>] upsample (ushort[<i>n</i>] <i>hi</i> , ushort[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((uint)hi[<i>i</i>] << 16) lo[<i>i</i>]
long[<i>n</i>] upsample (int[<i>n</i>] <i>hi</i> , uint[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((long)hi[<i>i</i>] << 32) lo[<i>i</i>]
ulong[<i>n</i>] upsample (uint[<i>n</i>] <i>hi</i> , uint[<i>n</i>] <i>lo</i>)	result[<i>i</i>] = ((ulong)hi[<i>i</i>] << 32) lo[<i>i</i>]

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.

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

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*.

<i>T</i> clamp (<i>T x</i> , <i>T min</i> , <i>T max</i>) <i>Tn</i> clamp (<i>Tn x</i> , <i>Ts min</i> , <i>Ts max</i>)	Clamp <i>x</i> to range given by <i>min</i> , <i>max</i>
<i>T</i> degrees (<i>T radians</i>)	<i>radians</i> to degrees
<i>T</i> max (<i>T x</i> , <i>T y</i>) <i>Tn</i> max (<i>Tn x</i> , <i>Ts y</i>)	Max of <i>x</i> and <i>y</i>
<i>T</i> min (<i>T x</i> , <i>T y</i>) <i>Tn</i> min (<i>Tn x</i> , <i>Ts y</i>)	Min of <i>x</i> and <i>y</i>
<i>T</i> mix (<i>T x</i> , <i>T y</i> , <i>T a</i>) <i>Tn</i> mix (<i>Tn x</i> , <i>Tn y</i> , <i>Ts a</i>)	Linear blend of <i>x</i> and <i>y</i>
<i>T</i> radians (<i>T degrees</i>)	<i>degrees</i> to radians
<i>T</i> step (<i>T edge</i> , <i>T x</i>) <i>Tn</i> step (<i>Ts edge</i> , <i>Tn x</i>)	0.0 if <i>x</i> < <i>edge</i> , else 1.0
<i>T</i> smoothstep (<i>T edge0</i> , <i>T edge1</i> , <i>T x</i>) <i>Tn</i> smoothstep (<i>Ts edge0</i> , <i>Ts edge1</i> , <i>T x</i>)	Step and interpolate
<i>T</i> sign (<i>T x</i>)	Sign of <i>x</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, ulong, or ulongn. *n* is 2, 3, 4, 8, or 16. **half and halfn types require the cl_khr_fp16 extension** [9.4.5].

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

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

longn isnormal (doublen) int isnormal (half) shortn isnormal (halfn)	Test for a normal value
int isordered (float <i>x</i> , float <i>y</i>) intrn isordered (floatn <i>x</i> , floatn <i>y</i>) int isordered (double <i>x</i> , double <i>y</i>) longn isordered (doublen <i>x</i> , doublen <i>y</i>) int isordered (half <i>x</i> , half <i>y</i>) shortn isordered (halfn <i>x</i> , halfn <i>y</i>)	Test if arguments are ordered
int isunordered (float <i>x</i> , float <i>y</i>) intrn isunordered (floatn <i>x</i> , floatn <i>y</i>) int isunordered (double <i>x</i> , double <i>y</i>) longn isunordered (doublen <i>x</i> , doublen <i>y</i>) int isunordered (half <i>x</i> , half <i>y</i>) shortn isunordered (halfn <i>x</i> , halfn <i>y</i>)	Test if arguments are unordered
int signbit (float) intrn signbit (floatn) int signbit (double) longn signbit (doublen) int signbit (half) shortn signbit (halfn)	Test for sign bit
int any (<i>Ti x</i>)	1 if MSB in component of <i>x</i> is set; else 0
int all (<i>Ti x</i>)	1 if MSB in all components of <i>x</i> are set; else 0
<i>T</i> bitselect (<i>T a</i> , <i>T b</i> , <i>T c</i>) half bitselect (half <i>a</i> , half <i>b</i> , half <i>c</i>) halfn bitselect (halfn <i>a</i> , halfn <i>b</i> , halfn <i>c</i>)	Each bit of result is corresponding bit of <i>a</i> if corresponding bit of <i>c</i> is 0
<i>T</i> select (<i>T a</i> , <i>T b</i> , <i>Ti c</i>) <i>T</i> select (<i>T a</i> , <i>T b</i> , <i>Tu c</i>) halfn select (halfn <i>a</i> , halfn <i>b</i> , shortn <i>c</i>) half select (half <i>a</i> , half <i>b</i> , short <i>c</i>) halfn select (halfn <i>a</i> , halfn <i>b</i> , ushortn <i>c</i>) half select (half <i>a</i> , half <i>b</i> , ushort <i>c</i>)	For each component of a vector type, result[<i>i</i>] = if MSB of <i>c</i> [<i>i</i>] is set ? <i>b</i> [<i>i</i>] : <i>a</i> [<i>i</i>] For scalar type, result = <i>c</i> ? <i>b</i> : <i>a</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} cross (float{3,4} <i>p0</i> , float{3,4} <i>p1</i>) double{3,4} cross (double{3,4} <i>p0</i> , double{3,4} <i>p1</i>) half{3,4} cross (half{3,4} <i>p0</i> , half{3,4} <i>p1</i>)	Cross product
---	---------------

<i>Ts</i> distance (<i>T p0</i> , <i>T p1</i>)	Vector distance
<i>Ts</i> dot (<i>T p0</i> , <i>T p1</i>)	Dot product
<i>Ts</i> length (<i>T p</i>)	Vector length
<i>T</i> normalize (<i>T p</i>)	Normal vector length 1

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

Vector Data Load/Store [6.13.7] [9.4.6]

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

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

<code>void vstore_half</code> (<i>float</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half to address (<i>p</i> + <i>offset</i>)
<code>void vstore_half_R</code> (<i>float</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half to address (<i>p</i> + <i>offset</i>)
<code>void vstore_half_R</code> (<i>double</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half to address (<i>p</i> + <i>offset</i>)
<code>void vstore_halfn</code> (<i>floatn</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half vector to address (<i>p</i> + (<i>offset</i> * <i>n</i>))
<code>void vstore_halfn_R</code> (<i>floatn</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half vector to address (<i>p</i> + (<i>offset</i> * <i>n</i>))
<code>void vstore_halfn_R</code> (<i>double</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half vector to address (<i>p</i> + (<i>offset</i> * <i>n</i>))

<code>void vstore_halfn_R</code> (<i>doublen</i> <i>data</i> , <i>size_t</i> <i>offset</i> , half * <i>p</i>)	Write a half vector to address (<i>p</i> + (<i>offset</i> * <i>n</i>))
<code>floatn vloada_halfn</code> (<i>size_t</i> <i>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)).
<code>void vstorea_halfn</code> (<i>floatn</i> <i>data</i> , <i>size_t</i> <i>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)).
<code>void vstorea_halfn_R</code> (<i>floatn</i> <i>data</i> , <i>size_t</i> <i>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)).
<code>void vstorea_halfn</code> (<i>doublen</i> <i>data</i> , <i>size_t</i> <i>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)).
<code>void vstorea_halfn_R</code> (<i>doublen</i> <i>data</i> , <i>size_t</i> <i>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)).

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.

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

Atomic Functions [6.13.11]

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

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

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

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

<code>void atomic_init</code> (<i>volatile A</i> * <i>obj</i> , <i>C</i> <i>value</i>)	Initializes the atomic object pointed to by <i>obj</i> to the value <i>value</i> .
<code>void atomic_work_item_fence</code> (<i>cl_mem_fence_flags</i> <i>flags</i> , <i>memory_order</i> <i>order</i> , <i>memory_scope</i> <i>scope</i>)	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</code> (<i>volatile A</i> * <i>object</i> , <i>C</i> <i>desired</i>)	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>void atomic_store_explicit</code> (<i>volatile A</i> * <i>object</i> , <i>C</i> <i>desired</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	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</code> <i>atomic_load</i> (<i>volatile A</i> * <i>object</i>)	Atomically returns the value pointed to by <i>object</i> . Memory is affected according to the value of <i>order</i> .
<code>C</code> <i>atomic_load_explicit</i> (<i>volatile A</i> * <i>object</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	Atomically returns the value pointed to by <i>object</i> . Memory is affected according to the value of <i>order</i> .
<code>C</code> <i>atomic_exchange</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> <i>desired</i>)	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>C</code> <i>atomic_exchange_explicit</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> <i>desired</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	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</code> <i>atomic_compare_exchange_strong</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> * <i>expected</i> , <i>C</i> <i>desired</i>)	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> . These operations are atomic read-modify-write operations.
<code>bool</code> <i>atomic_compare_exchange_strong_explicit</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> * <i>expected</i> , <i>C</i> <i>desired</i> , <i>memory_order</i> <i>success</i> , <i>memory_order</i> <i>failure</i> [, <i>memory_scope</i> <i>scope</i>])	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> . These operations are atomic read-modify-write operations.
<code>bool</code> <i>atomic_compare_exchange_weak</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> * <i>expected</i> , <i>C</i> <i>desired</i>)	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> . These operations are atomic read-modify-write operations.
<code>bool</code> <i>atomic_compare_exchange_weak_explicit</i> (<i>volatile A</i> * <i>object</i> , <i>C</i> * <i>expected</i> , <i>C</i> <i>desired</i> , <i>memory_order</i> <i>success</i> , <i>memory_order</i> <i>failure</i> [, <i>memory_scope</i> <i>scope</i>])	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> . These operations are atomic read-modify-write operations.
<code>C</code> <i>atomic_fetch_<key></i> (<i>volatile A</i> * <i>object</i> , <i>M</i> <i>operand</i>)	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> .
<code>C</code> <i>atomic_fetch_<key>_explicit</i> (<i>volatile A</i> * <i>object</i> , <i>M</i> <i>operand</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	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> .

Async Copies and Prefetch [6.13.10] [9.4.7]

T is type char, charn, uchar, uchar_n, short, shortn, ushort, ushortn, int, intrn, uint, uintn, long, longn, ulong, ulongn, float, floatn, optionally double or double_n, or half or halfn if the `cl_khr_fp16` extension is enabled.

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

<code>bool</code> <i>atomic_flag_test_and_set</i> (<i>volatile atomic_flag</i> * <i>object</i>)	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>bool</code> <i>atomic_flag_test_and_set_explicit</i> (<i>volatile atomic_flag</i> * <i>object</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	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</code> <i>atomic_flag_clear</i> (<i>volatile atomic_flag</i> * <i>object</i>)	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> .
<code>void</code> <i>atomic_flag_clear_explicit</i> (<i>volatile atomic_flag</i> * <i>object</i> , <i>memory_order</i> <i>order</i> [, <i>memory_scope</i> <i>scope</i>])	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> .

Values for key for atomic_fetch and modify functions

key	op	computation	key	op	computation
add	+	addition	and	&	bitwise and
sub	-	subtraction	min	min	compute min
or		bitwise inclusive or	max	max	compute max
xor	^	bitwise exclusive or			

Atomic Types and Enum Constants

memory_scope_sub_group requires the `cl_khr_subgroups` extension.

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

Atomic integer and floating-point types

† indicates types supported by a limited subset of atomic operations
 ‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture.
 § indicates types supported only if both 64-bit extensions are supported.

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

Atomic Macros

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

Address Space Qualifier Functions [6.13.9]

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

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

printf Function [6.13.13]

Writes output to an implementation-defined stream.

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

printf output synchronization

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

printf format string

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

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

Examples:

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

float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);
printf("f4 = %2.2v4f\n", f);
Output: f4 = 1.00,2.00,3.00,4.00

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

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

Miscellaneous Vector Functions [6.13.12]

Tm and Tn are type charn, uchar, shortn, ushortn, intr, 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, ushort, uint, or ulong.

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

Workgroup Functions [6.13.15] [9.17.3.4]

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

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

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

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

```
T work_group_reduce_<op> (Tx)
T sub_group_reduce_<op> (Tx)
```

Broadcast the value of a to all work-items in the work-group or sub_group. local_id must be the same value for all workitems in the work-group. n may be 2 or 3.

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

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

```
T work_group_scan_exclusive_<op> (Tx)
T work_group_scan_inclusive_<op> (Tx)
T sub_group_scan_exclusive_<op> (Tx)
T sub_group_scan_inclusive_<op> (Tx)
```

Pipe Built-in Functions [6.13.16.2-4]

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

int read_pipe (__read_only pipe T p, T *ptr)	Read packet from p into ptr.
int read_pipe (__read_only pipe T p, reserve_id_t reserve_id, uint index, T *ptr)	Read packet from reserved area of the pipe reserve_id and index into ptr.
int write_pipe (__write_only pipe T p, const T *ptr)	Write packet specified by ptr to p.
int write_pipe (__write_only pipe T p, reserve_id_t reserve_id, uint index, const T *ptr)	Write packet specified by ptr to reserved area reserve_id and index.

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

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

Enqueuing and Kernel Query Built-in Functions [6.13.17] [9.17.3.6]

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

```
int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags,
const ndrange_t ndrange, void (^block)(void))
int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags,
const ndrange_t ndrange, uint num_events_in_wait_list,
const clk_event_t *event_wait_list, clk_event_t *event_ret,
void (^block)(void))
int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags,
const ndrange_t ndrange,
void (^block)(local void *, ...), uint size0, ...)
int enqueue_kernel (queue_t queue, kernel_enqueue_flags_t flags,
const ndrange_t ndrange,
uint num_events_in_wait_list, const clk_event_t *event_wait_list,
clk_event_t *event_ret, void (^block)(local void *, ...), uint size0, ...)
```

Allows a work-item to enqueue a block for execution to queue. Work-items can enqueue multiple blocks to a device queue(s). flags may be one of CLK_ENQUEUE_FLAGS {NO_WAIT, WAIT_KERNEL, WAIT_WORK_GROUP}

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

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> : CL_COMPLETE 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 value.

Helper Built-in Functions [6.13.17.9]

<code>queue_t get_default_queue (void)</code>	Default queue or CLK_NULL_QUEUE
<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 2.1 and C Language 2.0 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 [5.3.3-4]

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

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

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

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

Copy Between Image, Buffer Objects [5.3.5]

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

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

Map and Unmap Image Objects [5.3.6]

`void * clEnqueueMapImage (cl_command_queue command_queue, cl_mem image, cl_bool blocking_map, cl_map_flags map_flags, const size_t *origin, const size_t *region, size_t *image_row_pitch, size_t *image_slice_pitch, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)`
map_flags: CL_MAP_{READ, WRITE}, CL_MAP_WRITE_INVALIDATE_REGION

Query Image Objects [5.3.7]

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

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

Image Formats [5.3.1.1]

Supported image formats: `image_channel_order` with `image_channel_data_type`.

Built-in support [Table 5.8]

CL_R (read or write): CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}

CL_DEPTH (read or write): CL_FLOAT, CL_UNORM_INT16

CL_DEPTH_STENCIL (read only): CL_FLOAT, CL_UNORM_INT24
(Requires the extension `cl_khr_gl_depth_images`)

CL_RG (read or write): CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}

CL_RGBA (read or write): CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}

CL_BGRA (read or write): CL_UNORM_INT8

CL_sRGBA (read only): CL_UNORM_INT8
(Requires the extension `cl_khr_srgb_image_writes`)

Optional support [Table 5.6]

CL_R, CL_A (read and write): CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}

CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8,16}

CL_DEPTH_STENCIL: Only used if extension `cl_khr_gl_depth_images` is enabled and channel data type = CL_UNORM_INT24 or CL_FLOAT

CL_LUMINANCE: CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16}

CL_RG, CL_RA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}

CL_RGB: CL_UNORM_SHORT_{555,565}, CL_UNORM_INT_{101010}

CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

CL_BGRA: CL_{SIGNED, UNSIGNED}_INT8, CL_SNORM_INT8

Notes

Image Read and Write Functions [6.13.14]

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

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

Read and write functions for 2D images

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

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

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

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

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

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

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

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

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

float4 read_imagef (aQual image2d_t image, int2 coord)

int4 read_imagei (aQual image2d_t image, int2 coord)

uint4 read_imageui (aQual image2d_t image, int2 coord)

float4 read_imagef (aQual image2d_array_t image, int4 coord)

int4 read_imagei (aQual image2d_array_t image, int4 coord)

uint4 read_imageui (aQual image2d_array_t image, int4 coord)

float read_imagef (aQual image2d_depth_t image, int2 coord)

float read_imagef (aQual image2d_array_depth_t image,
                  int4 coord)

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

half4 read_imageh (aQual image2d_t image, int2 coord)

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

half4 read_imageh (aQual image2d_array_t image,
                  int4 coord)

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

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

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

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

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

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

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

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

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

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

Read and write functions for 1D images

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

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

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

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

float4 read_imagef (read_only image1d_array_t image,
                   sampler_t sampler, {int2, float2} coord)

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

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

float4 read_imagef (aQual image1d_t image, int coord)

float4 read_imagef (aQual image1d_buffer_t image, int coord)

int4 read_imagei (aQual image1d_t image, int coord)

uint4 read_imageui (aQual image1d_t image, int coord)

int4 read_imagei (aQual image1d_buffer_t image, int coord)

uint4 read_imageui (aQual image1d_buffer_t image, int coord)

float4 read_imagef (aQual image1d_array_t image, int2 coord)

int4 read_imagei (aQual image1d_array_t image, int2 coord)

uint4 read_imageui (aQual image1d_array_t image, int2 coord)

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

half4 read_imageh (aQual image1d_t image, int coord)

half4 read_imageh (read_only image1d_array_t image,
                  sampler_t sampler, {int2, float2} coord)

half4 read_imageh (aQual image1d_array_t image, int2 coord)

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

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

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

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

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

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

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

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

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

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

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

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

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

Read and write functions for 3D images

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

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

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

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

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

float4 read_imagef (aQual image3d_t image, int4 coord)

int4 read_imagei (aQual image3d_t image, int4 coord)

uint4 read_imageui (aQual image3d_t image, int4 coord)

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

half4 read_imageh (aQual image3d_t image, int4 coord)

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

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

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

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

Extended mipmap read and write functions [9.17.2.1]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(Continued on next page >)

Image Read and Write (continued)

Extended mipmap read and write functions (cont'd)

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

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

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

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

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

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

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

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

float read_imagef(read_only image2d_array_[depth]_t image,
  sampler_t sampler, float4 coord, float lod)

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

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

float read_imagef(
  read_only image2d_array_[depth]_t image,
  sampler_t sampler, float4 coord, float2 gradient_x,
  float2 gradient_y)

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

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

void write_imagef(aQual image2d_[depth]_t image,
  int2 coord, int lod, float4 color)

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

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

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

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

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

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

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

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

```
void write_imagef(aQual image2d_array_[depth]_t image,
  int4 coord, int lod, float4 color)

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

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

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

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

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

Extended multi-sample image read functions [9.12.3]

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

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

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

float4 read_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)
```

Image Query Functions [6.13.14.5] [9.12]

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

Query image width, height, and depth in pixels

```
int get_image_width(aQual image{1,2,3}d_t image)
int get_image_width(aQual image1d_buffer_t image)
int get_image_width(aQual image{1,2}d_array_t image)
int get_image_width(
  aQual image2d_[array]_depth_t image)
int get_image_width(aQual image2d_[array]_msaa_t image)
int get_image_width(
  aQual image2d_[array]_msaa_depth_t image)

int get_image_height(aQual image{2,3}d_t image)
int get_image_height(aQual image2d_array_t image)
int get_image_height(
  aQual image2d_[array]_depth_t image)
int get_image_height(
  aQual image2d_[array]_msaa_t image)
int get_image_height(
  aQual image2d_[array]_msaa_depth_t image)

int get_image_depth(image3d_t image)
```

Query image array size

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

Query image dimensions

```
int2 get_image_dim(aQual image2d_t image)
int2 get_image_dim(aQual image2d_array_t image)
int4 get_image_dim(aQual image3d_t image)

int2 get_image_dim(aQual image2d_[array]_depth_t image)
int2 get_image_dim(aQual image2d_[array]_msaa_t image)
int2 get_image_dim(
  aQual image2d_[array]_msaa_depth_t image)
```

Query image Channel data type and order

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

int get_image_channel_data_type(
  aQual image1d_buffer_t image)

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

int get_image_channel_data_type(aQual
  image2d_[array]_depth_t image)

int get_image_channel_data_type(
  aQual image2d_[array]_msaa_t image)

int get_image_channel_data_type(
  aQual image2d_[array]_msaa_depth_t image)

int get_image_channel_order(aQual image{1,2,3}d_t image)
int get_image_channel_order(
  aQual image1d_buffer_t image)

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

int get_image_channel_order(
  aQual image2d_[array]_depth_t image)

int get_image_channel_order(
  aQual image2d_[array]_msaa_t image)

int get_image_channel_order(
  aQual image2d_[array]_msaa_depth_t image)
```

Extended query functions [9.18.2.1]

These functions require the `cl_khr_mipmap_image` extension.

```
int get_image_num_mip_levels(aQual image1d_t image)
int get_image_num_mip_levels(
  aQual image2d_[depth]_t image)

int get_image_num_mip_levels(aQual image3d_t image)
int get_image_num_mip_levels(
  aQual image1d_array_t image)

int get_image_num_mip_levels(
  aQual image2d_array_[depth]_t image)

int get_image_num_samples(
  aQual image2d_[array]_msaa_t image)

int get_image_num_samples(
  aQual image2d_[array]_msaa_depth_t image)
```

Access Qualifiers [6.6]

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

```
_read_only, read_only
_write_only, write_only
```

Sampler Objects [5.7]

Items in blue require the `cl_khr_mipmap_image` extension.

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

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

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

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

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

Sampler Declaration Fields [6.13.14.1]

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

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

normalized-mode:
  CLK_NORMALIZED_COORDS_{TRUE, FALSE}

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

filter-mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```

OpenCL Extensions Reference

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

Using OpenCL Extensions [9]

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

To control an extension: #pragma OPENCL EXTENSION *extension_name* : {enable | disable}

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

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

<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_device_enqueue_local_arg_types</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_priority_hints</code>
<code>cl_khr_srgb_image_writes</code>
<code>cl_khr_spir</code>
<code>cl_khr_subgroups</code>
<code>cl_khr_terminate_context</code>
<code>cl_khr_throttle_hints</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`)

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_object_type *gl_object_type,
    GLuint *gl_object_name)
```

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

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

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

Share Objects [9.6.6]

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

CL Event Objects > GL Sync Objects [9.7.4]

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

Requires the `cl_khr_gl_event` extension.

DX9 Media Surface Sharing [9.9]

The header file is `cl_dx9_media_sharing.h`. Enable the extension `cl_khr_dx9_media_sharing`.

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

media_adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR
media_adapter_set: CL_{ALL, PREFERRED}_DEVICES_FOR_DX9_MEDIA_ADAPTER_KHR

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

```

adapter_type: CL_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR

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

Direct3D 11 Sharing [9.10.7.3 - 9.10.7.6]

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

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

EGL Interoperability [9.18, 9.19]

Create CL Image Objects from EGL

These functions require the extension `cl_khr_egl_image`.

```
cl_mem clCreateFromEGLImageKHR (
    cl_context context, CLEGLDisplayKHR display,
    CLEGLImageKHR image, cl_mem_flags flags,
    const cl_egl_image_properties_khr *properties,
    cl_int *errcode_ret)
```

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

Create CL Event Objects from EGL

This function requires the extension `cl_khr_egl_event`.

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

Example of Enqueuing Kernels

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

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

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

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

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

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

A Complete Example [6.13.17.3]

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

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

    if (get_global_id(0) == 0)
    {
        enqueue_kernel(q,
            CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
            ndrange_1d(1),
            ^{evaluate_dp_work_A(q, ...)});
    }
}

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

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

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

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


OpenCL Reference Card Index

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

A					
Access Qualifiers	12	clEnqueueReadBuffer[Rect]	2	clRetainProgram	3
Address Space Qualifiers	5	clEnqueueReadImage	10	clRetainSampler	12
Address Space Qualifier Functions	9	clEnqueueReleaseD3D10ObjectsKHR	13	clSetDefaultDeviceCommandQueue	1
Architecture Diagram	2	clEnqueueReleaseD3D11ObjectsKHR	13	clSetEventCallback	4
Async Copies and Prefetch	8	clEnqueueReleaseDX9MediaSurfacesKHR	13	clSetKernelArg	3
Atomic Functions	8	clEnqueueReleaseEGLObjectsKHR	13	clSetKernelArgSVMPointer	3
Attribute Qualifiers	5	clEnqueueReleaseGLObjects	12	clSetKernelExecInfo	3
		clEnqueueSVM[Un]Map	3	clSetMemObjectDestructorCallback	2
B		clEnqueueSVMFree	3	clSetUserEventStatus	4
Barriers	4	clEnqueueSVMMem{cpy, Fill}	3	clSVMAlloc	3
Blocks	6	clEnqueueUnmapMemObject	2	clSVMFree	3
Buffer Objects	2	clEnqueueWriteBuffer[Rect]	2	clTerminateContextKHR	1
		clEnqueueWriteImage	10	clUnloadPlatformCompiler	3
C		clFinish	3	clWaitForEvents	4
cl_khr_*	13	clFlush	3	Code Examples	14
clBuildProgram	3	clGetCommandQueueInfo	1	Command Queues	1
clCloneKernel	3	clGetContextInfo	1	Common Built-in Functions	7
clCompileProgram	3	clGetDeviceIDs	1	Compiler Options	3
clCreateBuffer	2	clGetDeviceIDsFromD3D10KHR	13	Contexts	1
clCreateCommandQueueWithProperties	1	clGetDeviceIDsFromD3D11KHR	13	Conversions and Type Casting	2
clCreateContext	1	clGetDeviceIDsFromDX9MediaAdapterKHR	13	Copy Between Image, Buffer Objects	10
clCreateContextFromType	1	clGetDeviceInfo	1		
clCreateEventFromEGLsyncKHR	13	clGetDeviceAndHostTimer	1	D	
clCreateEventFromGLsyncKHR	13	clGetEventInfo	4	Data Types	5
clCreateFromD3D10BufferKHR	13	clGetEventProfilingInfo	4	Debugging options	3
clCreateFromD3D10Texture2DKHR	13	clGetExtensionFunctionAddressForPlatform	1	Device Architecture Diagram	2
clCreateFromD3D10Texture3DKHR	13	clGetGLContextInfoKHR	13	Direct3D 10 Sharing	13
clCreateFromD3D11BufferKHR	13	clGetGLObjectInfo	13	Direct3D 11 Sharing	13
clCreateFromD3D11Texture2DKHR	13	clGetGLTextureInfo	13	DX9 Media Surface Sharing	13
clCreateFromD3D11Texture3DKHR	13	clGetHostTimer	1		
clCreateFromDX9MediaSurfaceKHR	13	clGetImageInfo	10	E - F	
clCreateFromEGLImageKHR	13	clGetKernelArgInfo	3	EGL Interoperability	13
clCreateFromGLBuffer	13	clGetKernelInfo	3	Enqueuing & Kernel Query Built-in Functions	9
clCreateFromGLRenderbuffer	13	clGetKernelSubGroupInfo	4	Enqueuing Kernels Code Examples	14
clCreateFromGLTexture	13	clGetKernelWorkGroupInfo	3	Event Built-in Functions	10
clCreateImage	10	clGetMemObjectInfo	2	Event Objects	4
clCreateKernel	3	clGetPipeInfo	2	Execute Kernels	4
clCreateKernelsInProgram	3	clGetPlatformIDs	1	Extension Function Pointers	1
clCreatePipe	2	clGetPlatformInfo	1	Extensions	13
clCreateProgramWithBinary	3	clGetPlatformInfo	1	Fence Functions	8
clCreateProgramWithBuiltInKernels	3	clGetProgramBuildInfo	3	Flush and Finish	3
clCreateProgramWithIL	3	clGetProgramInfo	3	Function Qualifiers	5
clCreateProgramWithSource	3	clGetSamplerInfo	12		
clCreateSamplerWithProperties	12	clGetSupportedImageFormats	10	G - H	
clCreateSubBuffer	2	clIcdGetPlatformIDsKHR	1	Geometric Built-in Functions	7
clCreateSubDevices	1	clLinkProgram	3	Helper Built-in Functions	10
clCreateUserEvent	4	clReleaseCommandQueue	1		
clEnqueueAcquireD3D10ObjectsKHR	13	clReleaseContext	1	I	
clEnqueueAcquireD3D11ObjectsKHR	13	clReleaseDevice	1	Image Formats	10
clEnqueueAcquireDX9MediaSurfacesKHR	13	clReleaseEvent	4	Image Objects	10
clEnqueueAcquireEGLObjectsKHR	13	clReleaseKernel	3	Image Query Functions	12
clEnqueueAcquireGLObjects	13	clReleaseMemObject	2	Image Read and Write Functions	11-12
clEnqueueBarrierWithWaitList	4	clReleaseProgram	3	Integer Built-in Functions	7
clEnqueueCopyBuffer[Rect]	2	clReleaseSampler	12		
clEnqueueCopyBufferToImage	10	clRetainCommandQueue	1	K	
clEnqueueCopyImage	10	clRetainContext	1	Kernel Arguments and Queries	3
clEnqueueCopyImageToBuffer	10	clRetainDevice	1	Kernel Objects	3
clEnqueueFillBuffer	2	clRetainEvent	4	Kernel Query Built-in Functions	9
clEnqueueFillImage	10	clRetainKernel	3		
clEnqueueMapBuffer	2	clRetainMemObject	2	L	
clEnqueueMapImage	10			Library linking options	3
clEnqueueMarkerWithWaitList	4			Linker Options	3
clEnqueueMigrateMemObjects	2				
clEnqueueNativeKernel	4			M	
clEnqueueNDRangeKernel	4			Map and Unmap Image Objects	10
				Map Buffer Objects	2
				Markers, Barriers, Waiting for Events	4
				Math Built-in Functions	6
				Math Constants	6
				Memory Fence Functions	8
				Memory Objects	2
				Migrate Memory Objects	2
				O	
				OpenCL Class Diagram	2
				OpenCL Extensions	13
				OpenGL Sharing	13
				Operators	5
				Optimization options	3
				P	
				Partitioning a Device	1
				Pipe Built-in Functions	9
				Pipes	2
				Prefetch	8
				Preprocessor	3
				Preprocessor Directives & Macros	5
				printf Function	9
				Profiling Operations	4
				Program linking options	3
				Program Objects	3
				Q	
				Qualifiers	5
				Query Image Objects	10
				Query Image Functions	12
				Query List of Supported Image Formats	10
				Query Memory Object	2
				Query Program Objects	3
				Querying Platform Info & Devices	1
				R	
				Read, Write, Copy Buffer Objects	2
				Read, Write, Copy, Fill Image Objects	10
				Relational Built-in Functions	7
				S - T	
				Sampler Objects, Declaration Fields	12
				Scalar Data Types	5
				Separate Compilation and Linking	3
				Shared Virtual Memory	3,4
				SPIR binary options	3
				Supported Data Types	5
				SVM Sharing Granularity	3
				Synchronization & Memory Fence Functions	8
				Type Casting Examples	2
				Types	5
				U - V	
				Unload the OpenCL Compiler	3
				Unroll attribute qualifiers	5
				Vector Component Addressing	5
				Vector Data Load/Store	8
				Vector Functions	9
				Vector Data Types	5
				W	
				Waiting for Events	4
				Warning request/suppress	3
				Workgroup Functions	9
				Work-Item Built-in Functions	6



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.

Khronos Group and the Khronos Group logo are registered trademarks of the Khronos Group, and the Khronos OpenCL logo is a trademark of Apple Inc. and is used under license by Khronos.